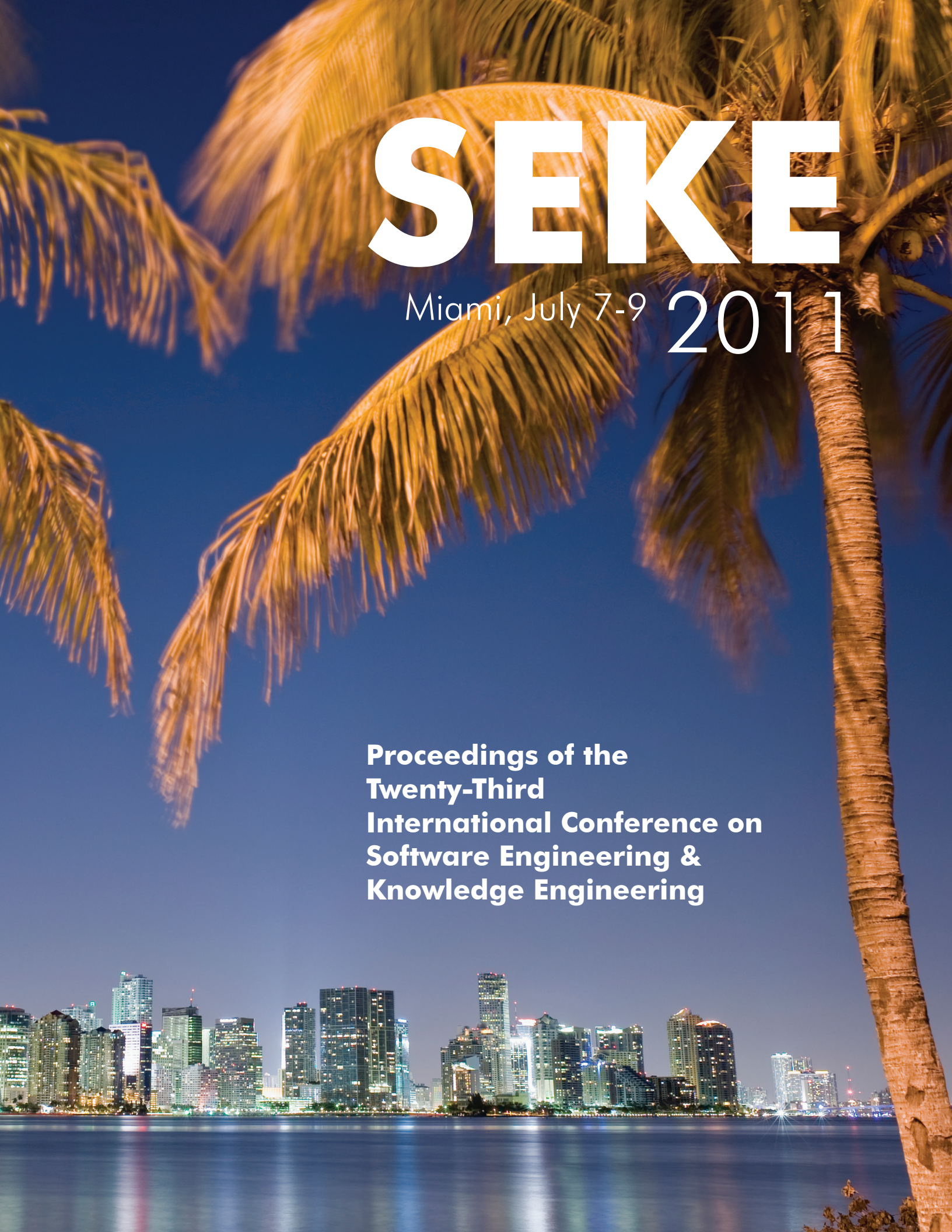

SEKE
Miami, July 7-9 2011
Proceedings of the
Twenty-Third
International Conference on
Software Engineering &
Knowledge Engineering

PROCEEDINGS

SEKE 2011

The 23rd International Conference on Software Engineering & Knowledge Engineering

Sponsored by

Knowledge Systems Institute Graduate School, USA

Technical Program

July 7-9, 2011

Eden Roc Renaissance Miami Beach, Florida, USA

Organized by

Knowledge Systems Institute Graduate School

ISBN-10: 1-891706-29-2 (paper)
ISBN-13: 978-1-891706-29-5

Additional Copies can be ordered from:
Knowledge Systems Institute Graduate School
3420 Main Street
Skokie, IL 60076, USA
Tel:+1-847-679-3135
Fax:+1-847-679-3166
Email:office@ksi.edu
http://www.ksi.edu

Proceedings preparation, editing and printing are sponsored by
Knowledge Systems Institute Graduate School

Printed by Knowledge Systems Institute Graduate School

SEKE 2011 Foreword

On behalf of the Program Committee Co-Chairs and the Program Committee of the 2011 International Conference on Software Engineering and Knowledge Engineering (SEKE2011), we sincerely welcome you to contribute and attend SEKE-2011 in Miami Beach, Florida, USA.

The International Conference on Software Engineering and Knowledge Engineering has entered its 23rd year. In the past twenty-two years, the International Conference on Software Engineering and Knowledge Engineering has provided a unique and important forum for academic and industrial researchers and practitioners to exchange research ideas, results and application experience in software engineering and knowledge engineering.

This year's technical program is prepared and organized by a great team of Program Co-Chairs, who are listed below.

Program Co-Chairs:
Du Zhang, California State University Sacramento, USA
Marek Reformat, University of Alberta, Canada
Swapna Gokhale, University of Connecticut, USA

It has been my great honor and pleasure to serve as the SEKE2011 Program Committee Chair and work with this great team and the program committee members to prepare a rich and solid technical program as well as the high quality conference proceedings. The published conference proceedings contain the papers accepted and selected for presentation at SEKE2011 based on a rigorous review process. I hope it will serve as a valuable reference for the research community in the coming years.

We received 224 submissions from 33 countries. The acceptance rate for full papers is 33%, and that for short papers is 29%. This year, 150 authors from 30 countries (Australia, Austria, Brazil, Canada, China, Cyprus, Denmark, Egypt, England, France, Germany, India, Italy, Japan, Korea, Malaysia, Mexico, New Zealand, Peru, Portugal, Scotland, Senegal, Singapore, Spain, Sweden, Switzerland, Tunisia, Turkey, UK, and USA) will present 75 regular papers, 65 short papers, and 5 demo posters at the conference.

As the Program Chair for this Conference, I am very grateful to have this opportunity to work with three distinguished SEKE2011 Program Committee Co-Chairs and the committed program committee members. Their excellent support and prompt review efforts led to the successful organization of SEKE2011 technical program. I want to extend my sincere and deepest thanks to Dr. Shihong Huang and Dr. Xiaoying Bai as the Publicity Co-Chairs, Dr. Hironori Washizaki as the Asia Liaison, Dr. Jose Carlos Maldonado as the South America Liaison. My appreciation also goes to the keynote speakers and special address presenter for sharing their visions, insights, and experiences with the conference attendee about emergent technologies and trends, research topics and issues in both academic research and industry applications. Moreover, I like to express my appreciation to the organizers of the new poster/demo session, namely, Dr. Farshad Samimi, Dr. Ming Zhao, and Dr. Raul Garcia Castro for their great contributions. In addition, I would like to thank Dr. S. K. Chang, the Steering Committee Chair, and Dr. Jerry Gao, the Conference Chair, for their excellent guidance throughout the conference preparation process. Last but not the least, I owe a special gratitude to the staff members of Knowledge Systems Institute, specially, to David Huang and Rachel Lu, for their great efforts and timely support.

Finally, I truly hope you will enjoy the technical program of SEKE2011, have productive discussion, great presentation and networking. Of course, I sincerely hope you will all explore and enjoy the sunshine state of Florida and various attractions in Miami Beach area.

Masoud Sadjadi
SEKE2011 Program Chair

The 23rd International Conference on Software Engineering & Knowledge Engineering (SEKE 2011)

July 7-9, 2011
Eden Roc Renaissance Miami Beach, Florida, USA

Conference Organization

Steering Committee Chair

Shi-Kuo Chang, *University of Pittsburgh, USA*

Steering Committee

Vic Basili, *University of Maryland, USA*
Bruce Buchanan, *University of Pittsburgh, USA*
C. V. Ramamoorthy, *University of California, Berkeley, USA*

Advisory Committee

Natalia Juristo, *Madrid Technological University, Spain*
Taghi Khoshgoftaar, *Florida Atlantic University, USA*
Guenther Ruhe, *University of Calgary, Canada*

Conference Chair

Jerry Gao, *San Jose State University, USA*

Program Chair

S. Masoud Sadjadi, *Florida International University, USA*

Program Co-Chairs

Marek Reformat, *University of Alberta, Canada*
Du Zhang, *California State University Sacramento, USA*
Swapna Gokhale, *University of Connecticut, USA*

Program Committee

Alain Abran, *L'ecole de technologie superieure, Canada*
Silvia Teresita Acuna, *Universidad Autonoma de Madrid, Spain*
Taiseera Albalushi, *Sultan Qaboos University, Oman*
Edward Allen, *Mississippi State University, USA*
Rosa Badia, *Barcelona Supercomputing Center, Spain*
Doo-hwan Bae, *Korea Advanced Institute of Science and Technology, Korea*
Ebrahim Bagheri, *National Research Council Canada, Canada*
Rami Bahsoon, *University of Birmingham, United Kingdom*
Xiaoying Bai, *Tsinghua University, China*
Purushotham Bangalore, *University of Alabama at Birmingham, USA*
Benoit Baudry, *Institut de Recherche en Informatique et Systemes Aleatoires, France*
Fevzi Belli, *Univ. Paderborn, Germany*
Ateet Bhalla, *NRI Institute of Information Science and Technology, India*
Alessandro Bianchi, *Department of Informatics - University of Bari, Italy*
Karun N. Biyani, *Michigan State University, USA*
Kai-yuan Cai, *Beijing University of Aeronautics and Astronautics, China*
Borzoo Bonakdarpour, *University of Waterloo, Canada*
Jean-michel Bruel, *IRIT, Universite Paul Sabatier, France*
Gerardo Canfora, *Universita del Sannio, Italy*
Jaelson Castro, *Universidade Federal de Pernambuco - UFPE, Brazil*
Raul Garcia Castro, *Universidad Politecnica de Madrid, Spain*
Cagatay Catal, *The Scientific & Technological Research Council of Turkey, Turkey*
Christine Chan, *University of Regina, Canada*
Keith Chan, *The Hong Kong Polytechnic University, Hong Kong*
Kuang-nan Chang, *Eastern Kentucky University, USA*
Ned Chapin, *InfoSci Inc., USA*
Gerardo Antonio Castanon Avila, *Tecnologico de Monterrey, Mexico*
Shu-ching Chen, *Florida International University, USA*
Zhenyu Chen, *Nanjing University, China*
Yung-pin Cheng, *Dept. of CS, National Taiwan Normal Univ., Taiwan*
Stelvio Cimato, *The University of Milan, Italy*
Peter Clarke, *Florida International University, USA*
Esteban Clua, *Universidade Federal Fluminense, Brasil*
Nelly Condori-fernandez, *Valencia University of Valencia, Spain*
Julita Corbalan, *Barcelona Supercomputing Center, Spain*
Fabio M. Costa, *Instituto de Informatica, Brasil*
Maria Francesca Costabile, *University of Bari, Italy*
Karl Cox, *University of Brighton, United Kingdom*
Jose Luis Cuadrado, *University of Alcala, Spain*

Juan J. Cuadrado-gallego, *University of Alcala, Spain*
Dilma Da Silva, *IBM, USA*
Ernesto Damiani, *The University of Milan, Italy*
Jose Luis De La Vara, *Universidad Politecnica de Valencia, Spain*
Deepak Dhungana, Lero, *The Irish Software Engineering Research Centre., Ireland*
Massimiliano Di Penta, *University of Sannio, Italy*
Scott Dick, *University of Alberta, Canada*
Jin Song Dong, *NUS, Singapore*
Jing Dong, *University of Texas at Dallas, USA*
Dirk Draheim, *University of Innsbruck, Austria*
Weichang Du, *University of New Brunswick, Canada*
Philippe Dugerdil, *HEG - Univ. of Applied Sciences, Switzerland*
Hector Duran, *Centro Universitario de Ciencias Economico Administrativas, Mexico*
Christof Ebert, *Vector Consulting Services, Germany*
Ali Ebnenasir, *Michigan Technological University, USA*
Raimund Ege, *NIU, USA*
Magdalini Eirinaki, *Computer Engineering Dept, San Jose State University, USA*
Faezeh Ensan, *University of New Brunswick, Canada*
Onyeka Ezenwoye, *South Dakota State University, USA*
Davide Falessi, *University of Rome, TorVergata, Italy*
Behrouz Far, *University of Calgary, Canada*
Robert Feldt, *Chalmers University of Technology, Sweden*
Eduardo B. Fernandez, *Florida Atlantic University, USA*
Marian Fernandez De Sevilla, *University of Alcala, Spain*
Gerardo Fernandez Escribano, *Universidad de Castilla-La Mancha, Spain*
Scott D. Fleming, *Oregon State University, USA*
Liana Fong, *IBM, USA*
Renata Fortes, *Instituto de Ciencias Matematicas e de Computacao - USP, Brazil*
Fulvio Frati, *The University of Milan, Italy*
Jerry Gao, *San Jose State University, USA*
Kehan Gao, *Eastern Connecticut State University, USA*
Felix Garcia, *University of Castilla-La Mancha, Spain*
Ignacio Garcia Rodriguez De Guzman, *University of Castilla-La Mancha, Spain*
Itana Gimenes, *Universidade Estadual de Maringa, Brazil*
Swapna Gokhale, *Univ. of Connecticut, USA*
Wolfgang Golubski, *Zwickau University of Applied Sciences, Germany*
Jeff Gray, *University of Alabama, USA*
Desmond Greer, *Queen's University Belfast, United Kingdom*
Eric Gregoire, *Universite d'Artois, France*
Christiane Gresse Von Wangenheim, *UFSC - Federal University of Santa Catarina, Brazil*
Xudong He, *Florida International University, USA*
Miguel Herranz, *University of Alcala, Spain*
Howard Ho, *IBM, USA*
Mong Fong Horng, *National Kaohsiung University of Applied Sciences, Taiwan*
Shihong Huang, *Florida Atlantic University, USA*
Clinton Jeffery, *University of Idaho, USA*
Jason Jung, *Yeungnam University, South Korea*
Natalia Juristo, *Universidad Politecnica de Madrid, Spain*
Selim Kalayci, *Florida International University, USA*
Eric Kasten, *Michigan State University, USA*

Taghi Khoshgoftaar, *Florida Atlantic University, USA*
Nicholas Kraft, *The University of Alabama, USA*
Sandeep Kulkarni, *Michigan State University, USA*
Vinay Kulkarni, *Tata Consultancy Services, India*
Gihwon Kwon, *Kyonggi University, South Korea*
Konstantin Laufer, *Loyola University Chicago , USA*
Juan Carlos Lavariega Jarquin, *Institute de Monterrey, Mexico*
Jeff Lei, *University of Texas at Arlington, USA*
Bixin Li, *School of Computer Science and Engineering, Southeast University, China*
Ming Li, *Nanjing University, China*
Tao Li, *Florida International University, USA*
Chien-hung Liu, *National Taipei University of Technology, Taiwan*
Shih-hsi Liu, *California State University, Fresno, USA*
Xiaodong Liu, *Edinburgh Napier University, United Kingdom*
Yan Liu, *Pacific Northwest National Laboratory, USA*
Yi Liu, *GCSU, USA*
Hakim Lounis, *UQAM, Canada*
Joan Lu, *University of Huddersfield, United Kingdom*
Heiko Ludwig, *IBM Research, USA*
Jose Carlos Maldonado, *ICMC-USP, Brazil*
Antonio Mana, *University of Malaga, Spain*
Vijay Mann, *IBM, India*
Hong Mei, *Peking University, China*
Hsing Mei, *Fu Jen Catholic Unicersity, Taiwan*
Emilia Mendes, *University of Auckland, New Zealand*
Ali Mili, *NJIT, USA*
Alok Mishra, *Atilim University, Turkey*
Ana M. Moreno, *Universidad Politecnica de Madrid, Spain*
Antonio Navidad Pineda, *University of Alcala, Spain*
Kia Ng, *ICSRiM - University of Leeds, United Kingdom*
Ngoc Thanh Nguyen, *Wroclaw University of Technology, Poland*
Allen Nikora, *Jet Propulsion Laboratory, USA*
Kunal Patel, *Ingenuity Systems, USA*
Eric Pardede, *La Trobe University, Australia*
Antonio Piccinno, *University of Bari, Italy*
Alfonso Pierantonio, *University of L'Aquila, Italy*
Zhou Qiang, *TsingHua University in Beijing, China*
Rick Rabiser, *Johannes Kepler University, Austria*
Lukasz Radlinski, *University of Szczecin, Poland*
Damith C. Rajapakse, *National University of Singapore, Singapore*
Rajeev Raje, *IUPUI, USA*
Jose Angel Ramos, *Universidad Politecnica de Madrid, Spain*
Marek Reformat, *University of Alberta, Canada*
Robert Reynolds, *Wayne State University, USA*
Daniel Rodriguez, *Universidad de Alcala, Spain*
Ivan Rodero, *The State University of New Jersey, USA*
Samira Sadaoui, *University of Regina, Canada*
Masoud Sadjadi, *Florida International University, USA*
Ramon Sagarna, *The University of Birmingham, United Kingdom*
Claudio Sant'Anna, *Universidade Federal da Bahia, Brazil*

Salvatore Alessandro Sarcia, *University of Rome "Tor Vergata", Italy*
Douglas Schmidt, *Vanderbilt University ISIS, USA*
Andreas Schoenberger, *Distributed and Mobile Systems Group, University of Bamberg, Germany*
Naeem (jim) Seliya, *University of Michigan - Dearborn, USA*
Tony Shan, *Keane Inc, USA*
Yi-dong Shen, *Institute of software/Chinese academy of sciences, China*
Michael Shin, *Computer Science/Texas Tech University, USA*
Yang Qiu Song, *IBM, China*
George Spanoudakis, *City University, United Kingdom*
Gerson Sunye, *Institut de Recherche en Informatique et Systemes Aleatoires, France*
Jeff Tian, *Southern Methodist University, USA*
Peter Troger, *Humboldt-Universitat zu Berlin, Germany*
Genny Tortora, *University of Salerno, Italy*
Mark Trakhtenbrot, *Holon Institute of Technology, Israel*
T.h. Tse, *The University of Hong Kong, Hong Kong*
Giorgio Valle, *The University of Milan, Italy*
Michael Vanhilst, *Florida Atlantic University, USA*
Sylvain Vauttier, *Ecole des mines d'Ales, France*
Silvia Vergilio, *Federal University of Parana (UFPR), Brazil*
Akshat Verma, *IBM, India*
Arndt Von Staa, *PUC-Rio, Brazil*
Huanjing Wang, *Western Kentucky University, USA*
Limin Wang, *VMware Inc., USA*
Hironori Washizaki, *Waseda University, Japan*
Victor Winter, *University of Nebraska at Omaha, USA*
Guido Wirtz, *Distributed Systems Group, Bamberg University, Germany*
Franz Wotawa, *TU Graz, Austria*
Haiping Xu, *University of Massachusetts Dartmouth, USA*
Jijiang Yan, *Tsinghua University, China*
Chi-lu Yang, *Taiwan Semiconductor Manufacturing Company Ltd., Taiwan*
Hongji Yang, *De Montfort University, United Kingdom*
Junbeom Yoo, *Konkuk University, South Korea*
Huiqun Yu, *East China University of Science and Technology, China*
Cui Zhang, *California State University Sacramento, USA*
Du Zhang, *California State University, USA*
Jing Zhang, *Motorola Inc., USA*
Min-ling Zhang, *College of Computer and Information Engineering, Hohai University, China*
Yong Zhang, *TsingHua University in Beijing, China*
Zhenyu Zhang, *The University of Hong Kong, Hong Kong*
Hong Zhu, *Oxford Brookes University, United Kingdom*
Xingquan Zhu, *Florida Atlantic University, USA*
Eugenio Zimeo, *University of Sannio, Italy*

Poster/Demo Sessions Co-Chairs

Raul Garcia Castro, *Universidad Politecnica de Madrid, Spain*
Farshad Samimi, *Trilliant, USA*
Ming Zhao, *Florida Int'l University, USA*

Publicity Co-Chairs

Xiaoying Bai, *Tsinghua University, China*
Shihong Huang, *Florida Atlantic University, USA*

Asia liaison

Hironori Washizaki, *Waseda University, Japan*

South America Liasion

Jose Carlos Maldonado, *University of Sao Paulo, Brazil*

Proceedings Cover Design

Gabriel Smith, *Knowledge Systems Institute Graduate School, USA*

Conference Secretariat

Judy Pan, *Chair, Knowledge Systems Institute Graduate School, USA*
Norrjhan Ali, *Knowledge Systems Institute Graduate School, USA*
Dennis Chi, *Knowledge Systems Institute Graduate School, USA*
David Huang, *Knowledge Systems Institute Graduate School, USA*
Rachel Lu, *Knowledge Systems Institute Graduate School, USA*

Table of Contents

Model-Driven Development

Software Engineering with Computer Intelligence & Machine Learning

Formal Methods

Concurrent Software

Knowledge Engineering Tools and Techniques

Semantic Web Technologies

Software Testing and Debugging

Automated Software Testing

Software Engineering Case Studies and Experience Reports

Embedded, Pervasive, and Ubiquitous Software

Software Project Management

Knowledge Acquisition

Knowledge Representation and Visualization

Web and Data Mining

Ontologies and Methodologies

Software Maintenance and Evolution

Measurement & Empirical Software Engineering

Component-Based Software Engineering

Adaptive and Self-Managing Software

Service-Oriented Architecture

Agile-Based Software Engineering

Software Engineering Tools and Environments

Software Dependability and Reliability

Software Process Modeling & Maturity

Software Security

Software Product Lines and Tools

Software Requirements Engineering

Software Architecture

Poster/Demo

Note: (S) indicates a short paper.
(P) indiecates a poster or demo, which is not a refereed paper.

Keynote I
Sustainable Software Systems for Real Time Applications

Dr. S. Sitharama Iyengar
Ryder Professor of Computer Science
Director of School of Computing and Information Sciences
Florida International University
and
Roy Paul Daniels Professor and Chairman
Department of Computer Science
Louisiana State University

Abstract

Development of software systems for Real Time Applications is at a critical juncture. Investments in software developments for these sytems will require transformative changes in the context of science, design and policies. This talk provides an overview in achieving sustainable software systems for sensor based real time applications. It will also focus on the consumer demand and policy incentives of these applications.

About the Speaker

Dr. Iyengar is a Member of the European Academy of Sciences, a Fellow of the Institute of Electrical and Electronics Engineers (IEEE), a Fellow of the Association of Computing Machinery (ACM), a Fellow of the American Association for the Advancement of Science (AAAS), and a Fellow of the Society for Design and Process Science (SDPS). He has received the Distinguished Alumnus Award of the Indian Institute of Science, the IEEE Computer Society's Technical Achievement Award, and the IEEE Golden Core Member award. Dr. Iyengar has published over 400 research papers in journals and conferences and has authored, co-authored, or edited 14 texts in the areas of parallel algorithms, sensor networks, wavelets, robotics, and computer modeling of complex biological systems. Cited well over 4,500 times, his research has been funded by DARPA, NSF, ONR, DOE, MURI, NRL, ARO, NASA, and state governments and the industry. Dr. Iyengar has graduated 42 Ph.D. students. He is the founding Editor-In-Chief of the International Journal of Distributed Sensor Networks and has been an Associate Editor for IEEE Transaction on Computers, IEEE Transactions on Data and Knowledge Engineering, and guest Editor of IEEE Computer Magazine. Dr. Iyengar is a pioneer in the field of distributed sensor networks, computational aspects of robotics, and oceanographic applications. He is best known for introducing novel data structures and algorithmic techniques for large scale computations in sensor technologies and image processing applications. His work has had significant impacts in buy-at-bulk network design problems which further affect a wide range of practical applications including the areas of communication and transportation networks. He has also pioneered a universal technique for finding the Price of Anarchy of every bottleneck congestion game, extending the game theory to applications in network routing, scheduling and cloud computing.

Keynote II
Applications & Services Exploration for the Broadband Mobile Systems

Bao-Shuh Lin, Director
Microelectronics and Information Systems Research Center (MIRC)
National Chiao Tung University, Taiwan

Abstract

The system characteristics of future broadband mobile systems are high throughput (data rate), low latency (delay), high mobility (km/hr), and high capacity. The current recognized broadband mobile systems should meet the requirements specified by IMT-Advanced. Those broadband mobile systems include 3GPP-LTE/LTE-advanced and IEEE-802.16e/802.16m (Mobile WiMAX 1.0/ Mobile WiMAX 2.0). In the mean time, the smart phones with powerful embedded CPU/GPU, high-resolution digital camera, GPS, various sensors and digital compass are becoming popular. With the combination of the deployment of broadband mobile systems and the usage of smart mobile phones, there are many potential appealing applications & services can be creating. In this talk, we will explore several software intensive applications and services include mobile augmented reality, mobile healthcare, and high speed rail.

About the Speaker

Prof. Bao-shuh P. Lin, an IEEE Fellow, is a Chair Professor of Department of Computer Science and the Chief Director of Microelectronics and Information Systems Research Center (MIRC), National Chiao Tung University (NCTU) in Hsinchu, Taiwan since September 2009. He was the Vice President of Industrial Technology Research Institute (ITRI) and the General Director of the Information & Communications Research Laboratories (ICL) of ITRI from 2001 to 2009. During the same period, he was also the Director of Committee of Communication Industry Development (CoCID), Ministry of Economic Affairs (MOEA) in Taiwan. He obtained his Ph.D. in Computer Science from the University of Illinois at Chicago, USA, and has published more than 130 technical papers and reports in ICT fields. In his long professional career, Dr. Lin has won numerous significant awards, has been actively involved in science and technology policy formulation, and helped initiate national programs on System on Chip, Broadband and Wireless Communications, All-IP Networking and Services, and Mobile Digital Life technologies. All these efforts have exerted tremendous impacts on economic and social benefits in Taiwan.

Visual Specification of Component-based Slow Intelligence Systems

Shi-Kuo Chang[1], Yingze Wang[1] and Yao Sun[1]
[1]Department of Computer Science, University of Pittsburgh, Pittsburgh, PA 15260, USA
{chang, yingzewang, yaosun}@cs.pitt.edu

Abstract

A new approach for the visual specification of component-based software systems is proposed. The main idea is to employ two visual representations - the I-card (information card) and the C-card (control card) - together to specify a component-based software system. This approach is especially useful in the specification and design of component-based slow intelligence systems. This approach is currently being applied to the design of slow intelligence system for social influence analysis.

Keywords

Visual specification, component-based software system, slow intelligence system, super component

1 Introduction

In the specification of component-based software systems, control and timing information is often unclear or even lost. If a formal, mathematical approach is used, control and timing information could be buried in the details and become unclear to the designer/programmer. If a less formal approach is used, such information could be lost.

In this paper, a new approach for the visual specification of component-based software systems is proposed. The main idea is to employ two visual representations - the *I-card (information card)* and the *C-card (control card)* - together to specify a component-based software system. This approach is seemingly simple, but actually quite powerful. This approach is also especially applicable to the specification and design of slow intelligence systems where *super components* are employed to search for solutions.

A slow intelligence system (SIS) is a general-purpose system characterized by being able to improve performance over time through a process involving *enumeration, propagation, adaptation, elimination and concentration* [Chang2010]. A SIS is a component-based software system, but it differs from ordinary component-based systems by employing super components, in the sense that multiple, similar components can be activated either sequentially or in parallel to search for solutions. The proposed visual specification approach is especially suitable for the specification and design of such systems with super components.

The paper is organized as follows: Section 2 explains in detail this visual specification approach using dual visual representations. In Section 3, we describe the user interface design to produce and manage the dual visual representations for the generic SIS system. Finally, Section 4 discusses future works.

As described in [Chang2011b], we are applying this approach to design the slow intelligence system for social influence analysis (SIA).

2 Visual Specification of Component-based Software Systems

2.1. Dual Visual Representations

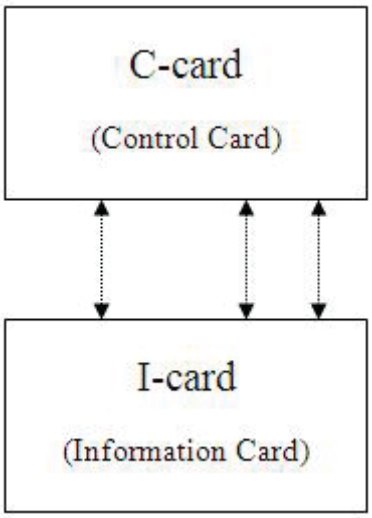


Figure 2.1. Dual visual representations.

The main idea is to employ two visual representations - the *I-card (information card)* and the *C-card (control card)* - together to specify a component-based software system. A generic example is illustrated in Figure 2.1. The I-card specifies the logical relationships among the components, and the C-card specifies the control and timing relationships among the components. Their inter-relationships are represented by bi-directional arcs (the dotted lines) connecting components, or other entities, in the I-card and the C-card.

2.2. An Example

In the following example, the two components X1 and X2 can be either simple components or super components. We will first give an example involving only simple components. Suppose there are two components X1 and X2. The component X1 is invoked first, followed by the invocation of component X2.

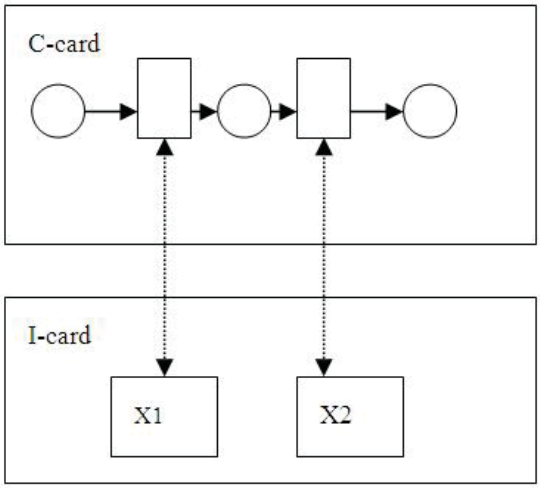


Figure 2.2. Dual visual representations by components and Petri net.

As shown in Figure 2.2, the two components are specified in the I-card as two rectangles. This specification for components can be further refined. For instance each component can be described by a detailed class diagram in UML. The timing relationships among the two components are specified by the Petri net in the C-card, where places are represented by small circles, and transitions by vertical rectangles. The Petri net's transitions in the C-card correspond to the components in the I-Card, therefore they are linked by dotted lines.

Although a Petri net is used in the above example, we can also replace the Petri net by other kinds of timing diagrams such as the sequence diagram or the activity diagram in UML. We can now give another example as illustrated by Figure 2.3. This example illustrates the simplicity, flexibility and versatility of this approach. We can use different diagrams in the I-card and the C-card to present different views of the component-based software system.

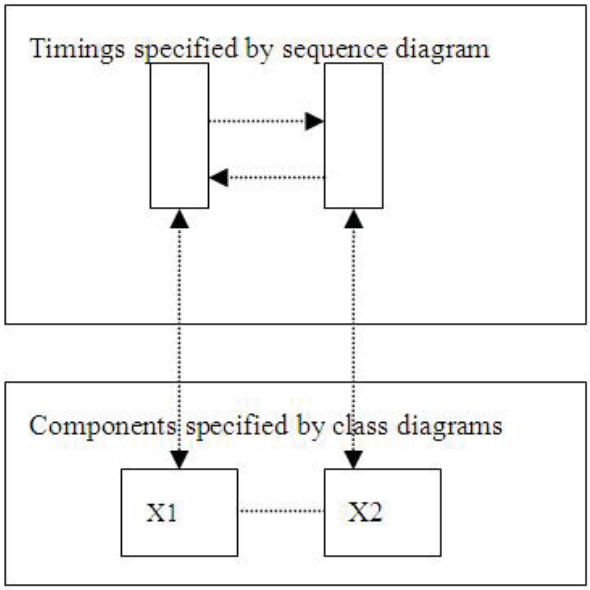


Figure 2.3. Dual visual representations by class diagram and sequence diagram.

It is also possible to associate multiple diagrams for multiple views of a component-based software system. However our primary interest in this paper is to introduce the notation for super components so that we can specify and design slow intelligence systems using this extended notation. Multiple view visualization and consistency checking will be the topic for a subsequent paper.

2.3. Super Components

As mentioned before, in the above example the two components X1 and X2 can be either simple components or super components. A *super component* X1 is a collection of similar components x_1^1, x_1^2, …, x_1^j, retrieved from some component database or generated by some algorithm. As shown in Figure 2.4 the visual representation of a super component is a box with double-edged border.

With super components, the Petri net in the C-card is actually a concise representation of a much larger underlying Petri net, such as the one shown in Figure 2.5. It should be pointed out this is by no means the only possible expanded Petri net. As noted above, depending upon the meaning of the association between the I-card and the C-card, different expanded Petri nets can be generated.

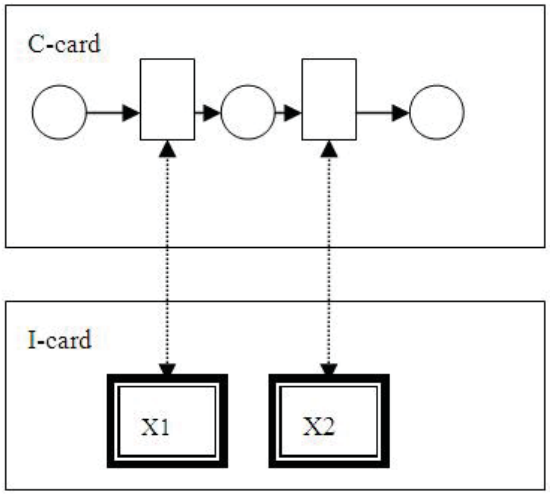


Figure 2.4. Super Components.

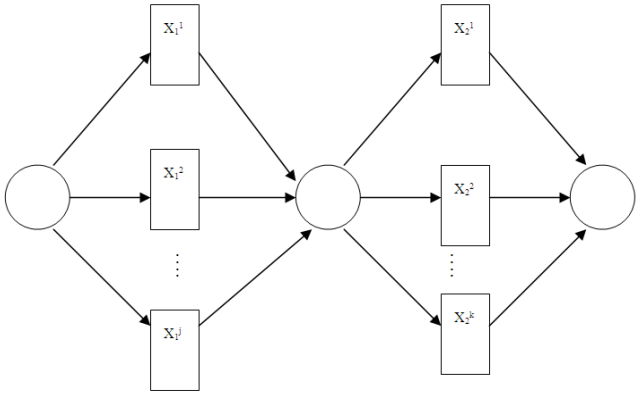


Figure 2.5. Expanded Petri net based upon super components.

With different visual diagrams embedded in the C-card, the interpretations could be different in each case. For instance if sequence diagrams are used instead, we can envision for each super component, a subsidiary sequence diagram could be generated to render a precise and detailed specification of the entire system.

2.4. SIS Design using Super Components

The super component can also be further refined. A super component is essentially a concise representation of a collection of components. Therefore we can employ a generic design pattern or a *component generator* to specify such a super component.

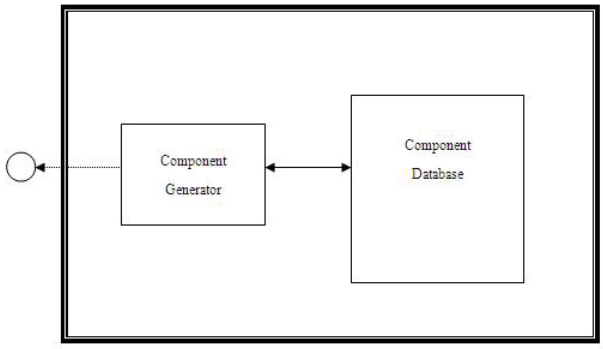


Figure 2.6. A component generator for super components.

In Figure 2.6 the small circle indicates the sub-system interface. The component generator produces customized components by accessing the component database. In practice, the component database could contain the parameters to generate customized components. In some cases the components can be algorithmically generated without referring to a component database. In which case, Figure 2.6 can be further simplified by eliminating the component database.

As noted in [Chang2010] a slow intelligence system can be designed based upon five SIS primitives (or five SIS design patterns): *enumerator*, *adaptor*, *eliminator*, *concentrator* and *propagator*. They can also be specified using the notation of super components. Therefore, with the notation of super components, and the dual visual representations using I-card and C-card, we can visually design a slow intelligence system. In the following sections, we will describe the design of a versatile user interface supporting visual design using the dual visual representations.

3 User Interface for SIS System

In this section we describe the user interface design to create and manage the dual visual representations for the SIS system. Based on SIS system framework, the user interface can be designed for three specific tasks: (1) Interface for designing control card. In our generic SIS system, control card (C-Card) can define functional dependency repertoire. (2) Interface for designing information card (I-Card). For SIS system, I-card can define candidate functions repertoire. (3) Interface for designing testing data, which in our SIS system is stored in test data repertoire.

3.1. User Interface for designing C-card

Each functional component is a super component or a simple component that can be represented by a transition in a Petri Net. Here super component is a collection of components; while simple component is a single component. We need to define the dependencies between

different functional components. User interface is used to draw Petri Net as shown in Figure 3.1.

We developed our user interface based on [Bonet 2007] PIPE tool, which is a powerful software for editing Petri Net. In Figure 3.1, particularly, unfilled rectangle represents transition for defining simple component. Filled rectangle represents super component. Circles represent the places. User can utilize the Petri-net to define the sequence and dependency among functional components. Moreover, we extend the meaning of Petri Net that places can represent the messages sending from one transition to another, as shown in Figure 3.2. Since slow intelligence system uses messages to communicate among each component. Thus, in message editor, user can define a list of messages related to the particular place, which enhance the power of Petri Net. Note that messages can be automatically mapped to MsgID defined in system, so that users don't need to know the exact MsgID. After designing the Petri-net for a specific system, user can save the diagram in xml format, such as the Petri Net Markup Language (PNML), which is an XML-based syntax for high-level Petri nets designed as a standard interchange format for Petri net tools [Weber2003].

We give an example as shown in Figure 3.1. This is a simple Petri Net with two transitions named "Enumerator" and "Diffusion Model". Enumerator represents a simple component while "Diffusion Model" represents a super component. This Petri Net shows that Diffusion Model depends on Enumerator. Such dependency relationship indicates that super component Diffusion Model should execute after Enumerator's execution. Also Enumerator can send message "Enumeration Notify" to Diffusion Model edited by place P1. The saved xml file of this Petri Net in PNML format is:

```
<pnml>
 <net id="net1">
   <name>
    <text>System C-Card</text>
   </name>
 <page id="page1-net1">
  <place id="p0">
   <graphics>
    <position x="30" y="55"/>
    <offset x="0" y="-10" />
   </graphics>
  </place>
  <place id="p1">
   <graphics>
    <position x="180" y="50"/>
    <offset x="0" y="-10" />
   </graphics>
   <messages>
    <value>521 Enumerator Diffusion Model
    </value>
   </messages>
  </place>
  <place id="p2">
   <graphics>
    <position x="335" y="50"/>
    <offset x="0" y="-10" />
   </graphics>
  </place>
  <transition id="Enumerator">
   <graphics>
    <position x="115" y="55"/>
    <offset x="0" y="0" />
   </graphics>
  </transition>
  <transition id="Diffusion Model">
   <graphics>
    <position x="265" y="50"/>
    <offset x="0" y="0" />
   </graphics>
  </transition>
  <arc id="a0" source="p0" target="Enumerator">
  </arc>
  <arc id="a1" source="Enumerator" target="p1">
  </arc>
  <arc id="a2" source="p1" target="Diffusion Model">
  </arc>
  <arc id="a3" source="Diffusion Model" target="p2">
  </arc>
 </page>
 </net>
</pnml>
```

3.2. User Interface for designing I-card

After designing the dependencies among functional components for SIS generic system, the user needs to devise the information card for each component (super component and/or simple component). For a super component containing a number of candidate components, we need to use I-Card to add each candidate component from database or from the template algorithm with different parameters to this super component. For a simple component in Petri-net, although there is only one component, we still need to add it from database or specific algorithm. In slow intelligence system, there are two types of functional components: system components like enumerator, eliminator, etc; algorithm components like diffusion model for social influence analysis. Each type of component can be simple or super. Usually system component is simple one while algorithm component is super one. The user interface design is shown in Figure 3.3.

When user right clicks on transition, the transition editor

will pop up. User can define the component type in comb box and edit the corresponding component I-Card. If it is the existing one, the user can load it. The system component I-Card editor interface contains a comb box to define name of component. Each type of system component has specified Key. When user selects one, the corresponding key will shown on table. The user only needs to define the value of the Key. Figure 3.3 gives an example. The system component I-Card is saved as a .txt file using format:

$$BEGIN and END are keywords
BEGIN ConcurrentComps
5
END ConCurrentComps

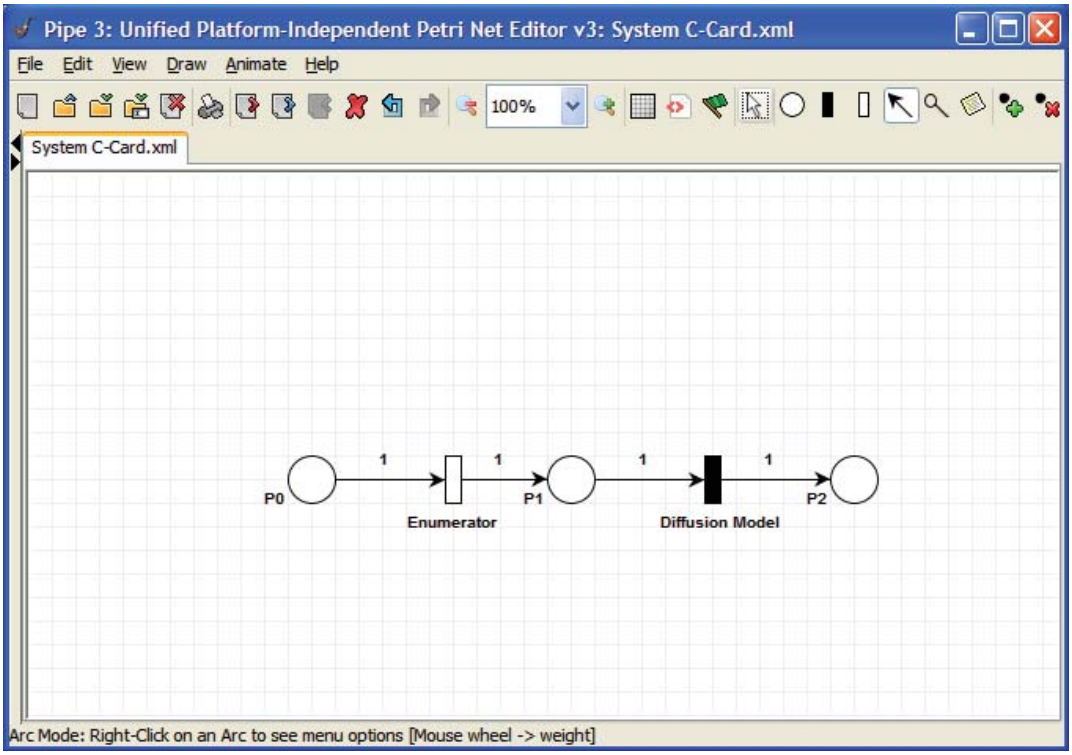


Figure 3.1. Screenshot for C-card Design.

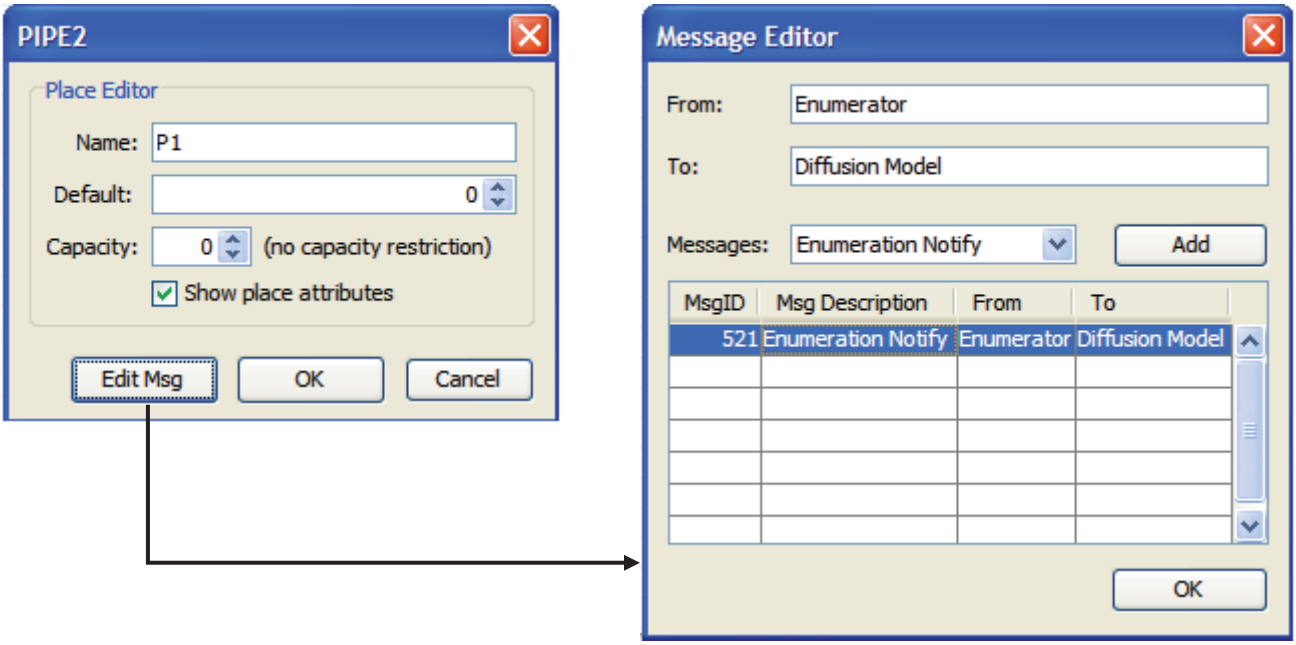


Figure 3.2. Screenshot for place editor and message editor design.

Algorithm component I-Card editor contains three Panels. In panel "property", if the transition is filled rectangle, the type should be super component and the user can add each candidate component to this super set. If the transition is unfilled rectangle, the type should be simple component and the user only needs to add one candidate component. Since the candidate (simple or super) components in each functional block do the same job using different algorithms, thus in the interface, "Functions" edit box describes the job of this functional block. User should choose the way that enumerator of SIS system (Component generator) performs, using all the candidate components or using only one of them. If user chooses "one candidate component", he/she should input the default candidate ID number that will be utilized. The list on right shows the name of "Candidate components" that user added. Also user can click "details" tab to see the details of each candidate component in a table. For each algorithm functional component (I-Card), there is a test data set for testing each candidate component in this block. Thus in panel "Testing Data", user can edit the dataset by T-Card editor in next subsection. This algorithm component I-Card can be saved as xml format.

We give a simple example as shown in Figure 3.3. This I-card corresponds to transition "Diffusion Model" in Figure 3.1, which is a super component containing two candidate components. The candidates are from template algorithm with different parameters. Thus the user can load the template and set different parameters as shown in Figure 3.3. Since the user defines by enumerating all candidate components, thus there is no default one. The saved xml file of this Petri Net could follow the format:

```
<Msg>
  <Head>
    <Name>
    </Name>
    <Description>
    </Description>
  </Head>
 <Body>
<Item>
      <Key>
      </Key>
    <Value>
    </Value>
  </Item>
 <Body>
</Msg>
```

Tab <Name> shows the name of I-Card. Tab <Description> shows brief description of this I-Card. Tab <Key> indicates the key text such as: Type, Candidate Name, etc. Tab <Value> indicates the value of user's input in textbox. Thus, <key> and <value> should be pairs. For example, some pairs as shown in Figure 3.3 are:

```
  <Item>
   <Key>Candidate Name
   </Key>
  <Value>Diffusion Model 2
  </Value>
 </Item>
  <Item>
   <Key>ID Number
   </Key>
  <Value>2
  </Value>
 </Item>
  <Item>
   <Key>Template
   </Key>
  <Value>Yes
  </Value>
 </Item>
  <Item>
   <Key>Parameter
   </Key>
  <Value>3.0
  </Value>
 </Item>
```

3.3.User interface for designing test data

Figure 3.4 shows the interface of designing testing data for testing each candidate component in algorithm functional component block. This split panel includes two parts. The left part is used to load testing data from existing files and to specify the output results property. User can browse existing file and define the place to store the result. The right part is designed for the purpose of simple testing and debugging. User can manually edit the data on the table. The input data could be any value including number, string, boolean variable. The output data is the correct result corresponding to input. Then user can compare their algorithm's results to these standard results.

We give an example as shown in Figure 3.4. This T-card corresponds to algorithm component I-Card defined in Figure 3.3. The T-Card is saved as a .txt file using format:

```
$$BEGIN and END are keywords
BEGIN InputDataList
/*List test data files*/
Twitter.txt          D:\SIS\Data\Twitter.txt
END InputDataList

BEGIN OutputPro
/*List output results property*/
txt        D:\SIS\Result
END OutputPro
```

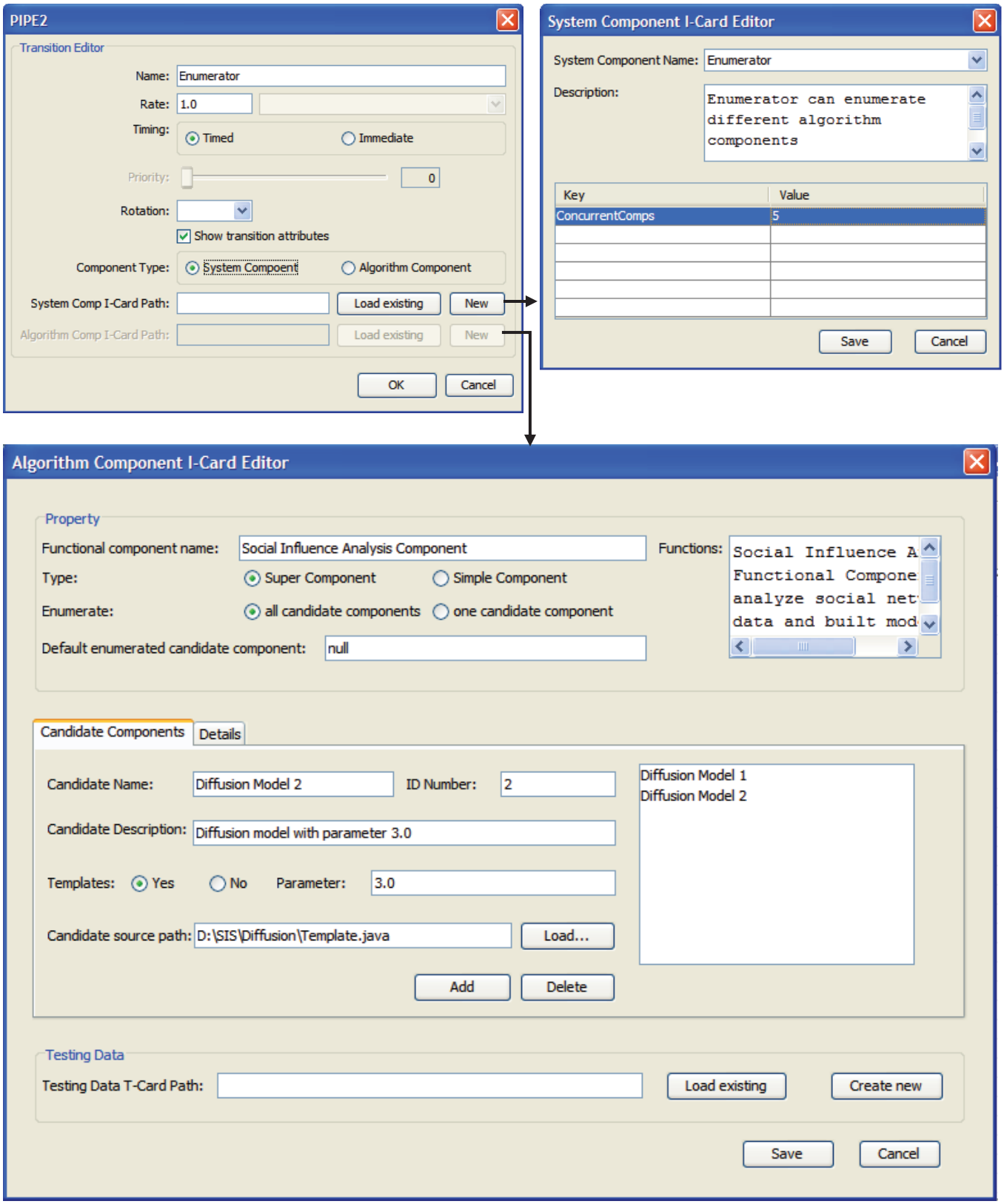


Figure 3.3. Screenshot for I-Card design.

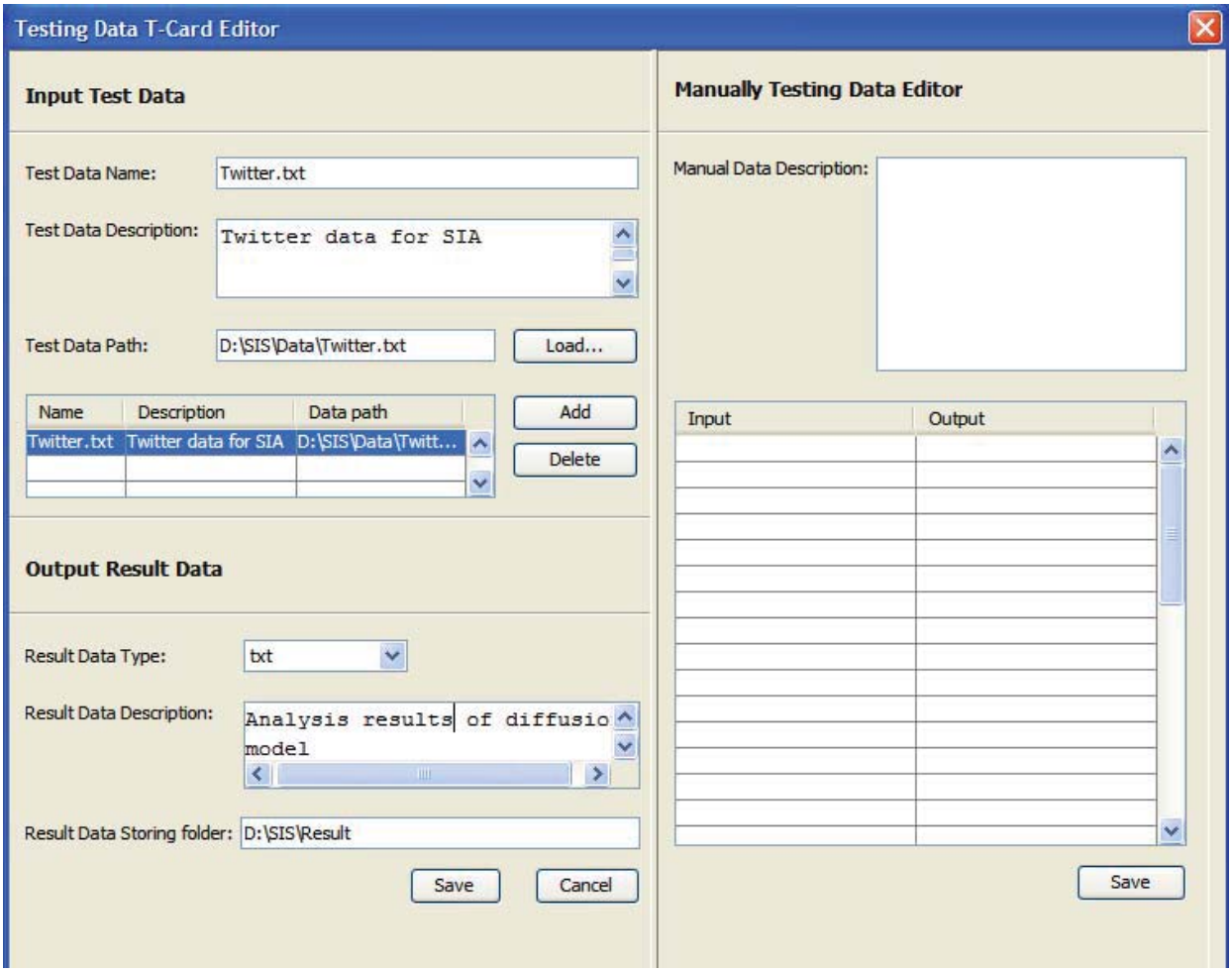


Figure 3.4. Screenshot for Testing Data design.

4 Discussion

In this paper we described a new approach for the visual specification of component-based software systems by employing two visual representations - the *I-card (information card)* and the *C-card (control card)*. The user interface for visual specification of component-based software system is currently being developed. For practical convenience, a *T-card* can be used to specify test data.

We can use different diagrams in the I-card and the C-card to present different views of the component-based software system. It is also possible to associate multiple diagrams for multiple views. Furthermore, the bi-directional arcs between the two visual representations can be labeled to specify the meaning of the associations so that different interpretations can be generated from the visual specification. How to check the consistency of multiple views, and how to synthesize a consistent spec from multiple views, will be topics for further research.

References

[Chang2010] Shi-Kuo Chang, "A General Framework for Slow Intelligence Systems", International Journal of Software Engineering and Knowledge Engineering, Volume 20, Number 1, February 2010, 1-16.

[Chang2011b] Shi-Kuo Chang, Yao Sun, Yingze Wang, Chia-Chun Shih and Ting-Chun Peng, "Design of Component-based Slow Intelligence Systems and Application to Social Influence Analysis", Proceedings of 2011 International Conference on Software Engineering and Knowledge Engineering, Miami, USA, July 7-9, 2011, 9-16.

[Weber2003] Michael Weber and Ekkart Kindler, "The Petri Net Markup Language ", Petri Net Technology for communication-Based Systems – Advances in Petri Nets, LNCS volume 472, 2003, 124-144.

[Bonet 2007] P. Bonet, C.M. Llado, R. Puijaner and W.J. Knottenbelt, "PIPE v2.5: A Petri Net Tool for Performance Modelling", Proc. 23rd Latin American Conference on Informatics (CLEI 2007), San Jose, Costa Rica, October 2007.

Design of Component-based Slow Intelligence Systems and Application to Social Influence Analysis

Shi-Kuo Chang[1], Yao Sun[1], Yingze Wang[1], Chia-Chun Shih[2] and Ting-Chun Peng[2]
[1]Department of Computer Science, University of Pittsburgh, Pittsburgh, PA 15260, USA
{chang, yaosun, yingzewang}@cs.pitt.edu
and
[2]Institute for Information Industry, Taiwan, Republic of China
{chiachun, markpeng}@iii.org.tw

Abstract

We present the architecture and operational characteristics of a generic SIS system. To design the SIS system we also provide the data structures and operating procedures to describe each system component. We are applying this approach to design the Slow Intelligence System for Social Influence Analysis. Some initial experimental results are also reported.

Keywords

Component-based software system, slow intelligence system, super component, social influence analysis

1 Introduction

A slow intelligence system (SIS) [Chang2010] is a general-purpose system characterized by being able to improve performance over time through a process involving *enumeration, propagation, adaptation, elimination and concentration*. A SIS continuously learns, searches for new solutions and propagates and shares its experience with other peers. A SIS is a system with multiple decision cycles such that actions of slow decision cycle(s) may override actions of quick decision cycle(s), resulting in poorer performance in the short run but better performance in the long-run.

A SIS is characterized by employing super components, in the sense that multiple components can be activated either sequentially or in parallel to search for solutions. In [Chang2011a] we explained in detail a visual specification approach using dual visual representations, and described the user interface design to produce and manage the dual visual representations for the generic SIS system. In this paper we further present the design of component-based slow intelligence systems, and the application of SIS to social influence analysis. Some initial experimental results are also reported.

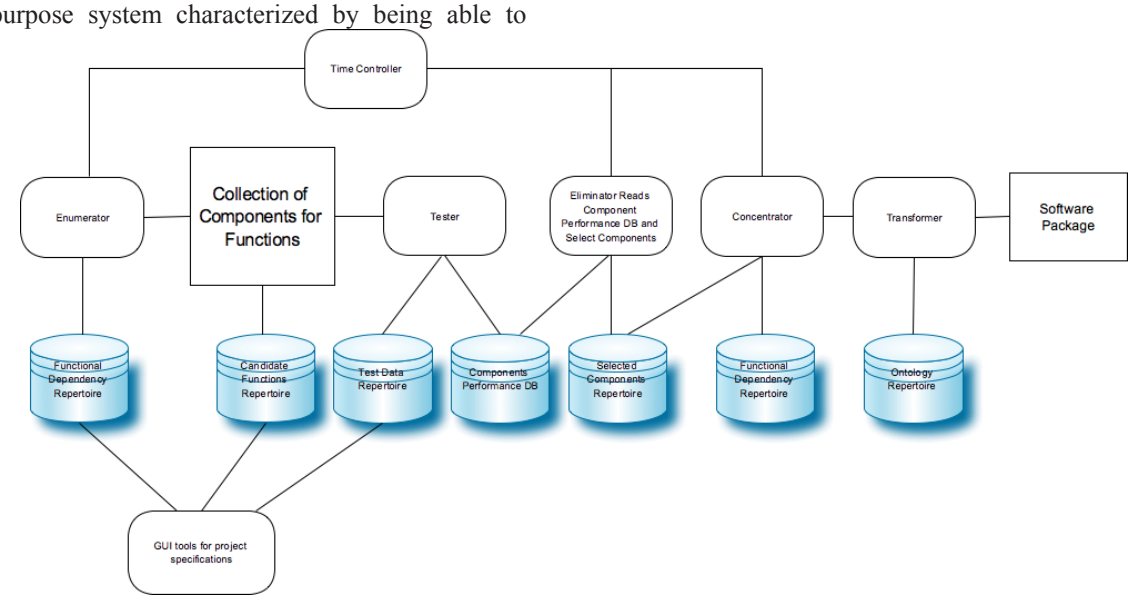


Figure 2.1. System architecture of Component-based Slow Intelligence System.

The paper is organized as follows: In Section 2 we present the architecture and operational characteristics of a generic SIS system. To design the SIS system we also provide the data structures (Section 3) and operating procedures (Section 4) to describe each key system component. In Section 5 we apply this approach to design the Slow Intelligence System for Social Influence Analysis (SIA).

2 A Generic Component-based SIS

2.1. System Architecture

Figure 2.1 illustrates the system architecture comprising the key components of the SIS system. The slow intelligence components are Enumerator, Eliminator, Concentrator and Time Controller. In addition, a customized Tester and a Transformer are included to provide more functionalities.

The system works as follows. The Enumerator reads in the specification of functional blocks and creates multiple candidate components for each functional block. The Tester tests all the functional blocks and records their performance in the DB. The Eliminator selects the best candidate components based upon their performance. The Concentrator packs the selected candidate components based on dependency specifications and generates a generic software package. When there are terminological and/or conceptual differences within the software package, Transformer is used to transform the Concentrator-generated software package to target software package that serves specific purpose. The Time Controller is sort of like an automatic system manager, it controls all the above mentioned components, telling them when to perform actions. That is to say, Enumerator, Eliminator and Concentrator perform their tasks multiple times upon Time Controller's instructions.

2.2. Specification

Specification is the very first phase of the system. As shown in Figure 2.2, a GUI program is used to specify a certain system. The GUI can be divided into three different parts. One for specifying functional blocks dependency, one for specifying pools of candidate components for functional blocks, and the other for specifying test data for functional blocks.

All the three specifications are stored in three different repertoires. *Functional blocks dependency repertoire* stores the dependencies between different functional blocks, for example, which functional block's output is which functional block's input. As what their names imply, *candidate functions repertoire* stores all the candidate components for functional blocks, and *test data repertoire* stores all the test data for use of testing candidate components for functional blocks.

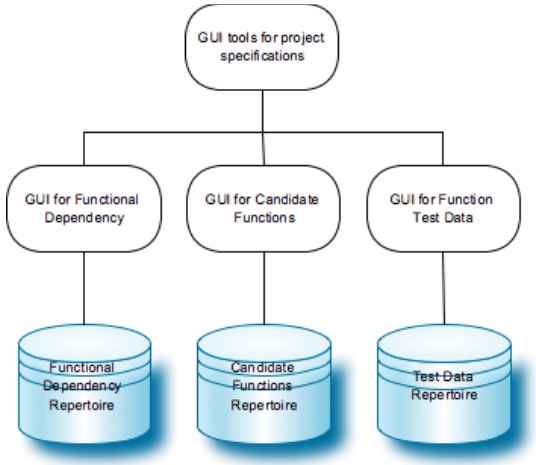


Figure 2.2. Specification of Component-based Slow Intelligence System.

In practice, we will use the dual visual representation method described in [Chang2011a] to design the GUI for our component-based Slow Intelligence System. We can use C-card to specify the functional block dependency, I-card to specify pools of candidate components for functional blocks, and T-card to specify test data.

2.3. Initial Run

In this phase, Enumerator first reads in the functional blocks dependencies from the functional blocks dependency repertoire and creates multiple candidate components for each functional block. The candidate components are selected from candidate functions repertoire. In Figure 2.3, we show in detail what is called *components for functions* in the whole system diagram. In the components for functions, all candidate components are classified into different *functional blocks*, which are actually what we have introduced as *super components* in [Chang2011a]. We use the same double-edged notation as in [Chang2011a] to depict the super components in Figure 2.3. And within each functional block, the candidate components do the same job using different algorithms or different parameters. We will discuss how the candidate components are generated in Section 4.1. Tester starts running in this phase and keeps running for ever until the system halts. Tester tests each candidate component with test data from test data repertoire and record their performance into components performance DB.

2.4. Multiple Runs

In this phase, Time Controller takes over the manager role of the system. The system goes into its main cycle of life. The system may go through multiple runs of Enumeration

and Elimination. Time Controller notifies Enumerator when to create new candidate components from candidate function repertoire and notifies Eliminator when to eliminate some candidate components based upon their performances recorded in components performance DB and store the results into selected components repertoire.

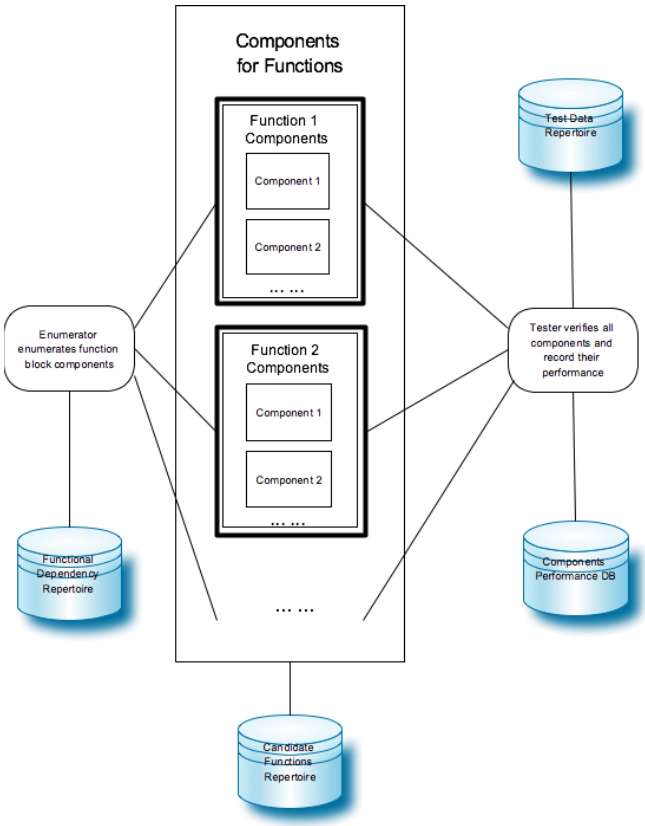


Figure 2.3. Initial Run of Component-based Slow Intelligence System.

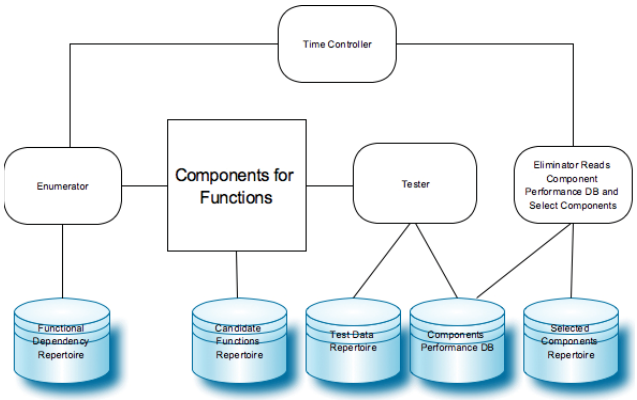


Figure 2.4. Multiple Runs of Component-based Slow Intelligence System.

2.5. Software Package Generation

This is the last phase of the system. The Concentrator reads in functional blocks dependencies from functional blocks dependency repertoire to determine the software package structure and generate a generic software package using the candidate components from selected components repertoire. Concentrator is also under control of Time Controller. When there are terminological and/or conceptual differences within the software package, Time Controller can notify Concentrator to generate intermediate software package from currently selected components and send it to the Transformer, which accesses an ontology repertoire storing the terminological and/or conceptual relations to map the intermediate software package to the target software package for specific applications.

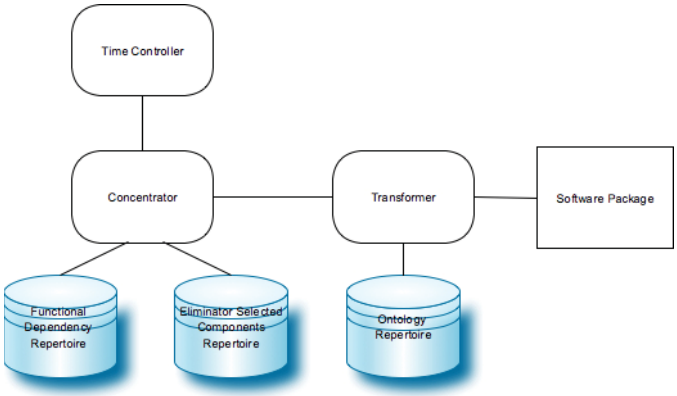


Figure 2.5. Generate Software Package.

3 Data Structures

In this section we will discuss the data structure for each repository mentioned in previous section.

3.1.Functional Blocks Dependency Repertoire

Functional blocks dependency repertoire stores the functional blocks dependencies specified by GUI tools for C-card interface described in [Chang2011a]. And it is used by Enumerator for creating functional blocks (super components)that is to be filled with candidate components. And it is also used by Concentrator to determine the functional blocks dependencies within a software package.

For functional blocks dependency repertoire, since it stores the precedence and dependencies between functional blocks, we can use structures very similar to a directed graph to store the information. We treat functional blocks as nodes and use edges to denote functional dependencies. Thus we can design the data structure as follows:

FunctionalBlockName1
FunctionalBlockName2
FunctionalBlockName3
... ...
FunctionalBlockName1->FunctionalBlockName2
FunctionalBlockName1-> FunctionalBlockName2
... ...

The first part of the data structure lists all the functional blocks that will appear in the system, and the second part lists all the functional blocks dependencies between functional blocks.

3.2. Candidate Functions Repertoire

Candidate functions repertoire stores all the candidate components specified using GUI tools for I-card interface described in [Chang2011a]. And it is used by Enumerator to generate candidate components to fill in the super components (functional blocks).

For candidate functions repertoire, we design its data structure as follows:

FunctionalBlockName1:[All or One]
Candidate1;Default;Generic
Candidate2;NonDefault;Specialized
... ...
FunctionalBlockName2:[All or One]
Candidate1;Default;Generic
Candidate2;NonDefault;Specialized
... ...

Basically, we store the candidate components by their functional block type followed by several different attributes. Here candidates denotes the address where the candidate components stores. Functional blocks can have an attribute called All or One. All means when Enumerator is generating candidate components for this functional block, it creates instances of all candidate components. One means when Enumerator is generating candidate components for this functional block, it creates only one instance of the default candidate component. Candidate components can also have different attributes. Default or Non-Default denotes whether this candidate component should be selected when the functional block has the attribute One in enumeration process. Generic or Specialized denotes whether this candidate component can have multiple instances. Generic means we can pass in different parameters to this candidate component to generate different instances. Specialized means this candidate component does not accept parameters and its instance is determined. We can have different attributes attached to candidate components when necessary.

3.3.Test Data Repertoire

Test data repertoire stores test data information for each functional block, these test data are specified by what is called T-card described in [Chang2011a]. It is used by Tester to retrieve the test data when testing different candidate components. For each functional block(super component), all its candidate components share the same test data.

For test data repertoire, the data structure is designed as follows:

FunctionalBlockName1:
Input1;Output1
Input2;Output2
... ...
FunctionalBlockName2:
Input1;Output1
Input2;Output2
... ...

3.4. Components Performance DB

Components performance DB stores all the candidate components' performance evaluated by Tester. And it is used by the Eliminator to select best or eligible candidate components. For Component performance DB, the data structure is designed as follows:

FunctionalBlockName1
CandidateName1: CorrectCount;IncorrectCount
Input1;Output1;Correctness;ResponseTime
Input2;Output2;Correctness;ResponseTime
... ...
CandidateName2: CorrectCount;IncorrectCount
Input1;Output1;Correctness;ResponseTime
Input2;Output2;Correctness;ResponseTime
... ...
... ...
FunctionalBlockName2
... ...

This means we use a list to store functional blocks, and for each functional block, we use a sub-list to store its candidate components and the candidate components' overall performance, and for each candidate component, we use another sub-list to store the candidate component's performance on certain inputs and outputs.

3.5. Selected Components Repertoire

Selected components repertoire stores all the eligible candidate components selected by Eliminator, and it is used by Concentrator to constitute a generic software package.

For component performance DB, the data structure is designed as follows:

FunctionalBlockName1:BestCandidate1
Candidate1
Candidate2
... ...
FunctionalBlockName2:BestCandidate2
Candidate1
Candidate2
... ...

This means we use a list to store functional blocks together with its best candidate component, and for each functional block, we use a sub-list to store its eligible candidate components that pass Eliminator's elimination criteria.

4 Operating Procedures for SIS System Components

In this section, we describe the operating procedures of key system components to show their functions.

4.1. Enumerator

The input to the Enumerator are functional blocks dependencies from functional blocks dependency repertoire and candidate components from candidate components repertoire. The output from Enumerator are multiple instances of candidate components for all functional blocks.

First the Enumerator reads in the functional blocks dependency specifications, retrieves all functional block names and generate a super component for each functional block. These super components are empty components that are to be filled with instances of corresponding candidate components in the next step. Next the Enumerator reads in information stored in candidate components repertoire. It checks each functional block if it needs only one candidate component instance or multiple candidate component instances. If it needs only one candidate component instance, it directly instantiate the default candidate component and fill in the super component. If it needs all candidate component instances, it instantiate all candidate component instances and fill them into the super component. There are also two ways to instantiate a candidate component. If a candidate component is generic, then we may pass in different parameters to generate multiple instances of one candidate component. If a candidate component is specialized, then only one determined instance is created for that candidate component.

4.2.Tester

The input to the Tester is all test data stored in the test data repertoire, the output of the Tester is each candidate component instance's performance, and these performances are recorded in the components performance DB.

First, the Tester reads in the test data information and classifies them by functional blocks (super components). Then it retrieves from the test data the inputs by functional block and sends the inputs to corresponding candidate components, then it checks the results returned from candidate components and compares them with the correct outputs. Finally Tester records the corresponding candidate components' performance into components performance DB. These performances are structured as described in section 3.4 in components performance DB. Tester should maintain the structure. It stores each candidate component's response time and correctness on each test input. It also maintains each candidate component's overall performance, like correctness rate and average response time.

4.3. Eliminator

The input to the Eliminator is candidate components' performance information stored in components performance DB and the output of Eliminator is eligible candidate components and best candidate component for each functional block(super component). The output is stored in selected components repertoire.

First, the Eliminator reads in all candidate components' performance information classified by functional block. Then the Eliminator eliminates unqualified candidate components by some threshold on correctness rate, i.e. eliminate candidate components whose correctness rate is below some value. The Eliminator stores the surviving qualified candidate components in selected components repertoire. We store these qualified candidate components information for next round of Enumeration and Elimination. Next, the Eliminator selects the candidate component with highest correctness rate as the best candidate for the corresponding functional block. If the correctness rates are the same for multiple candidate components, the Eliminator selects the one with shortest average response time. The best candidate component information is also stored in selected components repertoire. We may also introduce weights for different test data in the future to further evaluate candidate components.

4.4. Concentrator

The input to the Concentrator is the best candidate component information in selected components repertoire and the functional blocks dependency information in functional blocks dependency repertoire. The output is a generic software package.

First, the Concentrator reads in the functional blocks dependency information from the functional blocks dependency repertoire, then it reconstructs the functional blocks dependency graph from these information, that is a graph with functional blocks(super components) as nodes. Then the Concentrator retrieves the best candidate component from the selected components repertoire for each functional block and replace the functional blocks with their best candidate components. Finally the

Concentrator packs all best candidate components and generate a generic software package. When there are terminological and/or conceptual relations within the software package stored in an ontology repertoire, the Transformer can take this generic software package as the input and transform it into target software package to serve different specific purposes.

5 Application to Social Influence Analysis

In this section we describe how to apply this approach to the specification and design of Social Influence Analysis System. In large social networks, nodes (users, entities) are influenced by others for different reasons. How to model the diffusion processes over social network and how to predict which node will influence which other nodes in network have been an active research topic recently. Many researchers proposed various methods [Goval2010], [Yang2010], [Tang2009] in this area. How to utilize these algorithms and evolutionarily select the best one with the most appropriate parameters to do social influence analysis is our objective in applying the generic SIS system.

5.1. System Design and Implementation

First, we can utilize dual visual representations approach [Chang2011a] to design the system's control-card and information-card for the social influence analysis application. The generic SIS system architecture is shown in Figure 2.1. Omitting the details, the partial visual representations for SIA are shown in Figure 5.1.

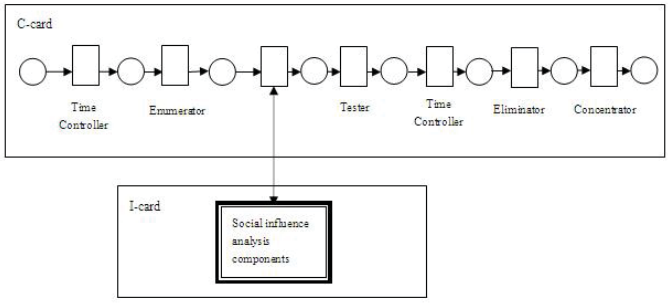


Figure 5.1. Partial visual representations for SIA system.

In this SIA system, there is only one functional block (super component) containing different candidate components for social influence analysis. All the candidate components in this super component are the different specified algorithms or the generic algorithm with different parameters doing the same job for analyzing social influence. In realization and implementation, we apply the algorithm proposed in [Yang2010], in which the authors developed a linear influence model (LIM) to analyze global influence a node has on the rate of diffusion through the implicit network. In its model, there are two parameters to control the performance. Thus we design the information card containing this generic algorithm with different parameters as the candidate components to build the super component for social influence analysis. Then our SIS system can test each parameterized algorithm and eliminate the worse ones and concentrate on the best one based on relative reduction in error.

As described in Section 2, an SIS system is a component based software system. In implementation, we use Java language. First a SIS server is implemented. The SIS server deals with all the messages between components. Messages are the only way components could communicate with each other and all the messages within the SIS system are routed through the SIS server. Components could be classified into two classes: system components and candidate components. Enumerator, Eliminator, etc. could all be treated as system components. They are responsible for initializing and managing the SIS system. And all the social influence analysis algorithms could be wrapped as candidate components in this case and provide a service to analyze social network data. They start running and wait to receive data message from the SIS server. Upon receiving the data message, they will work on the dataset and return the analysis results to SIS server. The SIS server could route the results to corresponding components such as Eliminator. Unless Eliminator closes a certain candidate component, candidate components are always running and servicing.

In social influence analysis case, the SIS system runs in the following manner. Enumerator sends initialization messages to SIS server and starts candidate components with different parameters. A data sender sends data to SIS server, SIS server routes the data to candidate components and wait for their response. When SIS server receives the response messages containing results from candidate components, the results are routed to Verifier and Eliminator for selection. After one round of elimination, the next round of enumeration starts.

We use Key/Value pairs to constitute a message. A message contains at least two Key/Value pairs, they are MsgID and Description. Customized Key/Value pairs could be added into messages. For instance, a Key/Value pair for dataset file could be added to data message. Fig. 5.2 and Fig. 5.3 provide examples of messages captured from the administrator tool of SIS system.

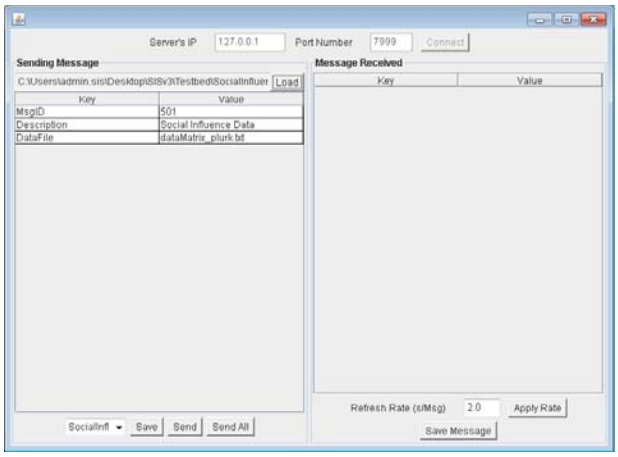


Figure 5.2. Data Message.

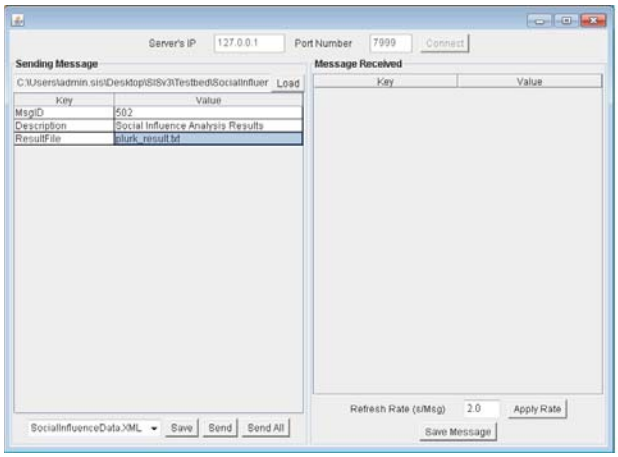


Figure 5.3. Result Message.

5.2. Experimental Setup and Initial Results

We did initial experiments on two different datasets. First, we collect the posts and responses over the Facebook between January 25th to April 22nd 2011. We choose five concepts of interest that related to some commercial products. All the concepts are in Chinese. Top 1000 active users are selected based on the number of posts in historical data. We use the model in [Yang2010] based on the influence of these 1000 users. We set the time interval unit to 3 hours. To form the matrix M, we give a number either 1 if the user u mentions the concept k in time interval t and 0 otherwise. To form the volume vector V, we count the number of users who mention/relate_to each concept k at each time t. Second, similarly we use the same scheme to collect the posts and response over Plurk between February 23rd, 2009 to April 21st, 2011. We use the same concepts of interest as facebook dataset and select 1000 active users over Plurk as well.

In [Yang2010], LIM model has a parameter L (i.e. influence of a node decays to zero after L time units). In solving the constrained linear least-squares problem:

$$\min \ \| V - M \cdot I \|_2^2 + \lambda \| I \|_2^2$$

there is a regularized parameter λ. Thus, two parameters in the model can be modified. Different combinations of these two parameters lead to different performance of the model. The SIS system can select the algorithm with the best combination. In Fig. 5.4 and Fig. 5.5, we show the comparison between the ground-truth volume V of a concept and the predicted volume by LIM with different parameters for the concept in each sub-figure. Due to the limited space, we only report the results of two concepts in both datasets with two different combinations of parameters L and λ. In practice, our SIS system can enumerate generic algorithm LIM with lots of combinations of different parameters, eliminate the worse ones, and concentrate on the best one based on the least error between the real volume and the predicted volume.

Acknowledgements

This study is conducted under the "Innovative and Prospective Technologies Project" of the Institute for Information Industry, which is subsidized by the Ministry of Economy Affairs of the Republic of China.

References

[Chang2010] Shi-Kuo Chang, "A General Framework for Slow Intelligence Systems", International Journal of Software Engineering and Knowledge Engineering, Volume 20, Number 1, February 2010, 1-16.

[Chang2011a] Shi-Kuo Chang, Yingze Wang and Yao Sun, “Visual Specification of Component-based Slow Intelligence Systems”, Proceedings of 2011 International Conference on Software Engineering and Knowledge Engineering, Miami, USA, July 7-9, 2011, 1-8.

[Goval2010] A. Goyal *et al.*, “Learning Influence Probabilities In Social Networks”, WSDM 2010.

[Tang2009] Jie Tang, Jimeng Sun, Chi Wang and Zi Yang, “Social influence analysis in large-scale networks”, KDD 2009.

[Yang2010] J. Yang and J. Leskovec, “Modeling Information Diffusion in Implicit Networks”, ICDM 2010.

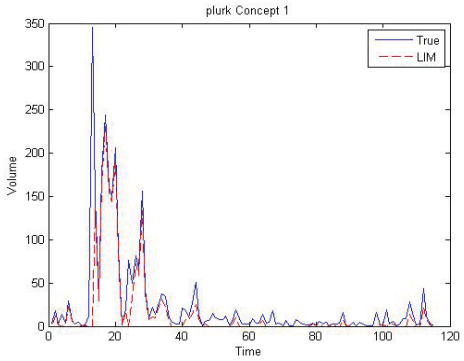
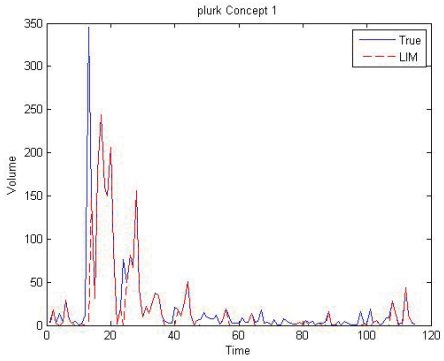
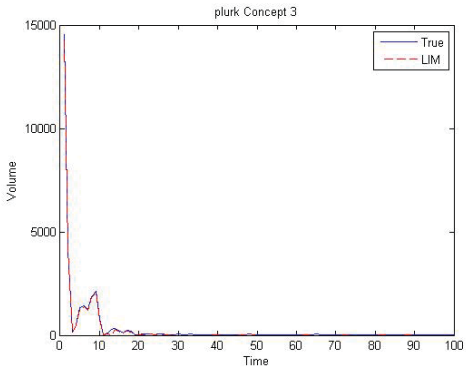
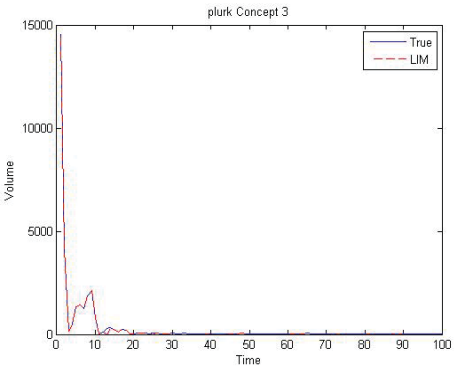
	L=2, λ =1	L=2, λ =0.0001
Concept 1		
Concept 3		

Figure 5.4. Results of concept 1 and concept 3 with two combinations of parameters in Plurk dataset.

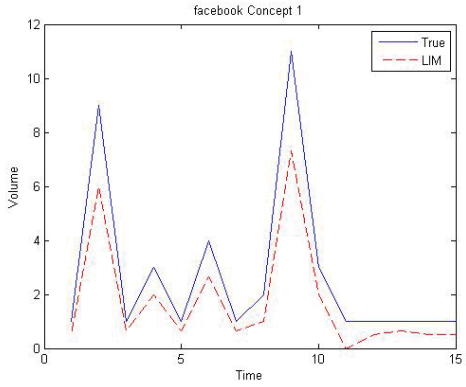
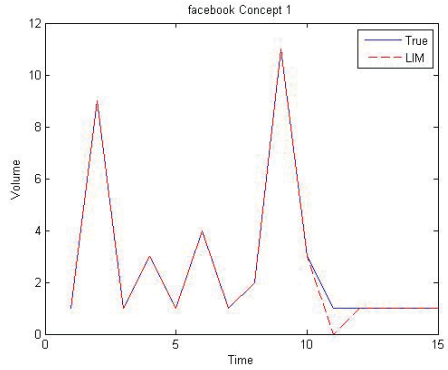
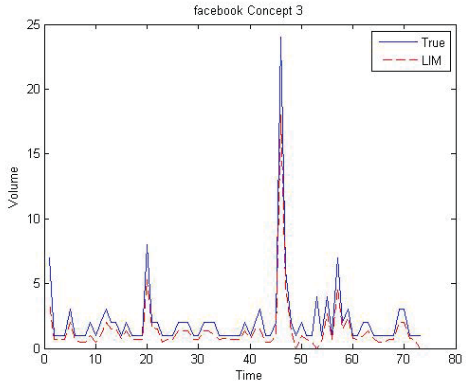
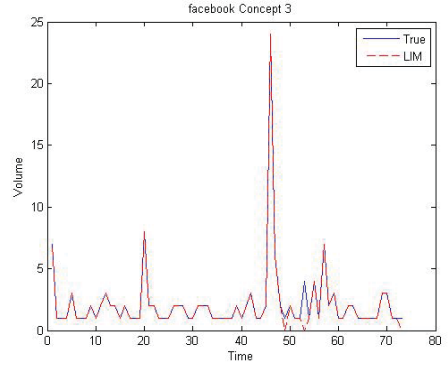
	L=2, λ =1	L=2, λ =0.0001
Concept 1		
Concept 3		

Figure 5.5. Results of concept 1 and concept 3 with two combinations of parameters in Facebook dataset.

Slow Intelligence System and Network Management: a case study

F. Colace, M. De Santo

DIEII – Università degli Studi di Salerno, Via Ponte Don Melillo, 1, 84084 Fisciano (Sa) Italy

e-mail: {*fcolace, desanto*}*@ unisa.it*

***Abstract* — The last decade has witnessed an intense spread of computer networks that has been further accelerated with the introduction of wireless networks: this growth has increased significantly the problems of network management. Especially in small companies the management of such networks is often complex and faults have significant impacts on their businesses. A possible solution is the adoption of the Simple Network Management Network administrators can manage network performance, find and solve network problems, and plan for network growth by the use of the SNMP. Over the past years much efforts has been given to make more effective the Simple Network Management Protocol and new approaches has been developed. In particular a promising approach involves the use of Ontology. The ontology based network management has recently evolved from a theoretical proposal to a more mature technology and this is the starting point of this paper where a novel approach to the network management based on the use of the Slow Intelligence System methodologies and Ontology based techniques is proposed. The Slow Intelligence System is a general-purpose system characterized by being able to improve performance over time through a process involving various phases as enumeration, propagation, adaptation, elimination and concentration. Therefore, the proposed approach aims to develop a system able to acquire, according to the Simple Network Management Protocol, information from the various hosts that are in the managed networks and apply actions in order to solve problems. To check the feasibility of this approach and its performance an experimental campaign in a real scenario has been designed and the first experimental results in a real scenario are showed.**

***Index Terms*— Ontology – Network Management – Slow Intelligence System**

I. INTRODUCTION

Networks and distributed computing systems are becoming increasingly important. This rash spread, however, resulted in increased difficulty in configuring and managing computer networks. The concept of network management is quite articulated. It involves activities such as the identification and management of various devices, monitoring their performance and much more. So efficient and intelligent configuration management techniques are to reach an automatic or semi-automatic configuration for these devices [1]. A solution for this problem can be the adoption of the Simple Network Management Protocol (SNMP). The SNMP is not only a protocol but can be considered as a general framework for the network management. This framework provide the following components [18][19]:

- network management objects known as MIB objects. In fact in the framework management information is represented as a collection of managed objects that together form a virtual information store, known as the Management Information Base (MIB)
- a data definition language known as SMI (Structure of Management Information) that defines the data types, an object model and rules for writing and revising management information
- a protocol SNMP for conveying information and commands between a managing entity and an agent executing on behalf of that entity within a managed network device
- security and administration capabilities

In literature ontology is considered a good way for supporting the network management and many papers deal with ontology based methodologies for network management [2]. Ontology based network management, in fact, has recently evolved from a theoretical proposal to a more mature technology. The main reason of this is in the significant role that ontology plays in the information model harmonization [3]. In the network management there are many management information models and a harmonization is needed. This harmonization can not be made just as a syntactic translation but a semantic translation is needed. Many papers deal with this approach: in [4] ontologies are used to provide this harmonized information model with an approach to map and merge different information model definitions, taking into account their semantics in a common ontology based model. A similar approach is in [5] where the management information is merged from several sources. Other works in recent years have also included related proposals about using ontologies for different aspects of

network management. For instance [6] proposed using ontologies for the integration of network management policies. That previously showed is the starting point of this paper. In fact it introduces a novel approach to the network management based on the use of the Slow Intelligence System methodologies [7] and ontology. The Slow Intelligence System is a general-purpose systems characterized by being able to improve performance over time through a process involving enumeration, propagation, adaptation, elimination and concentration phases. This approach works at its best when adopts the ontology for the representation of its knowledge base. So the proposed approach aims to develop a system able to acquire information from the various devices that are in the managed networks and apply solutions in order to solve problems. In particular the proposed system can handle multiple networks and adopt solutions that have proved successful in some other context. By the use of ontologies the system will be able to choose the right action to take when some hosts send alerts. The use of the Slow Intelligence System approach will allow the system to automatically infer the actions to take. In order to test the effectiveness of the proposed approach, it has been applied to various LANs that adopt the SNMP protocol for the network management. This paper is organized as follows. The next section introduces the proposed approach and describes the Slow Intelligence System and the ontology approach. The third section gives more details on the proposed approach and describes its operative workflow while the fourth section shows the experimental results. Finally some conclusions are provided.

II. A NETWORK MANAGEMENT FRAMEWORK BASED ON THE SIS APPROACH

In this section we will describe the architecture of the proposed Network Management tool through the description of its main components. In particular it will be showed how the framework works according to the Slow Intelligence System approach and by the use of the ontological formalism for the management of the knowledge base. The architecture of the proposed system is described in figure 1.

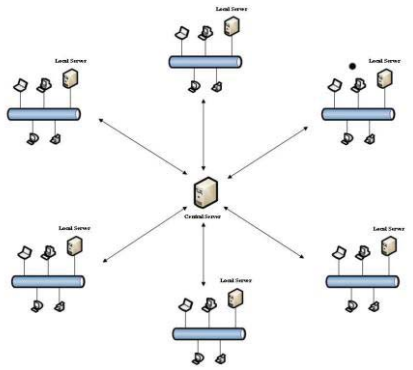


Figure 1 A possible working scenario

Each server manages a computer network and works according to the principles of the Slow Intelligence System.

II.1 The Slow Intelligence System

A Slow Intelligence System is a general-purpose system characterized by being able to improve performance over time [7]. A Slow Intelligence System continuously learns, searches for new solutions and propagates and shares its experience with other peers. It differs from expert systems in that the learning is implicit and not always obvious. A Slow Intelligence System seems to be a slow learner because it analyzes the environmental changes and absorbs that into its knowledge base while maintaining synergy with the environment. Usually a Slow Intelligence System solves problems by trying different solutions, is context-aware to adapt to different situations and to propagate knowledge and may not perform well in the short run but continuously learns to improve its performance over time [7]. A Slow Intelligence System workflow is typically composed by the following phases:

- Enumeration: In problem solving, different solutions are enumerated until the appropriate solution or solutions can be found.
- Propagation: The system is aware of its environment and constantly exchanges information with the environment. Through this constant information exchange, one SIS may propagate information to other (logically or physically adjacent) SISs.
- Adaptation: Solutions are enumerated and adapted to the environment. Sometimes adapted solutions are mutations that transcend enumerated solutions of the past.
- Elimination: Unsuitable solutions are eliminated, so that only suitable solutions are further considered.
- Concentration: Among the suitable solutions left, resources are further concentrated to only one (or at most a few) of the suitable solutions.

The sixth one, on the other hand, is rather unique for a Slow Intelligence System:

- Slow decision cycle(s) to complement quick decision cycle(s): SIS possesses at least two decision cycles. The first one, defined as the quick decision cycle, provides an instantaneous response to the environment. The second one, defined as the slow decision cycle, tries to follow the gradual changes in the environment and analyze the information acquired by experts and past experiences. The two decision cycles enable the SIS to both cope with the environment and meet long-term goals. Sophisticated SIS may possess multiple slow decision cycles and multiple quick decision cycles. Most importantly, actions of slow decision cycle(s) may override actions of quick decision cycle(s), resulting in poorer performance in the short run but better performance in the long run. The structure of a Slow intelligence System by the introduction of the basic building block and advanced building block.

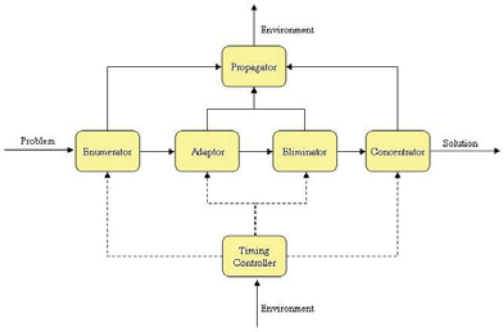


Figure 2 A basic building block BBB

Problem and solution are both functions of time, thus we can represent the time function for problem as x(t)problem, and the time function for solution as y(t)solution. The timing controller is also a time function timing-control(t). For the two-decision-cycle SIS, the basic building block BBB can be expressed as follows:

```
if timing-control(t) == 'slow'
then /* timing-control(t) is 'slow' */
        y(t)solution = gconcentrate (geliminate (gadapt
(genumerate(x(t)problem))))
else /* timing-control(t) is not 'slow' */
        y(t)solution = fconcentrate (feliminate (fadapt
(fenumerate(x(t)problem))))
```

where $g_{enumerate}$, g_{adapt}, $g_{eliminate}$, $g_{concentrate}$ are the transform functions for enumeration, adaptation, elimination and concentration respectively during slow decision cycles, and $f_{enumerate}$, f_{adapt}, $f_{eliminate}$, $f_{concentrate}$ are the transform functions for enumeration, adaptation, elimination and concentration respectively during quick decision cycles. An Advanced Building Block can be a stand-alone system as shown in Figure 3. The major difference between an ABB and a BBB is the inclusion of a knowledge base, further improving the SIS's problem solving abilities.

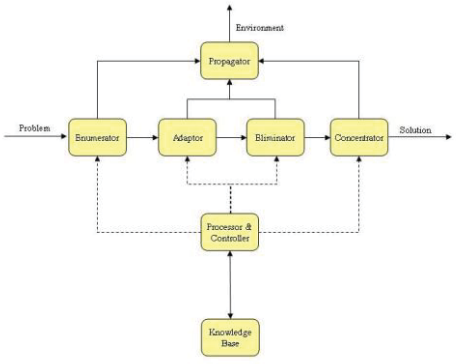


Figure 3 The advanced building block ABB

As showed in figure 3 the advanced building block works using a Knowledge Domain that contains all the information that the ABB needs in order to manage the various problems. Each ABB works with a well defined Knowledge Domain that can change through the interaction with the other peers. An effective way for the representation of the Knowledge Domain is the adoption of the ontology formalism. In the next paragraph more details on the Ontology formalism will be given.

II.2 The role of the Ontology in a Slow Intelligence System

The definition of ontology is still a challenging task [8]. The term 'ontology' has its origin in the Greek word 'ontos', which means 'being'. So in this sense ontology could be defined as a branch of philosophy dealing with the order and structure of reality. In the 1970s ontology came to be of interest in the computer science field. In particular the artificial intelligence community started to use the concept in order to create a domain of knowledge and establish formal relationships among the items of knowledge in that domain for performing some processes of automated reasoning, especially as a means for establishing explicit formal vocabulary to be shared among applications. The term 'ontology' was first used in the computer science field by Gruber who used the term to refer to an explicit specification of a conceptualization [9]. The use of this term is rapidly growing due to the significant role it plays in information systems, semantic web and knowledge-based systems, where the term 'ontology' refers to "the representation of meaning of terms in vocabularies and the relationships between those terms" [10]. Also this kind of definition is still satisfactory for each field where ontology can be applied and so perhaps a good practical definition would be this: "an ontology is a method of representing items of knowledge (ideas, facts, things) in a way that defines the relationships and classification of concepts within a specified domain of knowledge" [8]. Following this point of view, ontologies are "content theories", since their principal contribution lies in identifying specific classes of objects and the relations that exist in some knowledge domains [11][12]. Ontologies are usually classified into lightweight and heavyweight ontologies [12]. Lightweight ontologies include concepts, simple relationships among concepts (such as specialization is_a) and properties that describe concepts. Heavyweight ontologies add axioms and constraints to lightweight ontologies. Axioms and constraints clarify the intended meaning of the terms gathered in the ontology. Heavyweight and lightweight ontologies can be modelled by the use of different knowledge modelling techniques and they can be implemented in various kinds of languages which are usually divided in two groups: classical and ontology mark-up language [13]. The ontology mark-up languages, mainly used in the context of semantic web and of which the most important is OWL [10], have their own syntax, their own expressiveness, different knowledge representation paradigms and their own reasoning capabilities provided by different inference engines [13]. It is important to underline how database community as well as the object oriented design community build models using concept, relations and properties but they usually impose less semantic constraints. Ontologies are typically not static entities and so in recent years ontology evolution processes have drawn considerable attention of the researchers. The ontology evolution can be considered as the

"timely adaptation of an ontology to the arisen changes and the consistent management of these changes" [14]. This definition suggests that a successful evolution can only be achieved by having both "adaptation" and "change management". In the literature there aren't many approaches capable of handling these two tasks within one framework. Another core aspect of ontology evolution is how to guarantee the consistency of the ontology and the dependent applications [14][15]. In this sense many papers are introducing approaches and methodologies for the ontology evolution management and its change requirement description. In particular frameworks for the management of atomic and complex changes have been introduced [16]. A particular aspect of ontology application is in the analysis and comparison of particular ontologies that could be used to derive information beyond operational data. In this case ontologies could be for management support. This is a very interesting application field but some critical problems remain to be solved. In fact usually the lightweight ontology furnishes a very simple and generic representation of a context and so is not able to well manage a system or supporting users in the interactions with it. In particular if it represents the services and components of a system probably its computational and functional optimization could be not reached. However heavyweight ontology could be very difficult to define and includes some aspects that are not all the time useful and the risk of wasting system's resources to maintain heavyweight ontology is quite high. With the aim to avoid the above described problems, in this paper a lightweight plus ontology is proposed, which can be defined as $O = \{C, A, R_H, R\}$ where C is the concept set, A is the concept attributes set, R_H expresses the hierarchical relationships among the concepts and R is the set of non-hierarchical relations. By the introduction of the non-hierarchical relations, a lightweight plus ontology is more complex and semantically richer than the lightweight ontology, but is not as complex as heavyweight ontology because there are no axioms to consider. The lightweight plus ontology will be the starting point for the representation of the knowledge domain that is involved in the ABB.

III. The Proposed Network Management approach

The aim of this paper is the design and the implementation of a network management tool based upon the slow intelligence system approach. The system follows the architecture showed in figure 1 and in this paragraph more details on the operative workflow will be given. First of all the local server is described: it has the role to collect the information about the faults that are happening in the network and to solve them according the slow intelligence approach. At this aim each local server needs a knowledge domain represented by the following lightweight plus ontologies:

- O_{SNMP_i} $\{C_{SNMP\_i}, A_{SNMP\_i}, R_{HSNMP\_i}, R_{SNMP\_i}\}$: this ontology defines the entire structure of SNMP protocol events' signals that the local server "i" can manage. This ontology is part of a more general ontology O_{SNMP} developed by the analysis of SNMP standard (RFC 1157) and of the related Structure of Managed Information (RFC 2578)
- $O_{Fault\_i} = \{C_{Fault\_i}, A_{Fault\_i}, R_{HFault\_i}, R_{Fault\_i}\}$: this ontology describes each kind of possible errors that can occur within a LAN. This ontology is part of a more general ontology O_{Fault} developed by network manager experts that express also the relationships with the events that are in the O_{SNMP} ontology
- $O_{Cause\_i} = \{C_{Cause\_i}, A_{Cause\_i}, R_{HCause\_i}, R_{Cause\_i}\}$: this ontology defines the causes of the faults that may occur in a LAN. This ontology is part of a more general ontology O_{Cause} developed by network manager experts that express also the relationships with the faults that are in the O_{Fault} ontology
- $O_{Solution\_i} = \{C_{Solution\_i}, A_{Solution\_i}, R_{HSolution\_i}, R_{Solution\_i}\}$: this ontology defines the solutions that can be taken to recover from fault situations which occurred within a LAN. This ontology is part of a more general ontology $O_{Solution}$ developed by network manager experts that express also the relationships with the faults that are in the O_{Fault} ontology
- $O_{Action\_i} = \{C_{Action\_i}, A_{Action\_i}, H_{Action\_i}, R_{HAction\_i}, R_{Action\_i}\}$: this ontology aims to identify the actions to be taken in order to recover from fault's situations. This ontology is part of a more general ontology O_{Action} developed by network manager experts that express also the relationships with the faults that are in the $O_{Solution}$ ontology
- $O_{Component\_i} = \{C_{Component\_i}, A_{Component\_i}, R_{HComponent\_i}, R_{Component\_i}\}$: this ontology describes the components that are within the LAN "i". This ontology has to be developed by the network administrator of LAN "i" that defined also the relationships among the components and the SNMP events.
- $O_{Environment\_i} = \{C_{Environment\_i}, A_{Environment\_i}, R_{HEnvironment\_i}, R_{Environment\_i}\}$: this ontology describes the operative context where the LAN "i" works. This ontology has to be developed by the network administrator of LAN "i" that defined also the relationships among the environment and the SNMP events.

These ontologies represent the knowledge base of each advanced building block. The local server works as depicted in figure y and it acts like a slow intelligence and follows the following phases:

Enumeration Phase: in this phase the Local server tries to find all the actions that can be adopted in order to solve a fault. In particular the input of this stage is the SNMP event and the outputs are the actions that can be adopted. If the event has been managed in the past, the system adopts the previous actions and passes in the concentration phase. If the SNMP event has not been ever managed the enumeration module adopts the following functions:

$$F_{Enumeration}: E \times O_{SNMP\_i} \times O_{Fault\_i} \times O_{Solution\_i} \times O_{Action\_i} \rightarrow A^N$$

where E is the space of SNMP events and A is the space of the actions. In other words this function accepts as input the SNMP event that could be in the O_{SNMP_i} ontology. In this way it is possible, analyzing the ontologies, to find the actions that can lead to the solution of the fault. In general this function gives more than one actions that can be adopted and each of them has an effectiveness grade established by experts. At this point the system can evolve in the adaptation phase. If the function is not able to find an action the propagation phase has to be invoked.

Propagation Phase: in this phase the local server sends the SNMP event to the central server that tries to calculate the actions by the use of the function:

$F_{Enumeration}: E \times O_{SNMP\_CS} \times O_{Fault\_CS} \times O_{Solution\_CS} \times O_{Action\_CS} \rightarrow A^N$

If the central server is able to find the actions, it will send them to the local server "i" and it will send also the parts of ontologies that are needed for the event's resolution. In particular the following function will be invoked:

$S_{Propagation}: E \times O_{SNMP\_CS} \times O_{Fault\_CS} \times O_{Solution\_CS} \times O_{Action\_CS} \rightarrow O^N$

This function sends to the local server "i" the following ontologies

$O'_{SNMP\_CS} \subset O_{SNMP\_CS}$
$O'_{Fault\_CS} \subset O_{Fault\_CS}$
$O'_{Solution\_CS} \subset O_{Solution\_CS}$
$O'_{Action\_CS} \subset O_{Action\_CS}$

these ontologies have to be merged to the ontologies that are in the local server "i" in the following way

$O_{SNMP\_i} = O_{SNMP\_i} \cup O'_{SNMP\_CS}$
$O_{Fault\_i} = O_{Fault\_i} \cup O'_{Fault\_CS}$
$O_{Solution\_i} = O_{Solution\_i} \cup O'_{Solution\_CS}$
$O_{Action\_i} = O_{Action\_i} \cup O'_{Action\_CS}$

If the central server is not able to infer the actions the SNMP event has to be send to the other local servers in order to infer the actions. Also in this case each local server "j" calculates the actions by the use of the following function:

$F_{Enumeration}: E \times O_{SNMP\_j} \times O_{Fault\_j} \times O_{Solution\_j} \times O_{Action\_j} \rightarrow A^N$

and send to the central server the parts of ontologies that need for the actions' inference by the use of the function:

$S_{Propagation}: E \times O_{SNMP\_j} \times O_{Fault\_j} \times O_{Solution\_j} \times O_{Action\_j} \rightarrow O^N$

The central server collects the actions and updates its ontologies according to the previous described method. It sends the actions and the ontologies to the local server "i". If both the central server both the various local servers are not able to infer actions, the local server "i" has to send an error signal to the network administrator. The local server "i" collects the actions and the ontologies from the central server and invokes the adaptation phase.

Adaptation Phase: in this phase the inferred actions has to be customized according to the components that are in the LAN and the environment where the LAN works. In particular for each action the following function is invoked:

$A_{Adaptation}: A \times O_{SNMP\_j} \times O_{Fault\_j} \rightarrow A$

At the end of this phase the system obtains a list of adapted actions. Obviously not all the actions can be adapted and so for each adapted action an improvement for its effectiveness grade is set.

Elimination Phase: the system collects all the actions inferred in the previous phases ranking them according to this function:

$F_{Elimination}: A^N \rightarrow A$

This function can be implemented in various ways according to predefined strategy. In this case the adopted approach is the following:

$F_{Elimination} = \max_{i=1\ldots N}$ effectiveness grade (A_i)

At the end of this phase the concentration phase can be invoked

Concentration Phase: In this phase the local server "i" adopts the selected action. If this action leads to the problem's resolution and comes from the central server the local server "i" updates its ontologies. If the action does not lead to the problem's resolution a message is send to the network administrator that can decide both to adopt one of the other actions retrieved in the other phases both to solve the fault in a manual way.

Summarizing the operational workflow of the system can be described as follows:

Step 1: a SNMP message as result of a fault generated by a LAN's device is sent to a local server "i"

Step 2: The local server "i" receives the SNMP message. This is the beginning of the enumeration phase.

Step 3: The local server "i" tries to identify the problem through analysis of various ontologies that describe its knowledge base. If the SNMP event was managed in the past the concentration phase can start (step 9). Otherwise the system infers a list of actions that can applied for the resolution of the problem and generates the solutions and the actions that the various hosts in the LAN have to be apply. At this point the adaptation phase can start (step 7). If the local server "i" is not able to infer any actions the request is sent to the local server. In this way the propagation phase (step 4) can start.

Step 4: The central server tries to infer the actions that can solve the faults that the SNMP event sent by local server "i" represents. If it is able to find actions the central server sends them and the ontologies parts that are needed for the event management. In this way the adaptation phase (step) can start. Otherwise the central server sends the SNMP events to the other local servers

Step 5: The various local servers try to infer the actions from the received SNMP event. If actions are retrieved each local server sends them and the parts of ontologies needed for their inference.

Step 6: The central server collects the various answers from the local servers and send them local server "i" and the adaptation phase (step 7) can start. If no answers from local servers are received an empty action is sent to the local server.
Step 7: The local server "i" starts to adapt the actions according to the environment and components LAN's ontologies. After this phase the elimination phase can start. If in this phase no actions have been inferred a message to the network administrator have to be sent.
Step 8: The local server "i" selects the action to apply from the other ones collected in the other phases according to a predefined rule.
Step 9: The local server "i" can apply the action in order to recover the fault situation and update, if needed, its ontologies

IV. Experimental Results

In order to test the performance of the proposed system an experimental campaign has been designed. First of all the working scenario has been set: the system has to manage three laboratories during their normal working time. These laboratories are equipped in the following way:

First Laboratory
- **1 Cisco Router Cisco**
- **3 Cisco Catalyst Switches**
- **56 Personal Computers equipped with heterogeneous operative systems and applications**
- **2 Network Printers**

Second Laboratory
- **1 Cisco Router**
- **2 Nortel Switches**
- **40 Personal Computers equipped with heterogeneous operative systems and applications**
- **2 Network Printers**

Third Laboratory
- **1 Cisco Router Cisco**
- **2 Cisco Catalyst Switches**
- **42 Personal Computers equipped with heterogeneous operative systems and applications**
- **1 Network Printers**

In each of these laboratories a local server was settled and the system monitored the three LANs for one week collecting the various SNMP signal and managing the various faults. The local servers have been furnished by the various ontologies and in particular they adopted an O_{SNMP_i} covering about the 60% of concepts of the full O_{SNMP} that can manage about 250 events. Starting from the O_{SNMP_i} the experts has built the others ontologies. The system's performances have been evaluated according various approaches. The first parameter is the following:

$$CA = \frac{SolvedFaults}{Events}$$

The aim of this index is the evaluation of the effectiveness of the system in the resolution of the faults. For the evaluation also the precision and recall parameters has been introduced. The precision is defined in the following way:

$$precision = \frac{TruePositive}{TruePositive + FalsePositive} \qquad recall = \frac{TruePositive}{TruePositive + FalseNegative}$$

These parameters are typically used in information retrieval where a perfect precision score of 1.0 means that every result retrieved by a search was relevant whereas a perfect recall score of 1.0 means that all relevant documents were retrieved by the search. In our case the precision means how many events have been resolved in the correct way (the true positive) respect the number of events that system tried to solve. So in this case a false positive is a fault that the system managed in a wrong way. The recall represents how many events have been resolved in the correct way (the true positive) respect the number of events that system could solve. For the evaluation of the propagation phase the following parameters has been introduced:

$$RCA = \frac{Solved\_Faults\_After\_Central\_Server\_request}{SolvedFaults}$$

In order to evaluate the capacity of the system to share knowledge among the various local servers also the following ontological parameters [17] has been introduced:
- NOC: Number of Concepts in the ontology
- NOL: Number of Leaf Concepts in the ontology
- NONHR: Number of "non-hierarchical" relationships
- NOF: Number of Fanouts
- AF-C: Average Fanout per Class
- MaxDIT: Maximum Depth of Inheritance Tree

All these parameters have been evaluated each 24 hours for each local server "i" and an average value has been expressed for the evaluation of the system. The obtained CA, Precision, Recall and RCA are depicted in the appendix of this paper as the ontological parameters for the full O_{SNMP} ontology and the various O_{SNMP_i} ontologies. The obtained results confirm the effectiveness of the proposed approach. The system allows an effective sharing of knowledge among the various servers as the ontological parameters show. In fact after about the 30% of the managed events each system reaches good results in their management and the local servers improve their knowledge domains. The system shows a very good CA result and more in general after a first training phase achieves very good performances both from the precision both from the recall point of view. It is important to underline that the various local ontologies show a less complex structure of the full ontology and so, in this way, it is easier to manage them.

V. Conclusion

In this paper a novel method for network management has been introduced. This method is based on Ontology and Slow Intelligence System approach. It has been tested in an operative scenario and the first experimental seems to be good. In particular the proposed approach showed how an ontology based interoperability framework can help to improve several tasks in the network management value chain. The proposed approach introduces a powerful way for the improvement of the information model interoperability and allows the introduction of services for the automatic resolution of networks fault. The opportunity to continuosly upgrade the knowledge base allows to continuosly upgrade the capacity of the system to manage new faults. In particular network administrators will not only benefit from more powerful

applications, but they can transfer their expert knowledge into the management applications and in this way automating more and more network management tasks. The future works aim to improve the system by the use of new and effective methodologies for the ontology management and the use of some artificial intelligence approach for the automatic inference of action when the system is not able to find anyone.

REFERENCES

[1] Hui Xu, Debao Xiao, "A Common Ontology-based Intelligent Configuration Management Model for IP Network Devices", Proceedings of the First International Conference on Innovative Computing, Information and Control, pp. 385-388, 2006

[2] López de Vergara J.E., Guerrero A., Villagrá V.A., Berrocal J., Ontology Based Network Management: Study Cases and Lessons Learned, Computer Science Journal of Network and Systems Management, Volume 17, Number 3, pp. 234-254, 2009

[3] Wong A.K.Y., Ray P., Parameswaran N., Strassner J., Ontology mapping for the interoperability problem in network management, IEEE Journal on Selected Areas in Communications, 23(10), 2058-2068, 2005

[4] López de Vergara J.E., Villagrá V.A., Asensio J.I., Berrocal J., Ontologies: giving semantics to network management models, IEEE Network, Volume 17, Number 3, pp. 15-21

[5] Keeney J., Lewis D., O'Sullivan D., Roelens A., Boran A., Richardson R., Runtime semantic interoperability for gathering ontology-based network context, Proceedings of IEEE/IFIP Network Operations and Management Symposium (NOMS 2006), 56-65,Vancouver, 2006

[6] Van der Meer, S., Jennings B., O'Sullivan D., Lewis D., Agoulmine N., Ontology based policy mobility for pervasive computing, Proceedings 12th Workshop of the HP Open University Association, 2005, 211-224

[7] Shi-Kuo Chang, "A General Framework for Slow Intelligence Systems", International Journal of Software Engineering and Knowledge Engineering, Volume 20, Number 1, February 2010, pp. 1-15.

[8] Jepsen, T., Just What Is an Ontology, Anyway?, IT Professional, vol. 11, no. 5, pp. 22-27, Sep./Oct. 2009

[9] Gruber, T.R, Translation approach to portable ontology specification, Knowledge Acquisition, vol. 5, pp. 199-220, 1993

[10] "OWL Web Ontology Overview", W3C Recommendation, 10 february 2004, http://www.w3.org/TR/2004/REC-owl-features-20040210/

[11] Maedche A., Staab S., Ontology Learning for the Semantic Web, IEEE Intelligent Systems, vol. 16 no. 2, Mar/Apr 2001, Page(s): 72-79

[12] Corcho, O., A Layered Declarative Approach to Ontology Translation with Knowledge Preservation, Vol. 116 Frontiers in Artificial Intelligence and Applications, 2005

[13] Corcho, O., Gómez-Pérez, A., A Layered Model for Building Ontology Translation Systems, International Journal on Semantic Web and Information Systems, 1(2): 22-48, 2005.

[14] Haase, P., Stojanovic, L., Consistent evolution of OWL Ontologies, ESWC, Lecture Notes in Computer Science, vol. 3532, Springer

[15] Yinglin Wang, Xijuan Liu, Rongwei Ye, "Ontology Evolution Issues in Adaptable Information Management Systems," E-Business Engineering, IEEE International Conference on, pp. 753-758, 2008

[16] Zhang, L., Xia, S., Zhou, Y., Xia, Z., User Defined Ontology Change and its Optimization, Control and Decision Conference, 2008. CCDC 2008. Chinese, pp. 3586-3590, 2008

[17] Orme, A.M., Haining, Y., Etzkorn, L.H., Indicating Ontology Data Quality, Stability and Completeness Throughout Ontology Evolution, Journal of Software Maintenance and Evolution: Research and Practice, vol. 19, pp. 49-75, 2007

[18] Stallings, W., SNMP, SNMPv2, SNMPv3, and RMON1 and 2, Addison-Wesley, Reading, MA, 1999

[19] Case, J., Mundy, R., Partain, D., Introduction and Applicability Statements for Internet Standard Management Framework, RFC 3410, 2002, ftp://ftp.rfc-editor.org/in-notes/rfc3410.txt

APPENDIX

	SNMP_Events_1	SNMP_Events_2	SNMP_Events_3	SNMP_Events_4	SNMP_Events_5	SNMP_Events_6	SNMP_Events_7	Total
Local_Server_1	1423	1562	1233	1321	1401	720	603	8263
Local_Server_2	1378	1492	1541	1647	1230	601	532	8421
Local_Server_3	1401	1351	1321	1678	1345	654	472	8222

	CA_1	CA_2	CA_3	CA_4	CA_5	CA_6	CA_7	Average_Value
Local_Server_1	60,01%	77,40%	90,84%	91,07%	91,29%	95,83%	98,84%	84,12%
Local_Server_2	64,73%	73,93%	84,23%	87,37%	96,50%	97,67%	98,31%	83,47%
Local_Server_3	60,17%	75,94%	87,51%	89,27%	96,21%	97,55%	98,94%	84,19%

	Precision_1	Precision_2	Precision_3	Precision_4	Precision_5	Precision_6	Precision_7	Average_Value
Local_Server_1	72,01%	85,93%	91,58%	95,55%	96,75%	98,29%	99,67%	90,31%
Local_Server_2	72,05%	85,04%	92,78%	94,98%	96,98%	99,83%	99,81%	90,29%
Local_Server_3	73,05%	84,65%	88,11%	97,08%	98,40%	99,69%	99,79%	90,55%

	Recall_1	Recall_2	Recall_3	Recall_4	Recall_5	Recall_6	Recall_7	Average_Value
Local_Server_1	86,61%	93,29%	95,40%	96,94%	99,22%	100,00%	100,00%	95,59%
Local_Server_2	85,03%	91,61%	94,68%	96,25%	97,70%	98,49%	99,81%	94,30%

Local_Server_3	83,38%	89,30%	95,54%	98,23%	98,85%	99,07%	99,36%	94,59%

	RCA_1	RCA_2	RCA_3	RCA_4	RCA_5	RCA_6	RCA_7	Average_Value
Local_Server_1	22,13%	12,90%	8,93%	7,15%	3,36%	3,33%	2,01%	8,76%
Local_Server_2	17,60%	9,16%	5,62%	3,89%	2,36%	1,53%	0,19%	6,05%
Local_Server_3	24,08%	15,89%	10,64%	5,21%	3,17%	2,51%	0,86%	9,07%

Table 1 Obtained Results

SNMP_Full_Ontology	
NOC	378
NOL	250
NONHR	9
NOF	1732
AF_C	4,58
MaxDIT	7

O_{SNMP_i}	NOC_1	NOC_2	NOC_3	NOC_4	NOC_5	NOC_6	NOC_7
Local_Server_1	189	223	278	289	302	315	318
Local_Server_2	176	218	267	281	298	306	311
Local_Server_3	183	197	253	278	293	301	305

O_{SNMP_i}	NOL_1	NOL_2	NOL_3	NOL_4	NOL_5	NOL_6	NOL_7
Local_Server_1	150	172	184	195	200	203	204
Local_Server_2	158	171	181	190	198	202	203
Local_Server_3	143	162	176	184	191	193	194

O_{SNMP_i}	NONHR_1	NONHR_2	NONHR_3	NONHR_4	NONHR_5	NONHR_6	NONHR_7
Local_Server_1	6	7	8	8	8	9	9
Local_Server_2	5	6	7	8	8	9	9
Local_Server_3	7	8	9	9	9	9	9

O_{SNMP_i}	NOF_1	NOF_2	NOF_3	NOF_4	NOF_5	NOF_6	NOF_7
Local_Server_1	754	903	1163	1201	1278	1332	1389
Local_Server_2	730	930	1034	1175	1208	1299	1326
Local_Server_3	734	892	979	1078	1189	1256	1301

O_{SNMP_i}	AF_C_1	AF_C_2	AF_C_3	AF_C_4	AF_C_5	AF_C_6	AF_C_7
Local_Server_1	3,99	4,05	4,18	4,16	4,23	4,23	4,37
Local_Server_2	4,15	4,27	3,87	4,18	4,05	4,25	4,26
Local_Server_3	4,01	4,53	3,87	3,88	4,06	4,17	4,27

O_{SNMP_i}	MaxDIT_1	MaxDIT_2	MaxDIT_3	MaxDIT_4	MaxDIT_5	MaxDIT_6	MaxDIT_7
Local_Server_1	7	7	7	7	7	7	7
Local_Server_2	7	7	7	7	7	7	7
Local_Server_3	7	7	7	7	7	7	7

Table 2 Ontological Parameters for the full O_{SNMP} and the various O_{SNMP_i}

Extending Software Quality Models - A Sample In The Domain of Semantic Technologies

Filip Radulovic
Ontology Engineering Group
Departamento de Inteligencia Artificial
Facultad de Informática, Universidad Politécnica de Madrid
Madrid, Spain
fradulovic@delicias.dia.fi.upm.es

Raúl García-Castro
Ontology Engineering Group
Departamento de Lenguajes y Sistemas Informáticos
e Ingeniería Software
Facultad de Informática, Universidad Politécnica de Madrid
Madrid, Spain
rgarcia@fi.upm.es

***Abstract*—In order to correctly evaluate semantic technologies, which have become widely adopted in recent years, we need to put evaluations under the scope of a unique software quality model. This paper presents a quality model for semantic technologies. First, some well-known software quality models are described, together with methods for extending them. Afterwards, a new method for extending quality models is proposed and it is then used to define a quality model for semantic technologies by extending the ISO 9126 quality model. Finally, the proposed model is validated by analyzing existing semantic technology evaluations.**

I. Introduction

Software product quality has become an important concern in almost every domain or technology, and the specification and evaluation of quality during the software development process is of crucial importance for obtaining high quality software [1].

Quality models provide the basis for software evaluation and give a better insight of the characteristics that influence software quality by specifying a consistent terminology for software quality and by providing guidance for its measurement. Nevertheless, in order to use a quality model in a specific domain, it usually has to be extended to include the particularities of such domain.

Various methods for extending quality models have been proposed in the literature. They all follow a top-down approach, starting from general characteristics to concrete measures; for some cases, however, a bottom-up approach would be more effective as is the case of those which have many of evaluations to extract the quality model.

An example of this occurs in the semantic technology field. Semantic technologies provide new ways to express in machine processable formats knowledge and data that can be exploited by software agents. We have seen an exponential growth of semantic technologies in recent years and multiple evaluations of such technologies have been proposed, from general evaluation frameworks [2] to tool-specific evaluations [3], [4] and even characteristic-specific evaluations [5].

However, it is very difficult to compare semantic technologies because of the different evaluation characteristics used. Furthermore, there is no consistent terminology for describing the quality of semantic technologies, and available software quality models do not specify the quality characteristics specific to them.

This paper describes a bottom-up approach for specifying a software quality model by extending an existing one. Using this approach, we have defined a quality model for semantic technologies, starting from real semantic technology evaluations and extending the ISO 9126 quality model.

Clearly, during the definition of the quality model not every available evaluation can be taken into account. To validate and complete the quality model, we have performed a literature review over those semantic technology evaluations presented in the most relevant conferences in the semantic research field.

The reminder of this paper is organized as follows. Section II gives an overview of existing software quality models. Section III describes top-down methods for extending software quality models, while Section IV presents the bottom-up method that we have defined. Section V describes how we have applied such method to define a quality model for semantic technologies. Section VI presents the validation of the quality model and, finally, Section VII draws some conclusions and includes ideas for future work.

II. Review of Software Quality Models

Various software quality models have been described in the literature: McCall's model, Boehm's model, ISO 9126 model, and SQuaRE model. This section describes the models most used and identifies their main elements.

ISO 9126's Model. The International Organization for Standardization (ISO) identified the need for a unique and complete software quality standard and, therefore, produced the ISO 9126 standard for software quality [6]. The ISO 9126 standard defines three types of quality: internal quality, external quality, and quality in use.

Six main software quality characteristics for external and internal quality are specified: functionality, reliability, usability, efficiency, maintainability, and portability, all of which are further decomposed into sub-characteristics that are manifested externally when the software is used and are the result of internal software attributes [6]. The standard also provides the internal and external measures for sub-characteristics.

Regarding quality in use, the model proposes four characteristics: effectiveness, productivity, safety, and satisfaction.

The ISO 9126 standard gives a complete view of software quality with evaluation criteria and definitions of every software characteristic and sub-characteristic. Some authors also suggest that according to the nature of the product itself some new sub-characteristics can be added, the definitions of existing ones changed, or some sub-characteristics can be eliminated from the model [7].

However, as pointed out in [8], some practical problems with ISO 9126 arise, namely, the ambiguity in metric definitions and usability interpretation. Furthermore, the authors argue that the number of attributes and measures are missing, that some characteristics are too abstract, and that the standard itself is open to interpretations, which, according to the authors, questions its purpose.

SQuaRE's Model. Because of advances in technologies and changes of users' needs over time, some problems and issues have arisen with the ISO 9126 standard. Therefore, it is currently being redesigned and has been renamed SQuaRE (System and software Quality Requirements and Evaluation). At the time of writing this paper, the parts of the SQuaRE standard related to the quality model and evaluations are still under development (ISO 25010 and ISO 25040 respectively) and their final versions will be published during 2011.

As a summary of this section, Table I presents the elements of the quality models mentioned. In our work, and in the rest of the paper, we have adopted the terminology of the ISO 9126 standard.

TABLE I: Elements of mentioned quality models.

Structure/Model	McCall	Boehm	ISO 9126
First level	Factor	High level characteristic	Characteristic
Second level	Criteria	Primitive characteristic	Sub-characteristic
Third level	Metrics	Metrics	Measures
Relationships between entities	Factor-Metric	/	Measure-Measure

III. Approaches for Extending Software Quality Models

Existing software quality models (e.g., the ISO 9126 one) provide insight into characteristics that are general and common for almost every type of software. However, different types of software products have characteristics which are specific to them and, therefore, the actual application of software quality models usually requires reusing an existing quality model and extending it for a specific software product or domain.

To this end, starting from a certain quality model, its quality characteristics and sub-characteristics should be adapted according to the nature of the domain by excluding those out of scope, redefining others, and introducing new ones.

Some authors have proposed software quality models for various types of applications: B2B [1], mail servers [9], web-based applications [10], e-learning systems [11], and ERP systems [7]. All those authors have used the ISO 9126 standard as the basis software quality model, and have extended it to fit their particular domain.

Software quality model extensions can be performed following two main approaches [12]:

- A **top-down** approach that starts from the quality characteristics and continues towards the quality measures.
- A **bottom-up** approach that starts from the quality measures and defines the quality sub-characteristics that are related to each specific measure.

Franch and Carvallo proposed a method based on a top-down approach for customizing the ISO 9126 quality model [13]. After defining and analyzing the domain, their method proposes six steps:

1) *To determine quality sub-characteristics*. In this first step, according to the domain, some new quality sub-characteristics are added while other are excluded, or their definitions are changed.
2) *To define a hierarchy of sub-characteristics*. If needed, sub-characteristics are further decomposed according to some criteria.
3) *To decompose sub-characteristics into attributes*. In this step, abstract sub-characteristics are decomposed into more concrete concepts that refer to some particular software attribute (i.e., observable feature).
4) *Decomposing derived attributes into basic ones*. Attributes not directly measurable are further decomposed into basic ones.
5) *To state relationships between quality entities*. Relationships between quality entities are explicitly defined. Three possible types of relationships are identified: a) *collaboration* means that increasing the value of one entity implies increasing the value of another entity; b) *damage* means that increasing the value of one entity implies decreasing the value of another entity; and c) *dependency* implies that some values of one entity require that another entity fulfills some conditions.
6) *To determine metrics for attributes*. To be able to compare and evaluate quality, it is necessary to define metrics for all attributes in the model.

In building their quality model for B2B applications, Behkamal et al. proposed a method to customize the ISO 9126 quality model in five steps [1]. The main difference with the previous method is that in Behkamal's approach the quality characteristics are ranked by experts; thus, the experts should provide weights for all quality characteristics and sub-characteristics and these weights are later used to establish their importance, which can be time consuming and resource demanding. Besides, Behkamal's approach does not contemplate defining relationships between quality entities.

IV. A Bottom-Up Approach For Extending Software Quality Models

The approaches presented in the previous section follow a top-down approach and, at the time of writing this paper,

we have not found any example of a bottom-up approach in the literature. However, there are scenarios where it would be convenient to base on real practices the extension of the quality model because of the existence of a significant body of software evaluations and evaluation results (as in the case of the semantic technology field).

In our approach, evaluation results are used as the starting point from which the quality measures, sub-characteristics, and characteristics are specified.

The method for extending a software quality model consists in performing the following six consecutive steps:

1) *To identify basic measures*. The output of evaluating a software product with some input data (i.e., executing a test case) allows identifying the basic measures of a certain evaluation execution.
2) *To identify derived measures*. Basic measures can be combined to obtain derived ones, which are also related to one particular evaluation execution (i.e., test case).
3) *To identify quality measures.* Quality measures are measures related to a whole evaluation (i.e., multiple test cases using different input data) and are obtained by the aggregation of basic and derived measures.
4) *To specify relationships between measures*. In this step, which can be performed in parallel with the previous ones, relationships between measures are expressed either in an informal way (e.g., the collaboration, damage and dependency categories proposed in [13]) or more formally (e.g., with the formulas used for obtaining the measures, as proposed in [14]). For any derived measure defined it is recommendable to specify the function (or set of functions) that allows obtaining such derived measure from the basic ones. Similarly, for any quality measure it is also recommendable to identify the function that defines it based on other measures.
5) *To define domain-specific quality sub-characteristics*. Every software product from a particular domain has some sub-characteristics that are different from other software products and those sub-characteristics, together with more generic ones, should be identified and precisely defined. Every quality measure provides some information about one or several software sub-characteristics; therefore, based on the software quality measures defined in the previous step, software quality sub-characteristics are specified. Furthermore, it is not necessary that every quality sub-characteristic has only one measure that determines it, but rather a set of measures. Finally, if needed, some quality sub-characteristics can be combined into more general ones.
6) *To align quality sub-characteristics with a quality model*. In this step, the alignment with an existing quality model is established; i.e., the software quality sub-characteristics that have been previously defined are related to others already specified in the existing model. Depending on the domain and nature of the software product, some new quality characteristics can be specified, or existing ones can be modified or excluded.

V. Defining A Quality Model For Semantic Technologies

This section describes the definition of a software quality model in the domain of semantic technologies by following the bottom-up method presented in the previous section.

A. Identifying Basic Measures

The starting point for defining software quality measures has been the set of evaluation results obtained in the SEALS project[1], which has produced evaluation results for different types of semantic technologies (ontology engineering tools [15], reasoning systems [16], ontology matching tools [17], semantic search tools [18], and semantic web service tools [19]).

For each type of technology, different evaluation scenarios were defined, using in each of them different test data as input. In this step we identified the basic measures of each evaluation scenario (i.e., those outputs directly produced by the software during the evaluation).

Due to space reasons, we cannot present details about all evaluation scenarios. Therefore, we just present the outcomes of each step for one concrete scenario, that of evaluating the conformance of ontology engineering tools.

Different test suites are used for evaluating the conformance of ontology engineering tools, which are composed of different test cases each containing

- *Origin ontology*. The ontology to be used as input.

A test case execution consists in importing the file containing an origin ontology (O_i) into the tool, and then exporting the imported ontology to another ontology file (O_i^{II}), as shown in Fig 1.

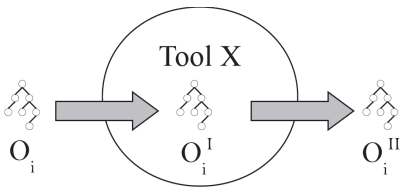


Fig. 1: Steps of a conformance test execution.

The basic measures obtained after a test case execution are

- *Final ontology*. The ontology that is produced by the tool when importing and exporting the origin ontology.
- *Execution Problem*. Whether there were any execution problems in the tool when importing and exporting the origin ontology.

B. Identifying Derived Measures

Based on the test data and the basic measures of one test execution, the following derived measures were specified:

- *Information added*. The information added to the origin ontology after importing and exporting it.
- *Information lost*. The information lost from the origin ontology after importing and exporting it.

[1] http://www.seals-project.eu

- *Structurally equivalent*. Whether the origin ontology and the final one are structurally equivalent.
- *Semantically equivalent*. Whether the origin ontology and the final one are semantically equivalent.
- *Conformance*. Whether the ontology has been imported and exported correctly with no addition or loss of information.

C. Identifying Quality Measures

From the derived measures in the conformance scenario, the following quality measures were obtained:

- *Ontology language component support*. Whether the tool fully supports an ontology language component.
- *Ontology language component coverage*. The ratio of ontology components that are shared by a tool internal model and an ontology language model.
- *Ontology information persistence*. The ratio of information additions or losses when importing and exporting ontologies.
- *Execution errors*. The ratio of tool execution errors when importing and exporting ontologies.

Similarly to the example of the conformance evaluation presented above, we have defined measures for the other types of tools. Table II summarizes the results obtained.

TABLE II: Number of measures obtained for semantic technologies.

Tool\Measures	Basic	Derived	Quality
Ontology engineering tools	7	20	8
Ontology matching tools	1	3	4
Reasoning systems	7	0	8
Semantic search tools	12	11	21
Semantic web service tools	5	10	11
Total	27	40	50

D. Specifying Relationships Between Measures

We have identified the relationships between measures in a formal way by defining the formulas used for obtaining derived and quality measures.

For example, the formula for the *Information added* derived measure calculates the structural difference between the origin and final ontologies:

$$\text{Final ontology} - \text{Origin ontology}$$

Similarly, the formula for the *Execution errors* quality measure calculates the percentage of tests with execution problems:

$$\frac{\text{\# tests where Execution problem = true}}{\text{\# tests}} \times 100$$

E. Defining Domain-Specific Quality Sub-characteristics

In this step, from the quality measures previously identified, we defined the set of quality sub-characteristics that are affected by those measures. In some cases we were able to reuse existing quality sub-characteristics but, in others, we had to define domain-specific ones.

In the conformance evaluation scenario, based on the measures and analysis presented above, we have identified three quality sub-characteristics:

- *Ontology language model conformance*. The degree to which the knowledge representation model of a software product adheres to the knowledge representation model of an ontology language. It can be measured with two different quality measures: *Ontology language component coverage*, and *Ontology language component support*.
- *Ontology processing accuracy*. The accuracy of the process of importing and exporting ontologies. It can be measured with *Ontology information persistence*.
- *Ontology processing robustness*. The ability of the software product to process ontologies correctly in the presence of invalid inputs or stressful environmental conditions. It can be measured with *Execution errors*.

Fig. 2 presents the basic measures, derived measures, quality measures, and quality characteristics of the conformance evaluation for ontology engineering tools.

In total, we have identified twelve semantic quality sub-characteristics. Three of them are those described for the conformance evaluation and the others are the following:

- *Ontology language interoperability*. The degree to which the software product can interchange ontologies and use the ontologies that have been interchanged.
- *Reasoning accuracy*. The accuracy of the reasoning process.
- *Ontology alignment accuracy*. The accuracy of the matching process.
- *Semantic search accuracy*. The accuracy of the semantic search process.
- *Semantic web service discovery accuracy*. The accuracy of the process of finding services that can be used to fulfill a given requirement from the service requester.
- *Ontology interchange accuracy*. The accuracy of the interchange of ontologies between tools.
- *Ontology processing time behaviour*. The capability of the software product to provide appropriate response and processing times when working with ontologies.
- *Reasoning time behaviour*. The capability of the software product to provide appropriate response and processing times when performing reasoning tasks.
- *Semantic search time behaviour*. The capability of the software product to provide appropriate response and processing times when performing search tasks.

Apart from these domain-specific quality sub-characteristics, we have identified the following general ones: *Operability*, *Productivity*, and *Satisfaction*.

Finally, we have also identified those sub-characteristics that are contained into others (e.g., *Reasoning time behaviour* is a sub-characteristic of *Ontology processing time behaviour*).

F. Aligning Quality Sub-Characteristics with a Quality Model

As ISO 9126 has been used by a number of authors, as mentioned in Section III, we have also adopted it for constructing the quality model for semantic technologies.

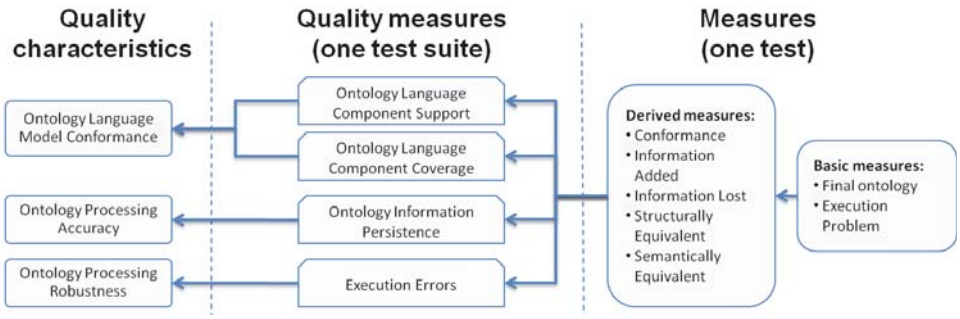


Fig. 2: Entities in the conformance scenario for ontology engineering tools.

In the previous step we have identified the set of quality sub-characteristics specific for semantic technologies. In this step, all the identified sub-characteristics were properly assigned to the ones that already existed in the ISO 9126 quality model.

For instance, *Ontology language model conformance* is defined as a sub-characteristic of *Functionality compliance* (i.e., the capability of the software product to adhere to standards, conventions or regulations in laws and similar prescriptions relating to functionality).

VI. Quality Model Validation

Since we started from a specific set of evaluations in order to define the quality model for semantic technologies, we performed a literature review to validate the quality model and to complete it if needed. The review was performed according to the procedures described in [20] and, due to space reasons, we will only present the final results.

We have analysed the proceedings of the two most relevant conferences in the semantic area: the International Semantic Web Conference (nine editions) and the European Semantic Web Conference (seven editions) to identify those publications that deal with semantic technology evaluation. We focused on publications that describe evaluation methods or suggest measures for evaluation as well as on publications that suggest new algorithms (e.g., for reasoning or semantic web service discovery) that are also evaluated.

In total, we have analysed fifty seven publications. Table III shows an overview of this analysis including, for each type of semantic technology, the evaluation measures used. Every evaluation measure is classified according to the quality sub-characteristics that our model describes and the number of occurrences is shown in brackets.

From the analysis we can observe that the quality model that we have proposed is quite complete regarding current semantic technology evaluations. Almost all the measures described in the publications fit the quality characteristics that our model describes. However, some measures found did not fit our model, and therefore we have defined new quality characteristics for them. These are

- *Semantic web service time behaviour*. The capability of the software product to provide appropriate response and processing times when performing semantic web service discovery tasks.

TABLE III: Measures used in conference publications.

Ontology engineering tools (2)
Ontology processing robustness: execution (1) *Ontology processing time behaviour*: execution time (1) *Ontology import/export accuracy*: information added/lost (1)
Ontology matching tools (21)
Ontology alignment accuracy: precision (19), recall (19), f-measure (13), measure at cut-off point (1)
Reasoning and storage systems (18)
Reasoning time behaviour: classification time (5), execution time (5), reasoning time (5), entailment time (1), labeling time (1), lattice operation time (1), justification time (1) *Ontology processing time behaviour*: loading time (4) *Reasoning accuracy*: reasoner errors (1), correct results (7), wrong classifications (1), fitness value (1)
Semantic search tools (5)
Semantic search time behaviour: query execution time (2), speed (1) *Semantic search accuracy*: recall (4), precision (3), reciprocal rank (1), f-measure (1), relevance (1) *Ontology processing time behaviour*: loading time (1) *Operability*: usability (2)
Semantic web service tools (11)
Semantic web service discovery accuracy: precision (12), recall (8), f-measure (1), returned sources (1), bpref (1), reciprocal rank (1)

- *Matching time behaviour*. The capability of the software product to provide appropriate response and processing times when performing matching tasks.

Fig. 3 shows an overview of the quality model for semantic technologies after completing it during the validation.

VII. Conclusions and Future Work

This paper presents a new method for extending software quality models, which is based on a bottom-up approach. It starts from existing evaluations and continues defining quality measures and quality sub-characteristics, which are aligned with an existing quality model.

In practice, the quality model to be extended is taken into account from the beginning of the method, even if the alignment to such quality model is the last step of the method. Therefore, it seems natural to follow a hybrid approach, which combines the bottom-up and top-down approaches, and a future extension of the method should also cover this approach.

We have used the method for defining a quality model for semantic technologies, which extends the ISO 9126 software quality model. Such quality model can provide a framework for the evaluation and comparison of semantic technologies.

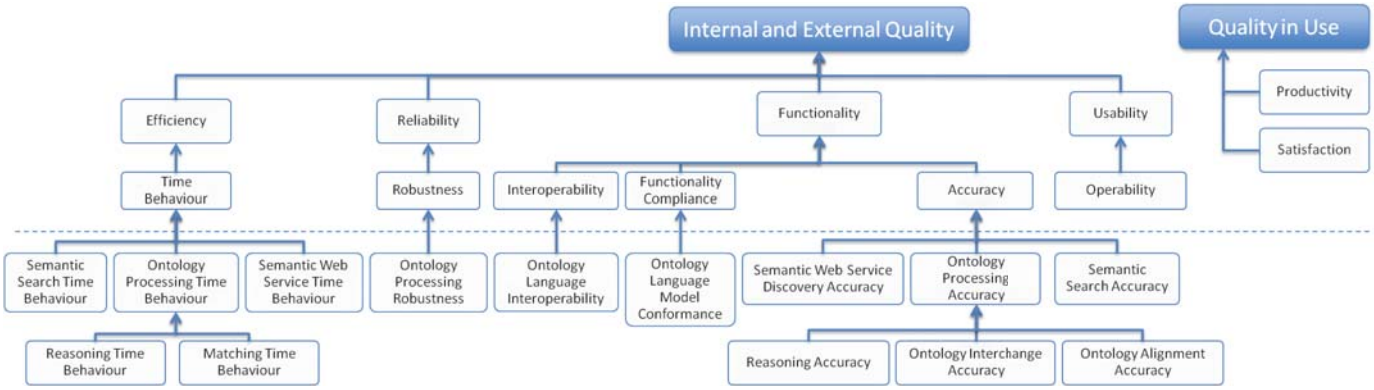


Fig. 3: External and internal quality characteristics for semantic technologies.

Although some problems with ISO 9126 have been identified (as described in [8]), we have introduced quality measures specific to semantic technologies, and we have also specified functions for all derived and quality measures, which result in reducing ambiguities in our model.

Furthermore, we can note that our quality model is easily extensible and that new quality measures or characteristics can be easily introduced and categorized, as shown during the validation process.

During such validation, we have concentrated only on the most relevant conferences in the semantic field. In order to get more complete results, we plan to extend our analysis to other conferences, as well as to relevant journals. This will help us to further extend and validate the model.

The ISO 9126 standard is being replaced by the SQuaRE standard; when the SQuaRE software quality model becomes available, the proposed quality model for semantic technologies will be adapted to it.

A future use of the proposed quality model, based on the evaluation results that are being obtained in the SEALS project, is to build a recommendation system for semantic technologies that will allow extracting semantic technology roadmaps. Such a system will provide users with guidance and recommendation of the semantic technologies that better suit their needs.

Acknowledgments

This work is supported by the SEALS European project (FP7-238975) and by the EspOnt project (CCG10-UPM/TIC-5794) co-funded by the Universidad Politécnica de Madrid and the Comunidad de Madrid.

Thanks to Rosario Plaza for reviewing the grammar of this paper.

References

[1] B. Behkamal, M. Kahani, and M. Akbari, "Customizing ISO 9126 quality model for evaluation of B2B applications," *Information and software technology*, vol. 51, no. 3, pp. 599–609, 2009.

[2] OntoWeb, "Ontoweb deliverable 1.3: A survey on ontology tools," IST OntoWeb Thematic Network, Tech. Rep., May 2002.

[3] Y. Guo, Z. Pan, and J. Heflin, "LUBM: A benchmark for OWL knowledge base systems," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 3, no. 2-3, pp. 158–182, 2005.

[4] P. Lambrix, M. Habbouche, and M. Perez, "Evaluation of ontology development tools for bioinformatics," *Bioinformatics*, vol. 19, no. 12, p. 1564, 2003.

[5] R. García-Castro and A. Gómez-Pérez, "Interoperability results for Semantic Web technologies using OWL as the interchange language," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 8, pp. 278–291, November 2010.

[6] ISO, "ISO/IEC 9126-1:2001, Software engineering – Product quality – Part 1: Quality model," International Organization for Standardization, Tech. Rep., 2001.

[7] P. Botella, X. Burgués, J. Carvallo, X. Franch, J. Pastor, and C. Quer, "Towards a quality model for the selection of ERP systems," *Component-Based Software Quality*, pp. 225–245, 2003.

[8] H. Al-Kilidar, K. Cox, and B. Kitchenham, "The use and usefulness of the ISO/IEC 9126 quality standard," in *2005 International Symposium on Empirical Software Engineering, 2005.* IEEE, 2005, p. 7.

[9] J. Carvallo, X. Franch, and C. Quer, "Defining a quality model for mail servers," in *COTS-based software systems: second international conference, ICCBSS 2003, Ottawa, Ont., February 10-13, 2003: proceedings*. Springer-Verlag New York Inc, 2003, p. 51.

[10] H. Zulzalil, A. Ghani, M. Selamat, and R. Mahmod, "A Case Study to Identify Quality Attributes Relationships for Web-based Applications," *IJCSNS*, vol. 8, no. 11, p. 215, 2008.

[11] I. Padayachee, P. Kotze, and A. van Der Merwe, "ISO 9126 external systems quality characteristics, sub-characteristics and domain specific criteria for evaluating e-Learning systems," in *The Southern African Computer Lecturers' Association, University of Pretoria, South Africa*, 2010.

[12] R. Dromey, "Software Product Quality: Theory, Model, and Practice," *Software Quality Institute, Brisbane, Australia*, 1998.

[13] X. Franch and J. Carvallo, "Using quality models in software package selection," *Software, IEEE*, vol. 20, no. 1, pp. 34–41, 2003.

[14] M. Bombardieri and F. Fontana, "A specialisation of the SQuaRE quality model for the evaluation of the software evolution and maintenance activity," in *Automated Software Engineering-Workshops, 2008. ASE Workshops 2008. 23rd IEEE/ACM International Conference on*. IEEE, pp. 110–113.

[15] R. García-Castro, S. Grimm, I. Toma, M. Schneider, A. Marte, and S. Tymaniuk, "D10.3 Results of the first evaluation of ontology engineering tools," SEALS Consortium, Tech. Rep., 2010.

[16] M. Yatskevich and A. Marte, "D11.3 Results of the first evaluation of advanced reasoning systems," SEALS Consortium, Tech. Rep., 2010.

[17] J. Euzenat, C. Meilicke, C. Trojahn, and O. Šváb Zamazal, "D12.3 Results of the first evaluation of matching tools," SEALS Consortium, Tech. Rep., 2010.

[18] S. N. Wrigley, K. Elbedweihy, D. Reinhard, A. Bernstein, and F. Ciravegna, "D13.3 Results of the first evaluation of semantic search tools," SEALS Consortium, Tech. Rep., 2010.

[19] S. Tymaniuk, L. Cabral, D. Winkler, and I. Toma, "D14.3 Results of the first evaluation of Semantic Web Service tools," SEALS Consortium, Tech. Rep., 2010.

[20] B. Kitchenham, "Procedures for performing systematic reviews," *Joint Technical Report NICTA Technical Report 0400011T1*, vol. 33, 2004.

A Technology of Profiling Inter-procedural Paths

Lulu Wang, Bixin Li

School of Computer Science and Engineering, Southeast University, Nanjing, China
Key Lab of Computer Network and Information Integration (Southeast University), Ministry of Education
Email: {wanglulu, bx.li}@seu.edu.cn

Abstract

This paper presents a novel approach (called PIP) for profiling inter-procedural paths, which makes extensions of previous work to inter-procedural situations. An inter-procedural profiling model, PCCG (Polymorphic Cluster Call Graph), is presented, which describes polymorphism, divides procedures into clusters, and improves profiling efficiencies. Theoretical analysis and experimental results show that different clustering strategies of PCCG have different advantages (lower cost in instrumentation or in execution) can meet custom needs. Compared with existing work, PIP is more practical as it profiles more cyclic paths, deals with polymorphism, and is totally accurate.

***Keywords*—Path profiling; inter-procedural paths; polymorphism; dynamic analysis;**

I. Introduction

Path profiling records the frequency of each executed path. It was introduced by Ball and Larus [1] in *Efficient Path Profiling* (EPP), which could enable the collection of profiles for acyclic intra-procedural paths at a reasonable cost. Since then, path profiles have been extensively used in a wide variety of areas such as computer architecture, compilers, debugging, program testing, and software maintenance. But same as EPP, most of these extensions work in single-procedure situations, that is, with inter-procedural paths still untreated. However, it is often desirable to obtain frequency counts of paths that extend across procedure boundaries [2].

Supported partially by the National Natural Science Foundation of China under Grant No. 60773105 and no. 60973149, and partially Supported by Doctoral Fund of Ministry of Education of China under Grant No. 20100092110022, and partially by National High Technology Research and Development Program under Grant No.2008AA01Z113.

Correspondence to: Bixin Li, School of Computer Science and Engineering, Southeast University, Nanjing, China. E-mail: bx.li@seu.edu.cn

Melski et al. have proposed an approach for profiling inter-procedural paths. They connect the flows of all procedures together and use EPP to enumerate the paths in this combined flow graph [3]. But this method can only profile paths without any backedges in any procedures.

Tallam et al. propose an approximate profiling which can deal with backedges, and extend this approach to profile inter-procedural paths [2]. But this method is not completely accurate and has severe restrictions on loops.

So in a word, existing inter-procedural profiling methods can only profile paths without backedges accurately, or profile certain cyclic paths inaccurately. In this paper, we gives an extension of PAP (*Profiling All Paths*) [4] to address the problem of collecting accurate profiles for inter-procedural paths (including loop iterations and procedure calls), which is called PIP (*Profiling Inter-procedural Paths*). PIP algorithm is based on an inter-procedural control flow model, PCCG (*Polymorphic Cluster Call Graph*), which describes polymorphic relations.

The rest of this paper is organized as follows: Section II illustrates the preliminaries; Section III discusses the details of PIP; Section IV gives experimental evaluations; Section V summarizes related work and Section VI concludes the paper.

II. Preliminaries

This section gives necessary basis. First EPP is used to explain the basic methodology of path profiling, then PAP is illustrated in details since PIP is quite related to it.

A. EPP

In order to collect path profiles, existing techniques use probe variable(s), that is, they instrument probe statements for the variable(s) into the program under profiling, and these statements are executed as the program runs. When

an execution is over, the value of the variable(s) (so called *pathid*) will determine the path of the execution. This profiling process requires a unique *pathid* for each path to make profiles correct and accurate.

EPP uses one integer variable r, and its probes are in the form of "$r := r + i$", where "i" is the edge **weight**, computed by BL algorithm [1]. Fig.1(a) shows how EPP computes the edge weights in a DAG (*Directed Acyclic Graph*, "N_a" means the number of paths from node "a"). EPP gives each acyclic path a unique *pathid*, so that the probe value of variant "r" can identify the corresponding path after the execution.

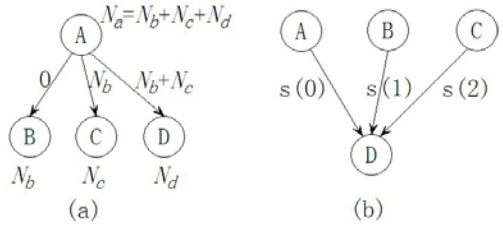


Fig. 1. Probes of EPP and PAP

B. PAP

In order to profile all paths (especially cyclic paths), we have presented PAP algorithm, which instruments probes in the form of "$r := r * s + i$" ("$s(i)$" for short, s is called **weight** and i is called **remainder**) on the inedges of each CFG node N, where s is the indegree of N, and i is the serial number of the inedge, as Fig.1(b) shows (single-inedge nodes don't need probes). After executions, using the probe value of r we can acquire the whole path by iteratively finding out the former node and restoring the former probe value (this phase is called *backwalk*).

Different from EPP, PAP does not need to change backedges, and can accurately profile both acyclic and cyclic paths.

In order to deal with the probe-value overflowing that may happen during executions, a *breakpoint* mechanism is devised to use more storage to handle longer paths, i.e., when overflowing occurs, the position and probe value before overflowing (called a *breakpoint*, which contains an integer and a node), are recorded, and then the probe variable is reset and current execution is continued. As studied in [4], with several *breakpoints* for each path, PAP can profile paths that are much longer and with more loops.

III. Inter-procedural Profiling

PIP's steps can be briefly shown in Fig.2: code splitting is used to handle polymorphism, as discussed in Section III.A; procedure clustering and PCCG is used to reduce the cost of code splitting, as discussed in Section III.B; instrumentation and optimization of probes are discussed in Section III.C; program execution and *backwalking* in PIP make no differences to those in PAP, and are not specially discussed here. For convenient explanation, intra-procedural structures is ignored for PIP, since those can be handled by PAP.

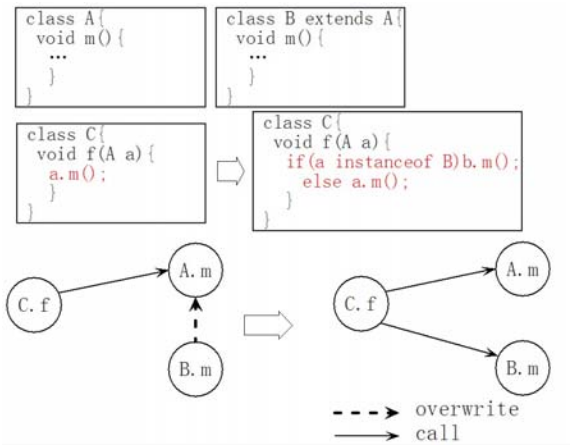


Fig. 3. An example of code splitting

A. Code Splitting

In polymorphic situations, every call edge, where polymorphism may happen, can be equally replaced by several edges, which each stands for an actual call relation that possibly occurs in some executions. By such replacement, a polymorphic inter-procedural CFG can be transferred into the same form of an intra-procedural CFG. As Fig.3 shows, class B extends class A, and overwrites A.m(). C.f() calls A.m(), and in dynamic scenarios, B.m() may be called instead of A.m(). To deal with such a polymorphic call, we can simply split the call statement. As shown in Fig.3, in the code of C.f(), the input parameter is verified and the target callee procedure can be precisely fixed. And correspondingly, the call edge is divided into two ones.

B. PCCG and Procedure Clustering

Though code splitting can handle polymorphic relations, it is still hard to perform when the inheritance relations are very complicated, as one call may need to be replaced by many calls. It is thought to make procedures classified into clusters, and some code splitting may be saved if the callees fall into one cluster.

In order to describe the polymorphic information which uses one procedure to replace another in executions, this

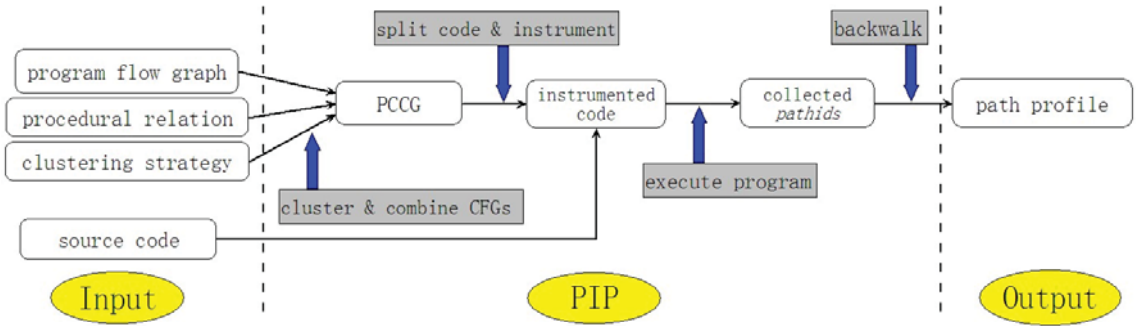


Fig. 2. Framework of PIP

paper presents a new method-call model, PCCG, which is a directed graph, and consists of three sets, CS, PS and CE:

- CS is the cluster set, where each cluster consists of at least one procedure;
- PS is the procedure set, where each procedure is contained by one cluster;
- CE is the set of call edges (including return edges), where each edge points from a procedure to a cluster.

PCCG can be generated based on a call graph and PE (polymorphic edges), according to following steps: first, the potential call edges are added to PCCG based on PE (like how Fig.3 does); then the procedures with polymorphic relations are clustered and redundant edges are removed; at last, corresponding return edges are added to the graph.

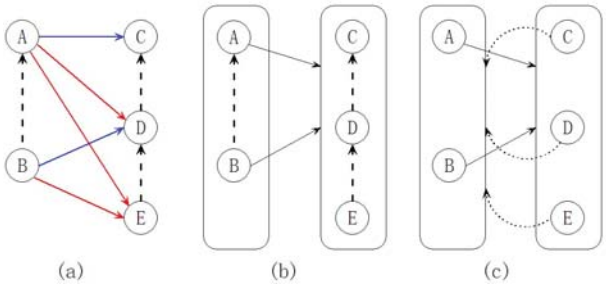


Fig. 4. A PCCG example

Fig.4 shows an example of how to generate a PCCG. Fig.4(a) has five procedures (each is a single-procedure cluster), and originally, there are only two calls, (A, C) and (B, D). Based on three polymorphic relations (described as dotted arrows) in it, three potential call edges are put on the graph, (A, D), (A, E) and (B, E). Fig.4(b) classifies all five procedures into two clusters, and removes redundant call edges. After that, Fig.4(c) adds return edges on the graph.

Based on PCCG, procedure clustering can be performed, which leads to less code splitting because calling procedures in one cluster does not need to be split.

C. Instrumentation and Optimization

1) Instrumentation: As given in [4], PAP instruments all inedges of each CFG node sequentially. Here on PCCG, PIP instruments all inedges of each cluster in the same way, which also ensures a unique *pathid* for each inter-procedural path. The difference is, in PAP, paths consist of edges from node to node in CFG; while in PIP, paths consist of edges from procedure to cluster in PCCG. Such a difference does not affect the accuracy, because the latter path representation can precisely identify the former one.

2) Optimization: In a PCCG, each call edge points from a procedure to a cluster, so does each return edge. For a procedure in the callee cluster, if all its return edges have the same probe on them, then these probes can be replaced by only one probe, which is at the end of the procedure instead of on return edges. This replacement does not influence the execution of probes along any path, but reduces the probe amount and makes instrumentation easier to perform.

To make more probes replaceable, if a callee cluster has multi return edges, the weight of probes on these edges is increased, then the remainders of them could be reassigned same if possible.

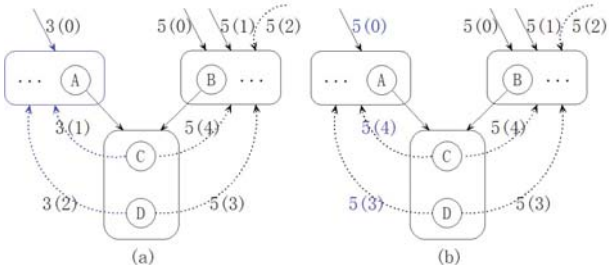


Fig. 5. An example of optimization

An example is shown in Fig.5 to explain such an optimization. Fig.5(a) displays three clusters, and the callee

cluster contains two procedures, C and D, which are called by A and B. The cluster of A has one other inedge, and the cluster of B has three other inedges (call edges and return edges). Since C and D return into both A and B, A has three inedges in all and the weight of probes on them is 3, while B has five and the weight is 5. The probe on the return edge (C, A) is 3(1), and the probe on (C, B) is 5(4). To make the probes on (C, A) and (C, B) same, 5 is used as the uniform weight, and the probes can be reassigned as Fig.5(b) shows, where all return edges from C has the same probe 5(4). After that, this probe can be put to the end of C, and only one probe is needed instead of two. So does procedure D. By such a process, weights "spread" along the return edges from a procedure, and the amount of probes gets reduced.

From all discussed above, PIP algorithm is given in Algorithm 1.

Input: PS : the set of procedures
CE : call edges
PE : polymorphic edges
sc : the cluster strategy
Output: ps : the probe set
/* generate the PCCG */
use sc to cluster procedures and get CS;
initialize a PCCG with CS, PS and CE;
remove redundant call edges and add return edges to the PCCG;
/* compute probes */
foreach *cluster* c *in* CS **do**
    int $weight$ = c.fanIn(); //the indegree of c
    int i = 0;
    foreach *inedge* $e \rightarrow c$ **do**
        ps.addProbe(e, $weight$, i++);
    end
end
/* optimize probes on return edges */
foreach *procedure* p *in* PS **do**
    if p *returns into more than one cluster and can be optimized* **then**
        spread the weight to make return edges from p have the same probe, assumed to be (w, i);
        foreach *return edge* e *from* p **do**
            ps.remove(e);
        end
        ps.add($p.exit$, w, i);
    end
end

Algorithm 1: PIP

D. The Trade-Off

From former parts of this section, PIP has been presented, which works on PCCGs, combines procedures into clusters, and brings following benefits.

First, it reduces probes by optimizing their placements; second, it saves code splitting when polymorphic calls exist; third, it generates a chance of custom selection: the bigger the clusters are, the easier instrumentation is; but on the other hand, the more storage is needed since the *pathids* get bigger as the weights grow.

So in clustering there is a trade-off between the convenience of instrumentation and the storage cost of executions: fewer probes bring more storage.

IV. Experiment

As Fig.2 shows, for a program under profiling, PIP needs the clustering strategy to perform. So here experiments are carried out to watch the performances of different strategies.

A. Measured Aspects

PIP's overhead be classified into three categories, *static cost* (the cost for instrumentation), *storage cost* (the cost for storing *pathids*), and *dynamic cost* (mainly influenced by the *backwalk* cost). Section III.D displays a trade-off between *static cost* and *storage cost*. Here we explore the relationship between *static cost* and *dynamic cost*.

For a strategy used by PIP, its *static cost* can be measured by the probe number (which also represents the *storage cost*) and code splitting; its *dynamic cost* can be measured by the number of unique *breakpoints* since fewer unique *breakpoints* lead to less *backwalk* cost since duplicate *breakpoints* only need to be *backwalked* once.

So in order to test the efficiencies of different strategies, we collect their probe amounts on the same PCCG and the number of unique *breakpoints* on the same paths.

B. Tested Strategies

Some clustering strategies are presented and measured in experiments:

- sa: each cluster contains a single procedure;
- sb: two procedures in one cluster if they have polymorphic relations;
- sc: all procedures are in one cluster;
- $s(n)$: two procedures are in one cluster if they are commonly called by n or more procedures.

Obviously, $s(1)$, sb and sc can avoid code splitting, and $s(k)$ needs no more code splitting than $s(k+1)$.

sa, $s(2)$, $s(3)$, $s(4)$, $s(5)$, sb, sc are chosen to be experimentally inspected on probe numbers.

C. Experimental Results

To perform these strategies, we randomly generate PCCGs (sized of 100 procedures) and paths based on three parameters:

- RP: the ratio of calls between each two procedures;
- NR: the number of polymorphic relations;
- APL: the average length of random paths.

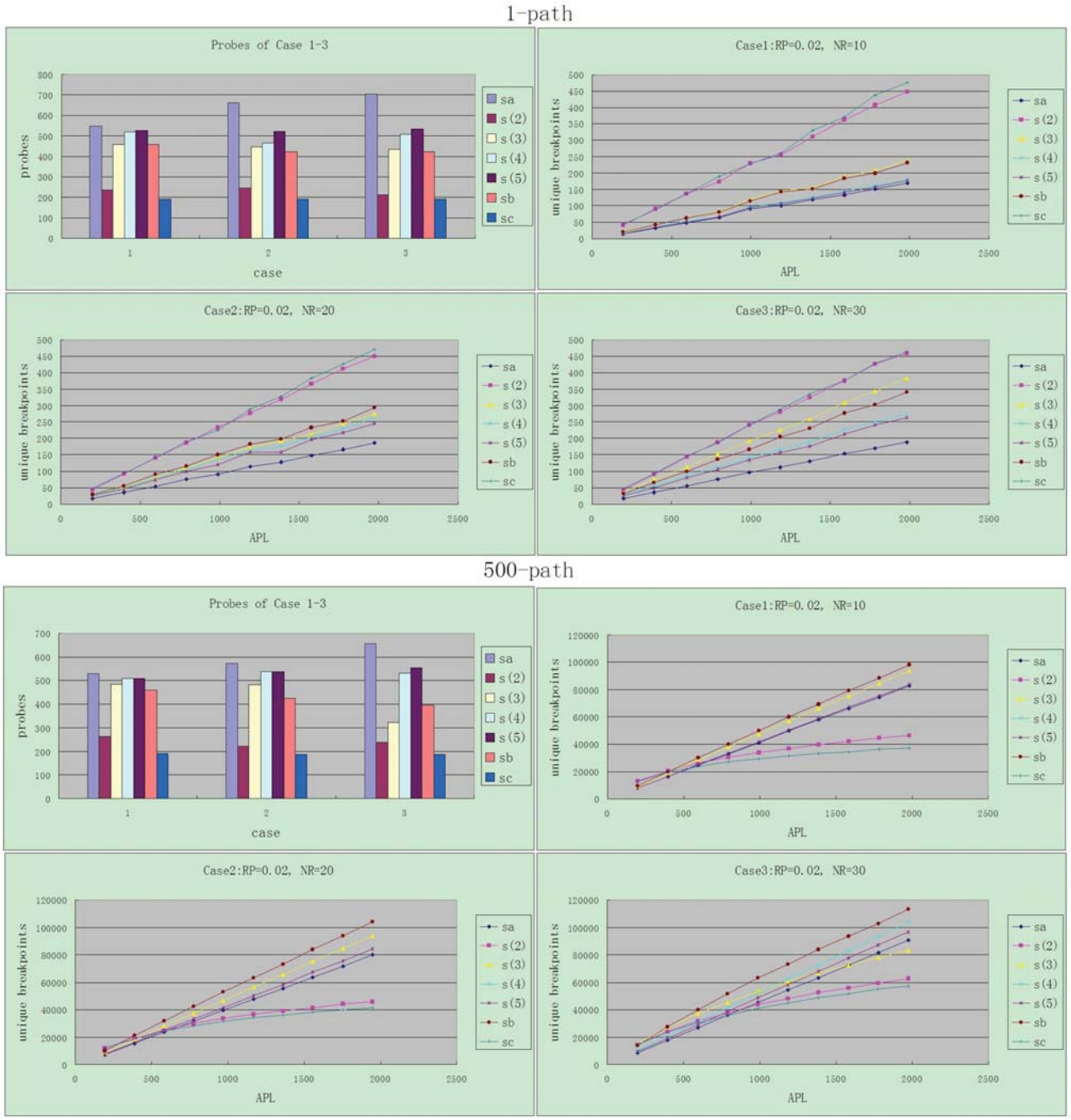


Fig. 6. Experimental results of breakpoints with 1 path and 500 unique paths

Based on random PCCGs and random paths, we count the probes needed by each strategy given above, and collect the unique *breakpoints* of the paths.

With different assignments of parameters, the experimental results of probe numbers and unique *breakpoints* are shown in Fig.6, which contains two parts, each deals with one path and 500 unique paths.

D. Analysis

From the results in Fig.6, we can draw following observations and conclusions.

First, as in 1-path situation, the number of unique *breakpoints* increases almost linearly as the path lengths grow;

TABLE I. Comparison of related work

Techniques	[3]	[2]	PIP
Cyclic	no	yes	yes
Completely accurate	yes	no	yes
Polymorphism handled	no	no	yes
Overflowing handled	no	no	yes

Second, generally in 1-path cases, if a strategy needs fewer probes, then it needs more unique *breakpoints*; but in case 3 of 1-path situation, compared with $s(3)$, sb uses fewer probes and still fewer unique *breakpoints*, which means it is possible to find a strategy which performs well in both aspects;

Third, as in 500-path situation, the growths of unique *breakpoints* of some strategies, such as sc, show obvious nonlinearity. So it is known sc is more efficient than others in both *static cost* and *dynamic cost*. Furthermore, it is inferred that with longer unique paths executed, PIP can use less average *dynamic cost* on a certain length; and compared with 1-path situation, with more paths executed, less average *dynamic cost* on each path.

V. Related work

As we know, during various profiling techniques, only three papers [2, 3, 5] involve inter-procedural profiling.

Ammons et al. present *Flow Sensitive Profiling* and *Context Sensitive Profiling*, which are combined together to provide an efficient approximation for inter-procedural path profiling [5].

Different from [5], [3] truly profiles paths with call and return edges. It also makes an optimization on "multi calls" to avoid profiling on some invalid paths.

In [2], Tallam et al. have extended their profiling into inter-procedure scenarios, which can profile *overlapping paths*. But though with high precision, their method is approximate, and still has limitations on loops. On the storage cost, a four-dimensional array is used to represent inter-procedural overlapping paths, so it may cost much space in complicated scenarios.

Moreover, neither of them consider polymorphic relations.

PIP extends PAP into inter-procedural situations, and can profile all paths with finite loops accurately (with enough storage). It divides procedures into clusters, and perform an optimization in instrumentation, which can reduce the amount of probes and improve the efficiency.

As shown in Table.I, PIP is compared with the approaches of [2, 3] in details. On one hand, PIP' ability is higher than other two techniques, i.e., PIP can profile longer paths as well as paths with more loop iterations; On the other hand, profiling longer paths leads to bigger cost, as bigger *pathids* need more storages.

VI. Conclusion

Focused on profiling inter-procedural paths, this paper has presented a profiling model PCCG and the PIP algorithm. PIP can handle paths with loops, across procedural boundaries, and under polymorphic situations, i.e., it gives an accurately profiling for cyclic inter-procedural paths, which would be useful in program understanding, profile-directed optimizations, and so on. By optimizations on procedure clustering and probe locations, it is easier to perform and achieves higher efficiency.

From theoretical analysis and experimental results based on random PCCGs and paths, we can draw following conclusions:

- Bigger clusters, which need less code splitting and fewer probes, result in less *static cost*;
- A trade-off exists between *static cost* and *storage cost*;
- With certain strategies used, PIP's *dynamic cost* is more efficient as more and longer paths are executed: its growth is obviously sublinear as the number of unique path and the average path length increase;
- Different clustering strategies for clustering give different benefits.

Furthermore, as discussed in Section IV.D, a proper strategy needs to be chosen based on the three types of cost. Although four types of strategies are tested in experiments, it is still a further problem to inspect more types, and to explore an algorithm for searching optimal strategies.

References

[1] T. Ball and J. R. Larus. Efficient path profiling. In: *International Symposium on Microarchitecture (MICRO)*, 1996. 46-57

[2] S. Tallam, X. Zhang, and R. Gupta. Extending path profiling across loop backedges and procedure boundaries. In: *International Symposium on Code Generation and Optimization (CGO)*, 2004. 251-264

[3] D. Melski, T. W. Reps. Interprocedural Path Profiling. In: *Proceedings of the 8th International Conference on Compiler (CC)*, 1999. 47-62

[4] L. Wang, B. Li, X. Zhou. Profiling of All Paths. Submitted to JSS.

[5] G. Ammons, T. Ball, J. R. Larus. Exploiting Hardware Performance Counters with Flow and Context Sensitive Profiling. In: *SIGPLAN conference on Programming language design and implementation (PLDI)*, 1997. 85-96

Efficiency and portability: guidelines to develop websites

Cleriston Araujo Chiuchi, Rogéria Cristiane Gratão de Souza, Adriana Barbosa Santos, Carlos Roberto Valêncio
Computer Science and Statistics Departament
UNESP – São Paulo State University
São José do Rio Preto, São Paulo, Brazil
cleriston_cac@hotmail.com, {rogeria, adriana, valencio}@ibilce.unesp.br

***Abstract*— The use of the Internet as a means of ensuring greater visibility for products, services and information offered by companies is gaining strength in recent decades. However, it is known that to ensure satisfaction and subsequent virtual customer loyalty, it is necessary to guarantee the quality of the websites, allowing indiscriminate access regardless of the resources used, as well as rapid responses to possible requests. In order to assist this process, this paper presents a set of guidelines for the development of websites having quality characteristics, efficiency and portability as per ISO 9126 norms. An observational analysis of e-commerce websites was done which showed that they are inadequate as to the proposed guidelines, making them difficult to access available content. Therefore, the adoption of the proposed guidelines can greatly contribute to increasing the quality of websites and, consequently, enable quick and effective access regardless of the resources used.**

Keywords - websites quality assurance; websites development guidelines; efficiency and portability characteristics

I. INTRODUCTION

Internet use has grown amazingly and is increasing in the number of new users. This has encouraged many companies to migrate the publicity of their products and services to the Internet, seeking access to a larger public. However, this entails the need to guarantee the users indiscriminate access to documents, products and services, regardless of where they are and the device used. With this scenario, it is necessary to develop new techniques to help the process of quality assurance, since the websites have unique features not offered by traditional software engineering.

This article then presents a set of guidelines that contributes to the creation of websites with emphasis on quality characteristics of efficiency and portability, as defined in ISO / IEC 9126 [1], including the need to ensure agility and indiscriminate access to information.

This article is organized as follows: section 2 presents related work in software quality; section 3 presents the guidelines established in this work; section 4 presents the results of an observational analysis undertaken in the context of Brazilian e-commerce websites in relation to proposed guidelines; and finally section 5 presents the conclusion.

II. RELATED WORK

Quality is an important factor for any software, regardless of their characteristics and restrictions. The ISO / IEC 9126 [1] established a standard of quality characteristics to be measured against during the development of software systems. However, the Web environment is an even greater challenge because of its unique characteristics, spawning various works ([2] [3] [4] [5] [6] [7]) which seek to establish elements that will ensure the quality of a website project. In this context, it is observed that quality efficiency and portability represent key aspects in websites, since their quality and interaction experience may be jeopardized by various factors, such as access device, means of access and implementation of specific features present in websites.

Works related to efficiency usually focus on usability, i.e., although important, efficiency is often not dealt with explicitly in works that cover the quality theme in websites ([8] [9] [10], [11]). The use of mobile devices to access websites makes it necessary to adapt the websites to the hardware characteristics of those devices [10]. Thus, the work related to portability seeks to minimize such problems ([12], [13], [14],[15]), often caused by lack of concern for quality when websites are developed.

Because of the above, this paper focuses on providing a set of guidelines in order to direct the development of quality websites, with emphasis on efficiency and portability characteristics, as defined in ISO / IEC 9126 [1]. With this, it is possible to improve the quality of websites, helping to ensure user satisfaction and guarantee indiscriminate access to information.

III. GUIDELINES FOR DEVELOPING WEBSITES

Creating quality websites is a challenge to designers and developers. To define the guidelines presented in this study, current information regarding the use of websites was gathered in order to assist the developers in the creation, with available resources, of a website with quality that meets today's demands. In this context, we emphasize that today, although it is possible to access the Web through various types of networks and different connection speeds, the Internet speed available to most of the population is less than 56 Kbps [9]. In Brazil, the newest CETIC [16] research points out that about 34% of the population has an Internet connection of up to 256 Kbps, a fact that causes a number of problems to access some websites, because it can create a barrier between the user and application, substantially hindering access to useful information. Against this scenario, in establishing guidelines, this study considered only three speed levels: below 256 kbps; from 256 kbps to 1 Mbps; and above 1Mbps.

The presentation of each of the proposed guidelines was conducted according to a defined set of relevant information in

order to facilitate their understanding and use. This combination is shown schematically in Figure 1 and detailed below. It should be noted that for a given category, different guidelines can be established.

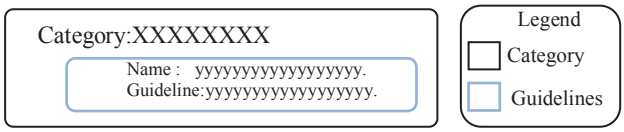


Figure 1 Structure of the guidelines.

Category: reflects the group that belongs to the established guideline(s).

Name: assigns a unique identifier for the corresponding guideline, based on the context of the metric considered as a basis [3].

Guideline: defines rules to guide the design of websites as a way to contribute to the production of results that meet the requirements of portability and efficiency.

In the following subsections, the guidelines established to aid in the development of websites following the proposed structure are presented.

3.1 Category: Media

The use of media to interact with the user in the form of tutorials or newsletters has been an element used in websites as a means of assisting users in the performance of a task.

a) Name: Duration.

Guideline: Due to the high volatility of Internet users, videos should not be too long, i.e., the recommended duration should be between two and ten minutes [17], to ensure effective execution and portability to different access devices.

b) Name: Response time.

Guideline: As to efficiency, the size of the videos and audios can be defined by taking into account three factors: the first is that a user needs to receive a response from the website up to one second after the click [6], i.e., the user must be able initiate the execution of video or audio until one second; the second factor is the use of streaming data, which is the ability to play media while downloading the rest of the file; finally, the last factor is to consider the connection speed of the user. Thus, in order to provide a satisfactory interaction with the user, it is necessary that the download of a new second of media be possible for each second done. Therefore, the media size is defined by the equation 1:

$$Average\ size = download\ per\ second\ X\ display\ time \quad (1)$$

In this context, based on a time range of two to ten minutes from the previous guideline and the proposed levels of access speed considered in this work, you can establish the following maximum size of videos and audios described in Table 1. In the item portability, it should be noted that most mobile devices that are used to access videos and audios do not have great processing power and storage capacity. Cell phones, for instance, seldom have an internal memory size greater than 100 MB. Thus, the adoption of media to support the use of a website should be done with caution so as to always consider the size of such media related to desired portability.

Table 1. Maximum size of videos and audios

	Media	
Speed	2 minutes	10 Minutes
up to 256 kbps	up to 3,840 KB	up to 19,200 KB
between 256 kbps and 1 Mbps	between 3,840 KB and 17,520 KB	between 19,200 KB and 87,600 KB
above 1 Mbps	from 17,520 KB	from 87,600 KB

c) Name: Image.

Guideline: Whereas a user can access information from a website from a mobile device, more precisely with less processing power such as cell phones, some precautions should be taken in relation to the use of images. Therefore, to improve the portability of the website, we suggest the preferred use of JPEG and GIF in an attempt to ensure a better user experience. To improve efficiency, we recommend that the resolution be set correctly inside the tags, specifying the value of the image that is loaded by the browser to prevent the resizing of images loaded by the browser [17]. As an example of the effectiveness of this policy, considering the use of HTML, the implementation could be done by adding the width and height attributes within the HTML image tag, as follows:

<img src="img.gif" width="28" height="21" >

The use of actual image values of width and height helps the browser to build the page properly, improving aspects of portability and efficiency of websites related to different devices and thus improving user experience.

3.2 Category: Maximum access time

The access time to a website represents an element that will be part of the quality evaluation of the user, directly influencing the user's decision about whether or not to continue using that website.

Name: Loading rate

Guideline: The metrics considered in this guideline are heavily dependent on the maximum download time that the user is willing to wait to see the website or any additional element. An additional feature is defined as being any external component from the page the user wants to see, for example, downloading a text file, an image with higher resolution, etc. Nielsen [11] defines ten seconds as being the maximum time users will wait before they decide to choose another page. Taking into account this time and the download speed, it is possible to calculate what the recommended size of a page or additional element of the website should be. The page size is the sum of all the sizes of each element that composes it, or is the sum of all images, texts, scripts, animations, and other elements that make up the website page. You can calculate the maximum size with the following equation 2:

$$Maximum\ Size = download\ speed\ X\ maximum\ waiting\ time \quad (2)$$

There are videos that do not need a complete loading when opening a page, since the total size of the video can be accessed by streaming if the user so wishes. Therefore, the size of the videos only represents the size of the application that will play them. In developing websites focused entirely on

mobile phones, it is recommended that pages have sizes of up to 20 KB [15]. As for the other devices, based on levels of access speed considered in this work, you can establish the following maximum sizes for a page described on Table 2:

Table 2. Maximum size for a page

Speed	Size
up to 256 kbps	up to 320 KB
between 256 kbps and 1 Mbps	between 320 KB and 1,280 KB
above 1 Mbps	from 1,280 KB

3.3 Category: Interface

The quality of the interface of a website aims to facilitate its use by the user through the organization of its established features.

a) Name: Text.

Guideline: It is recommended that websites be developed to make available a version where the images are not loaded by the browser whenever the user so wishes. This guideline is motivated by two factors: the first is the inability of some browsers to display the images and the second is to reduce the amount of downloaded data, thus reducing the access time to websites [18]. Moreover, it is necessary to allow the user to browse the page without any loss, even without viewing the images. As an example of how to put this guideline into effect, considering the use of HTML, the implementation of a text-only version could be done by adding the alt image tag within the HTML, as follows:

< img alt=”Name of the picture” ...>

With this, instead of viewing the image, the user will only see the information according to the tag. This is a feature that provides benefits to access websites via mobile devices, since it reduces the amount of information to be accessed, and consequently stored, without affecting the usability of the website.

b) Name: Title.

Guideline: The use of titles for all images of the website helps the overall readability of the site and its use is recommended.

c) Name: Reading.

Guideline: The overall reading of the site is the user's ability to understand the information contained in it. The overall readability of the site, even when it is not displaying images, should remain unchanged. Therefore, the guidelines related to the text-only version and inclusion of titles in the images must be implemented, so that the user has no trouble in understanding the information even when not viewing the images.

d) Name: Compactness.

Guideline: The navigation map is a representation in the form of a tree structure of possible paths to be followed in a website. Map compacting considers the number of levels required to gain access to pages that have the desired information. It is known that the less effort made by the user to access the information he wants, better will be the compactness of the navigational map. Therefore, the amount of levels established in this guideline for the maximum size of the navigational map, takes into account that the pages have used the maximum loading rate proposed in the maximum access time guideline. In addition, three other facts were considered: 1) studies show that users should be able to perform simple tasks within a website in one minute [10], 2) the time visiting a website lasts from one to two minutes, so the user must be able to reach the target website within that time or he will give up [19], 3) a user would spend, on more than 50% of the times, at least ten seconds to see each page of the website [19]. Therefore, considering that of the two minutes (average maximum time of one visit) the user can use up to one minute to perform the proposed task, we can then calculate the maximum amount of pages that can be visited before reaching the last one through the following equation 3:

$$Number\ of\ pages = \frac{Downlod\ Time + Reading\ Time}{Remaining\ time\ of\ the\ visit.} \quad (3)$$

The remaining time is the time of the visit less the time spent performing the task the user wants to accomplish. With that, the maximum use of up to three pages is indicated for the desired task to be accomplished. The high compactness of the navigational map contributes to the ease of access to websites through mobile devices, since they do not always have elements of interaction, such as mouse or keyboard.

e) Name: Scroll**.**

Guideline: The need for scrolling or rolling of the website page should be avoided. Studies show that the user will choose, in more than 75% of the times, a link which is available on the interface after the loading time. Therefore, the developer should avoid the use of long pages. Mainly for cell phones, this should be done due to the small size of their screens.

f) Name: Words.

Guideline: Users of websites tend to read only a little of the information contained on a page, and the more information available, the less will the user read. Studies show that users read at least 50% of the information contained on page that has 111 words or less [8]. Above this amount, the percentage of words read falls. Therefore, the use of short texts, with an option to expand them if the reader so desires, is recommended [8]. The use of few words improves visualization of websites when accessed by mobile devices, since they have very small screens.

g) Nome: Scripts and flash.

Guideline: The use of scripts and flash on websites influences the user’s experience in different ways, depending on the means of access used. The use of scripts can be very useful to developers, providing a powerful tool which can often improve elements related to the usability of websites. However, the use of scripts and the use of flash animation must be seen through a different perspective when developing an application that will be accessed via mobile devices. The different browsers used to access websites do not always have today’s technologies so can cause serious problems, preventing the user from accessing the desired information. Therefore, one should avoid incorporating scripts and flash on items considered essential in the website.

h) Name: Map.

Guideline: The use of a navigational map, when downloading several pages of the website, helps the user to quickly locate the desired information without the need for extensive searching. Accessing the website through networks that have a low speed or high cost per KB, the use of the navigational map can reduce the time needed to reach the desired information.

i) Name: Dimension.

Guideline: Setting the size of the resolution established for a monitor may affect the experience that the user will have with the website. According to a study conducted by Market net share[1], about 21% of Internet users use a 1024x768 resolution and 17% a 1280x800. Due to this, the use of one of the two aforementioned resolutions is recommended to better serve users. In the category of mobile devices, specifically for mobile phones, the instruction is to use pages having a width of at least 120 pixels. Currently most mobile browsers implement functions that facilitate the viewing of websites with resolutions greater than this value [19].

j) Name: Frame.

Guideline: A frameset is a tag used in the HTML that gives the developer the division of the website in multiple frames, including the loading of other pages within each of these frames. When the Web started, the use of frames to create sites was very common, but due to problems related to navigation and location of pages by search engines, frames were gradually forgotten and currently only a few websites still use them. Therefore, it is recommended not to use frames in a website.

k) Name: Search.

Guideline: It is recommended that a search engine on the website be created for users to be able to quickly look for the content they want without the need for an exhaustive search through the site.

IV. WEBSITES ANALISIS

The Internet has changed the way how companies conduct their business; it has become increasingly vital to their success. In recent years, e-commerce has grown and now moves billions of dollars in the global economy. In Brazil, e-commerce profit grew 30% in 2009 and peaked at 6.3 billion dollars[2]. According to [20], 33% of the people already use mobile phones to buy goods over the Web and 47% to compare prices of products before purchasing. Another important fact is that users prefer to access the websites available on www instead of applications developed specifically for mobile phones.

Given the importance of the use of Internet for dissemination of company products and the need to develop quality websites accessible through any device, which are decisive factors for customer satisfaction, this article examined five Brazilian e-commerce websites: Americanas.com (www.americanas.com.br); Submarino (www.submarino.com.br) ; NetShoes (www.netshoes.com.br); Compra Fácil (www.comprafacil.com.br); Saraiva (www.saraiva.com.br). The purpose of this analysis was to verify the impact of the adequacy, or not, to guidelines which aim to ensure indiscriminate access to information on websites, thus increasing the visibility of products and services and ensuring user satisfaction.

The analysis was done to rate total adequacy (T), partial (P) or inadequacy (I) of websites regarding the proposed guidelines, using the following access devices: a desktop, a laptop and a smartphone. To do this, we adopted that for the guidelines that have a set of values that vary according to the provided resources, we analyze the narrower range of values. Thus, the established connection speed for analysis was up to 256 kbps, which is the most used in Brazil. Moreover, it was observed that the browser used by smartphones and notebooks did not display images. Results are shown in Table 3.

Table 3. Web Site Analysis

Guidelines	Analyzed Websites				
	Americanas	Subamrino	Netshoes	Compra Fácil	Saraiva
Media					
01- Length	T	T	T	T	-
02- Size	I	I	I	I	-
03- Images	P	I	I	T	I
Maximun access time					
04- Loading Rate	I	I	I	I	I
Interface					
05- Text	P	P	P	P	P
06- Title	T	T	T	T	T
07- Reading	T	P	P	T	T
08- Compactness	I	I	I	I	I
09- Scroll	I	I	I	I	I
10- Word	I	I	I	I	I
11- Scripts and flash	I	I	I	I	I
12- Map	I	I	I	I	T
13- Dimension	I	I	I	I	I
14- Frame	T	T	T	T	T
15- Search	T	T	T	T	T

4.1 Media

For this category, we found that this feature is beginning to be explored by Brazilian e-commerce websites as a way to demonstrate the benefits offered by their products. Furthermore, we observed that, generally, video viewing is done through a video portal available on the Internet, which makes access via smartphone difficult. Among the five analyzed websites, four had videos which only rated total adequacy for the "Length" guideline. Moreover, the guideline "Size" was not followed in any of the analyzed cases so, consequently, the efficiency of devices used for accessing was lacking. As for the "Images", only one of the websites had total adequacy to the guideline. The remaining images had been resized or without its actual size defined in the tag, which resulted in a partial rate for one site and inadequacy for the other three. This deficiency has led to problems related to portability and efficiency during the analysis, reducing the

[1] http://marketshare.hitslink.com/report.aspx?qprid=17

[2] http://www.e-bit.com.br

quality of user experience when accessing the site and causing the unnecessary loading of information and poor design of the page by the browser to a smartphone or notebook.

4.2 Maximum access time

In the guideline "Loading Rate" all websites examined had pages that exceeded the size limit to the speed selected. As a result, there were difficulties in accessing information through different devices, significantly jeopardizing desired efficiency.

4.3 Interface

In this category it was possible to observe some critical points regarding the efficiency and portability of analyzed websites. The guideline "Text" had been partially followed in the five analyzed websites as it was found that none of the websites provided a function that allowed the user to not access the images. This led to serious problems because the purchase button disappeared and was not easily found.

During the use of a smartphone, it became evident that the deficiency in this category was because guidelines were not followed, endangering, in many cases, the use of the website and causing a loss of quality. The guideline "Title" was fully met in all websites. The guideline "Reading" was met in part in two websites, jeopardizing the quality of interaction with websites when it was not possible to view images.

The guideline "Compactness" could be considered as met only in situations where you used the search as way to find the desired product. However, we found that when this feature was not used, the amount of levels needed to access the page with the desired information went beyond the limit with three pages. Moreover, the guidelines "Scroll", "Word" and "Dimension" were not met in any of the analyzed websites, making it difficult to visualize the information shown. The inadequacy of websites in relation to the "Scripts and flash" guideline caused a big problem when accessed via smartphone as this device does not support these technologies.

Thus, during tests with smartphones, the use with scripts that were not supported by the device prevented the access to links and thus denied access to various areas of the websites. The guideline "Map", rated total adequacy in only one of the analyzed websites. This made it difficult to search for information in the other websites. Finally, we observed that the guidelines "Frame" and "Search" were fully met on all websites. The guideline "Search" is used as a tool to aid navigation in all analyzed websites, which could not be otherwise when it comes to e-commerce.

V. CONCLUSION

It is undeniable that the search for websites that satisfactorily meet the needs of users is a requirement of the market and becomes a factor influencing customer loyalty and consequently the visibility of organizations. This paper therefore presented a set of guidelines that can help with the construction of websites that allow indiscriminate access to information regardless of resources.

Towards this end, the priority is for quality characteristics, portability and efficiency. In e-commerce websites, availability and easy access for users is an important element for the success of the enterprise. As a result of the validation performed on five Brazilian e-commerce websites, we observed that the ones that only partially met the guidelines established in this work, or simply did not follow them, had their access to information jeopardized by low efficiency and lack of portability. Therefore, we conclude that more attention is needed to the design quality of a website as a way of ensuring greater availability and ease of access, regardless of the devices and network used.

REFERENCES

[1] ISO/IEC 9126. (2001) Software engineering — Product quality, 2001.

[2] DiLucca, G.A., Fasolino, A.R., Tramontata, P., Visagio, C.A. (2004) . Towards the definition of a maintainability model for web applications. 8º European Conference on Software Maintenance and Re-engineering. Tempere, 2004.

[3] Calero, C., Ruiz, J., Piattini, M. (2005). Classifying web metrics using the web quality model. Online Information Review. vol. 29, Nº 3, pp. 227-248. 2005.

[4] Welling, R., White, L. (2006). Website Performance measurement: promise and reality. Managing Service Quality. vol. 6, pp. 654-670. 2006.

[5] Lee, Y., Yeom, G. (2007). A Quality Chain Methodology for Ternary Web Services Quality View. 5th ACIS International Conference on Software Engineering Research, Management & Applications (SERA 2007), pp. 91-97. 2007.

[6] Behkamal, B., Kahani, M., Akbari, M. K. (2009). Customizing ISO/IEC 9126 quality model for evaluation of B2B applications. Information and Software Technology. vol. 51. pp. 599-609. 2009.

[7] Choudhury, M. M., Choudhury, M. A. Identification of the caracteristics of e-commerce websites. Webology, 7(1), Article 77. Available at: http://www.webology.org/2010/v7n1/a77.html 2010.

[8] Nielsen, J. (2008) Velocity How Little Do Users Read? http://www.useit.com/alertbox/percent-text-read.html . 2008.

[9] Nielsen, J. (2009a) Mobile Web 2009 = Desktop Web 1998 http://www.useit.com/alertbox/mobile-usability.html . 2009.

[10] Nielsen, J. (2009b). Mobile Usability. http://www.useit.com/alertbox/mobile-usability.html. 2009.

[11] Nielsen, J. (2009c) Powers of 10: Time Scales in User Experience . Disponível em: http://www.useit.com/alertbox/timeframes.html. 2009.

[12] Bertini, E., Santucci, G. (2004). Modelling Internet based applications for designing mult-devices adaptative interfaces. Proceedings of the Working Conference on Advanced Visual Interfaces AVI'04, Gallipoli, Italy. pp. 252 – 256. 2004.

[13] Lencevicius, R., Metz, E. (2006). Performance Assertions for Mobile Devices. Proceedings of the 2006 International Symposium on Software Testing and Analysis ISSTA'06. Portland, EUA. pp. 225 – 232. 2006.

[14] Ahmadi, H., Kong, J. (2008). Efficient Web Browsing on Small Screens. Proceedings of the Working Conference on Advanced Visual Interfaces, AVI'08. Napoli, Italy. pp. 23-30. 2008.

[15] Ravi, J.; Yu, Z.; Shi, W. (2009). A survey on dynamic Web content generation and delivery techniques. Journal of Network Computer Applications. pp. 943-960 2009.

[16] CETIC pesquisa -- TIC DOMICÍLIOS e USUÁRIOS 2009 - ÁREA URBANA. http://www.cetic.br/usuarios/tic/2009/index.htm. 2009. (in portuguese)

[17] Nielsen, J. (2009d) Velocity of Media Consumption: TV vs. the Web. http://www.useit.com/alertbox/media-velocity.html. 2009.

[18] W3C Web Content Accessibility Guidelines 2.0 http://www.w3.org/TR/WCAG20/ 2008.

[19] Weinreich, H.; Obendorf, H.; Herder, E.; Mayer, M. (2008). Not Quite the Average: An Empirical Study of Web Use . *ACM Transactions on the Web*, vol. 2, Nº. 5. 2008.

[20] Ertel, K. (2011) Explosion in Mobile Retail Provides Opportunity for Retailers. For See Results. 2011 .

Automatic Deployment and Monitoring of Software Processes: A Model-Driven Approach

Marília Aranha Freire, Fellipe Araújo Aleixo
Federal Institute of Education, Science and Technology of Rio Grande do Norte (IFRN)
Natal, Brazil
{marilia.freire, fellipe.aleixo}@ifrn.edu.br

Uirá Kulesza, Eduardo Aranha, Roberta Coelho
Federal University of Rio Grande do Norte (UFRN),
Natal, Brazil
{uira, eduardo, roberta}@dimap.ufrn.br

Abstract— **This paper presents a model-driven approach that aims at monitoring the software development processes, through the adoption of workflow management systems in order to allow the automatic collection of quantitative measures during software development project execution. Our approach provides support to the definition, deployment, execution and monitoring of software processes. It has been designed and implemented using existing model-driven technologies, which promote the transformation of process descriptions, specified using the Eclipse Process Framework (EPF), to workflow specifications that can be installed in the jBPM workflow engine.**

Keywords: Software Processes, Model-Driven Development, Process Metrics

I. INTRODUCTION

The increasing scale and complexity of software systems requires the definition and application of systematic methods, techniques and tools along the software life cycle. Large and medium software projects demand the definition and continuous improvement of software processes in order to promote the productive development of high-quality software. Over the last decades, the software engineering community has proposed many methods and techniques for the definition and improvement of software processes, such as: process models [1], process productivity metrics [6], maturity models, and process methodologies. All of them contribute to provide a more systematics process definition, execution, measurement and improvement. As a consequence, they contribute to the quality and productivity of the software development process.

Recent research work has proposed notations and formalisms to the process definition [1, 2]. Rational Method Composer (RMC) [1] consists of a suite of commercial tools for editing, modeling, viewing, configuration and publishing specifications of software processes. The Eclipse Process Framework (EPF) [2] is an open-source framework used for process definition based on the Unified Modeling Architecture (UMA) metamodel. In addition, there are many recent studies that investigate the integration of approaches and languages for process modeling and execution [3] [4] [5].

Despite the recent advances in the software process research area, little effort has been made to integrate techniques and tools that allow the definition, execution and monitoring of software processes. The integration of such techniques and tools can support the automated monitoring of software processes, thus contributing to the process activity estimation, quality control, productivity assessment, and project management. Moreover, current research work has not explored: (i) the adoption of existing and modern workflow systems to execute software processes specified in workflow languages; and (ii) the modeling, specification and integration of process metrics with software processes, and their respective deployment in workflow systems.

In this context, this paper presents a model-driven approach to the definition, execution and monitoring of software processes. The approach allows: (i) the modeling and specification of software processes with their respective productivity metrics; (ii) the automatic transformation of software process specifications to workflow specifications; (iii) the deployment and execution of these workflow specifications that represent software processes in workflow engines; and (iv) the monitoring of the process execution through a set of specific metrics automatically collected.

Our approach has been implemented using existing model-driven technologies. Software processes are specified using the Eclipse Process Framework (EPF) and automatically transformed to a specification written in the jPDL workflow language, which can then be deployed and executed in the jBPM workflow engine. QVTO and Accelleo languages are used to support the model-to-model and model-to-text transformations, respectively.

In addition to the support of automatic mapping of process elements to workflow abstractions, our approach still addresses: (i) the automatic weaving of metric collection actions inside process model elements that are subsequently refined to workflow actions or events - which will be responsible to gather information used to quantify each process productivity metric; and (ii) the process specification is also refined to customize Java Server Faces (JSF) web pages, which are used during the workflow execution to collect valuable information about the current status of the software process.

The remainder of this paper is organized as follows. Section II presents our model-driven approach for the definition, execution and monitoring of software processes. Section III details its current implementation and illustrates its application

to a software process instantiated from the OpenUP framework. Section IV presents related works. Finally, Section V presents the conclusions and points out future work directions.

II. AUTOMATIC EXECUTION AND MONITORING OF SOFTWARE DEVELOPMENT PROCESSES

This section gives an overview of our model-driven approach that provides support to the definition, deployment, execution and monitoring of software processes. Next, we describe the main systematic activities that define our approach.

1. Process Modeling and Specification. The first activity of our approach is the modeling and specification of a software process. The EPF framework and associated tools are used to support the process definition. Existing process frameworks, such as OpenUP (available in the EPF repository), can be reused and customized in this activity. The main responsible for this activity is the process engineer, who produces EPF process specifications (output artifact of this activity).

2. Metric Integration with Software Processes. After the modeling and specification of the software process, our approach allows process engineers to select and specify the metrics that will be automatically monitored and collected during the process execution. A pre-defined set of process metrics is already available to be selected and composed with the software process. The metrics integration involves the definition of the start and the end activities of the monitored software process. In order to support the metrics inclusion, we defined a new metamodel which helps the definition and integration of the metrics with EPF specifications. The output of the metrics inclusion is the EPF process specification refined with activities related to the automatic collection of the metrics.

3. Process to Workflow Transformation. In this activity, the refined EPF process specification is automatically transformed to a workflow description. To do so, we apply model-to-model and model-to-text transformations to automatically map abstractions from the process domain to the workflow domain, thus enabling the instantiation and deployment of the software process in a workflow engine. In the current implementation of our approach, the EPF process is transformed to a workflow specification described in the jPDL language, which is used by the jBPM workflow system.

4. Workflow Deployment and Execution. The final activity of our approach aims at deploying and executing the workflow specification which represents a software process in the target workflow engine. It allows monitoring the software process activities, including the quantification of specific process metrics related to the team and project productivity. Team members of the software project under development must interact with the workflow system in order to maintain updated the progress of the process activities.

III. OUR APPROACH IN ACTION

In this section, we detail the implementation our approach for process definition, customization and execution, using the EPF framework and the jBPM workflow engine. In addition, we illustrate the application of our approach to the OpenUP software process through three different stages. Next subsections describe how each activity of the proposed approach was applied in order to define a software process based on the OpenUP framework.

A. Process Modeling and Specification

The first activity of our approach is the process modeling and specification. The Eclipse Process Framework (EPF) and associated tools are used to support this activity. Existing process frameworks, such as OpenUP, can be used as a base to define the elements (activities, roles, artifacts, iterations) of the development process to be modeled. Fig. 2 (Step 1) shows a fragment of a process derived from the OpenUP. It models an iteration of OpenUp Elaboration phase, with the following activities highlighted: *Identify and Refine Requirements*, *Develop Solution Increment* and *Test Solution*. Such activities will be subject to be monitored during the execution of the software processes.

The processes from the EPF framework are specified according to the Unified Metamodel Architecture (UMA), which is a subset of the Software Process Engineering Metamodel (SPEM) defined by OMG. The UMA is specified as an Ecore metamodel using the Eclipse Modeling Framework (EMF). This brings facilities to the subsequent activities of our approach, which requires the EPF model transformation with different purposes. In addition, there are many existing model-driven platforms and technologies that were developed based on the EMF. We have also extended our approach to support the specification and integration of metrics for the process defined. After the process specification is necessary to define the metrics that have to be intercalated with the process tasks. Next subsection describes how this extension is implemented.

B. Metrics Integration in Software Processes

The metrics integration activity starts when the process engineer selects and specifies the metrics from a pre-defined set of process metrics, already available. In our current approach implementation, we have initially focused in metrics that quantify the duration of significant project tasks, such as process tasks, activities, steps and iterations.

During the modeling of an OpenUP based process, we have selected 2 project metrics: "*Duration of UC requirements tasks*" and "*Duration of UC development tasks*". The first one is defined to intercept the process activities to count the time interval required to accomplish activities related to the specification of use cases. The last one is responsible to intercept the process activities to count the time interval related to activities for developing use cases (UC).

In order to promote the metric modeling and integration with an EPF process model, it was necessary to define a separate metric model. Fig. 1 shows the metamodel for composing metrics in software processes. It allows specifying where the metric will be quantified in the workflow that represents the process execution. It supports process activities and tasks like intervention points for the metrics collection. Therefore, it is necessary to specify for each metric which activities or tasks it must operate through a simplified metric model. The metric metamodel defines several abstractions related to a metric plan for a development process. The

MetricPlanModel element aggregates the set of metrics defined for the process (Fig. 1).

The central model element is the *Metric*. It is used to describe all metrics defined for the process. Each metric has a unique identifier (*id* attribute), and a *name* and *description* that provide information about the metric. The *type* parameter of the *Metric* abstraction contains the metric type that can be one of the following: *hardData*, *softData* and *normalizedData*. It is specified as an enumeration named *MetricType*. This classification identifies the kind of information used in measurement [6]. The hard data information can be quantified with little or no subjectivity. The soft data information refers to the kind of information in which human opinions must be evaluated or, in other words, information where precision is not possible. Finally, the normalized data is used to more fairly evaluate projects that have different characteristics. They represent standard metrics for comparative purpose. Our approach focuses currently only on the hard data measurements.

The *form* property represents the type of event (task) quantified by the metric. The *continuous ColectType* means that the task cannot be paused and the metric value is computed as the interval from start until the end of the task or activity under measurement. The *intercalated ColectType* means that the task can be paused and the metric value is the sum of each timeslice spent on it. Finally, the *unit* property is used for indicating the unit of the metric. For the metrics modeled for our case study, we have the following option for the *MetricUnit* enumeration: *minutes* or UC.

While modeling a metric plan, we have to choose a base process model element that is used for the metric monitoring. These model elements can be currently process activities (*ActivityMetric*) or tasks (*TaskMetric*). The *ActivityMetric* element is used to define that a metric must intercept processes elements of the activity type. It adds two attributes for the metric. The "*activityBegin*" and "*activityEnd*" attributes store, respectively, the begin and end activity names that this metric will intercept, thus representing a sequence of tasks to be monitored. If the metric is intended to intercept only one activity, the *activityEnd* attribute must be blank. If there is not a direct flow between the activities, only these two activities will be considered. Finally, the *TaskMetric* element adds only one new attribute, named *tasksBase*, that must contain the names of all tasks affected by the metric being defined.

Fig. 2 illustrates a metric model for our OpenUp based process. It specifies two metrics for activities interceptions: *Duration of UC development tasks* and *Duration of UC requirements tasks*. The *Duration of UC development tasks* metric, for example, is defined to intercept and count the time spent the process activities related to the development of a use case (UC). The model exemplified in Fig. 2 (Step 2) sets activities from "develop_solution" activity to "test_solution" activity. It means that all activities between them have to be monitored. The value *continuous* setting to the form attribute implies that the time measured by this metric is the time interval between the start and the end of each activity specified.

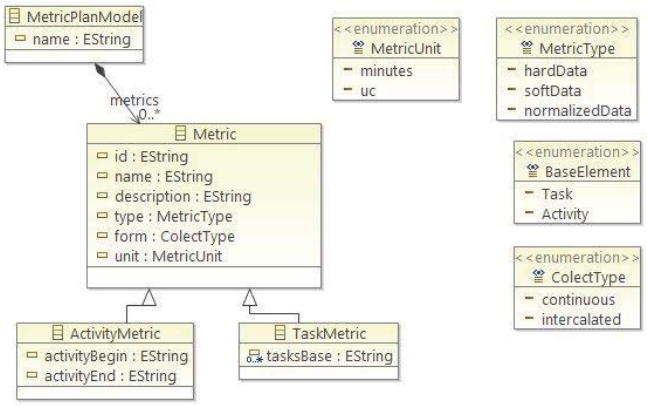


Figure 1 Metric Metamodel

C. *Process to Workflow Transformation*

After the modeling of a software process and the respective metrics that must be collected during its execution, our approach focuses on the model-to-model and model-to-text transformations of the process model to a workflow specification, which can be deployed in a workflow engine.

1) *Model to Model Transformation*

The M2M transformation is responsible for both: (i) realizing the process and metrics integration; and (ii) transforming the process model to a workflow specification. When applying this transformation, all process activities and tasks specified in the metric model must derivate new elements that will be responsible for the automatic collection of the productivity metrics. In addition, the transformation also maps all the process abstractions modeled using the UMA metamodel to workflow abstractions modeled using the jPDL language, which is adopted by the jBPM workflow system.

Table 1 shows the mapping from the process abstractions from the UMA metamodel to their respective jPDL workflow element produced by the M2M transformation. As you can see, there is a direct mapping between existing Activity elements in the UMA process models to specific node elements from the jPDL language. For example, the "Activity without predecessor" is mapped to the "start-node", and the "Activity with more and one successor" is mapped to the "fork-node".

TABLE I. MAPPINGS FOR JPDL ELEMENTS

UMA or Metric Element	JPDL Element
Activity	task-node
Worker-order	transition
Activity without predecessor	start-node
Activity without successor	end-node
Activities with more than one successor	fork-node
Activities with more than one predecessor	join-node
Task	task
Metric	event and action

On the other hand, each metric is associated with JPDL actions elements that are triggered by JPDL event elements at the end of each task execution. These actions elements are responsible for the data collection through their association with existing Java classes. Fig. 2 (Stage B), for example, shows the result of the process-to-workflow transformation. As we can see, there are two jPDL event elements responsible to quantify the use case productivity metric for the different.

The integration between the UMA process model and our metric model can be seen as a transversal composition. The integration is performed using the QVTO transformation language, which receives a UMA process model and a metric model as input, and it produces as output a jPDL process execution model (workflow model). Fig. 2 presents the metric and process input models in the Stage A, as well as the jPDL source code output in the Stage B. At the end of this stage, we have a workflow model that is produced as the weaving of the process and metrics models. Every measurement event can be seen as an aspect advice that acts on the task execution. We must consider that the jPDL elements that enable the measurements (actions and events) are not mapped visually to the workflow definition, but they are represented as elements that can be triggered and processed in background during the workflow execution. The task-node *Test Solution*, for example, has an event of "*task-end*" type which contains an action named *UCDevTime* that is associated with the *UCDevActionHandler* class responsible for the metric calculation.

2) Model to Text Transformation:

The model-to-text transformation is used in our approach to map and refine the resulting jPDL specification – produced as output from the model-to-model transformation – to workflow code assets that will be deployed in the jBPM engine to tracking and monitoring the software process execution. This transformation has been written in Acceleo template language. The jBPM workflow engine used in our approach enables the creation of web forms implemented in Java Server Faces (JSF) framework, from a jPDL workflow model definition. Such forms can be used to track the process workflow with the aim of storing information about the tasks and/or decisions taken during its execution. The transformation was created for the composition of the JSF dynamic web pages that will collect additional information for the tasks of our workflow. Specific jBPM taglibs are used in these JSF pages to indicate to the workflow engine to run the form and associate it with the process.

The model-to-text transformation is also responsible for generating a configuration file (forms.xml) that associates each task of our workflow to a JSF form and Java classes that represent the custom actions responsible to quantify the specified measurements.

The following code assets are produced as result of the transformation: (i) process-definition.xml – the jPDL specification that represents the process workflow; (ii) forms.xml – a configuration file that associates the JSF pages to the jPDL model generated by M2M transformation; (iii) JSF web pages with forms for each process task; and (iv) Java classes associated with each existing metric that will be executed after the conclusion of specific workflow tasks as described in the jPDL workflow specification. Due to space restriction, these code assets are not presented in this paper, but they can be found at [7].

D. Workflow Deployment and Execution

All code assets generated from the process-to-workflow transformation are organized and stored in a jPDL project in Eclipse IDE. Through simple manual settings, the jPDL project and associated code assets can be easily deployed and run on the jBPM workflow engine.

Fig. 2 (Stage C) shows the process execution after the deployment of workflow code and configuration assets in the jBPM engine. The execution can be monitored in the engine console deployed as a web application in the JBoss application server. After the deployment of the process workflow, the user can run a new process instance. Step 4 of Fig. 2 shows a process instance summary where links are displayed for viewing information about the process execution, such as start date, status, tasks instances, process image, process variables, among other information.

Step 5 shows the *Test_Solution* task form during execution. When performing each task, the software engineer (or project manager) informs required information in the task execution interface (JSF form), and she/he can end the task after its completion by using the transition button. The process instance can be checked for tracking issues at any time. All information from the process workflow and process instances is stored in a database.

Step 7 presents the task form during its execution. It shows the *id*, *name*, *status*, *start date*, *end date* and *actions* regarding the tasks that belong to the execution of the process workflow instance. As you can see, the "*Identify and Refine Requirements"* and "*Develop Solution"* tasks are running. Step 6 shows the values of process variables after the end of their associated tasks. The form displays the value 1265 for the variable *UCDevTime*, and the value 608 for the variable *UCReqTime*. The values of these variables were quantified in *minutes* and they are associated with the metrics responsible for the measurement of time spent in the use cases requirements specification and development during an iteration of the OpenUP Elaboration phase. These metrics are collected for each process iteration.

IV. RELATED WORK

Recent studies have promoted the integration of approaches and languages for modeling the process execution. However, these research works mainly emphasize the modeling of processes using new proposed modeling languages. None of them focuses monitoring software processes with automatic metric gathering in workflow systems.

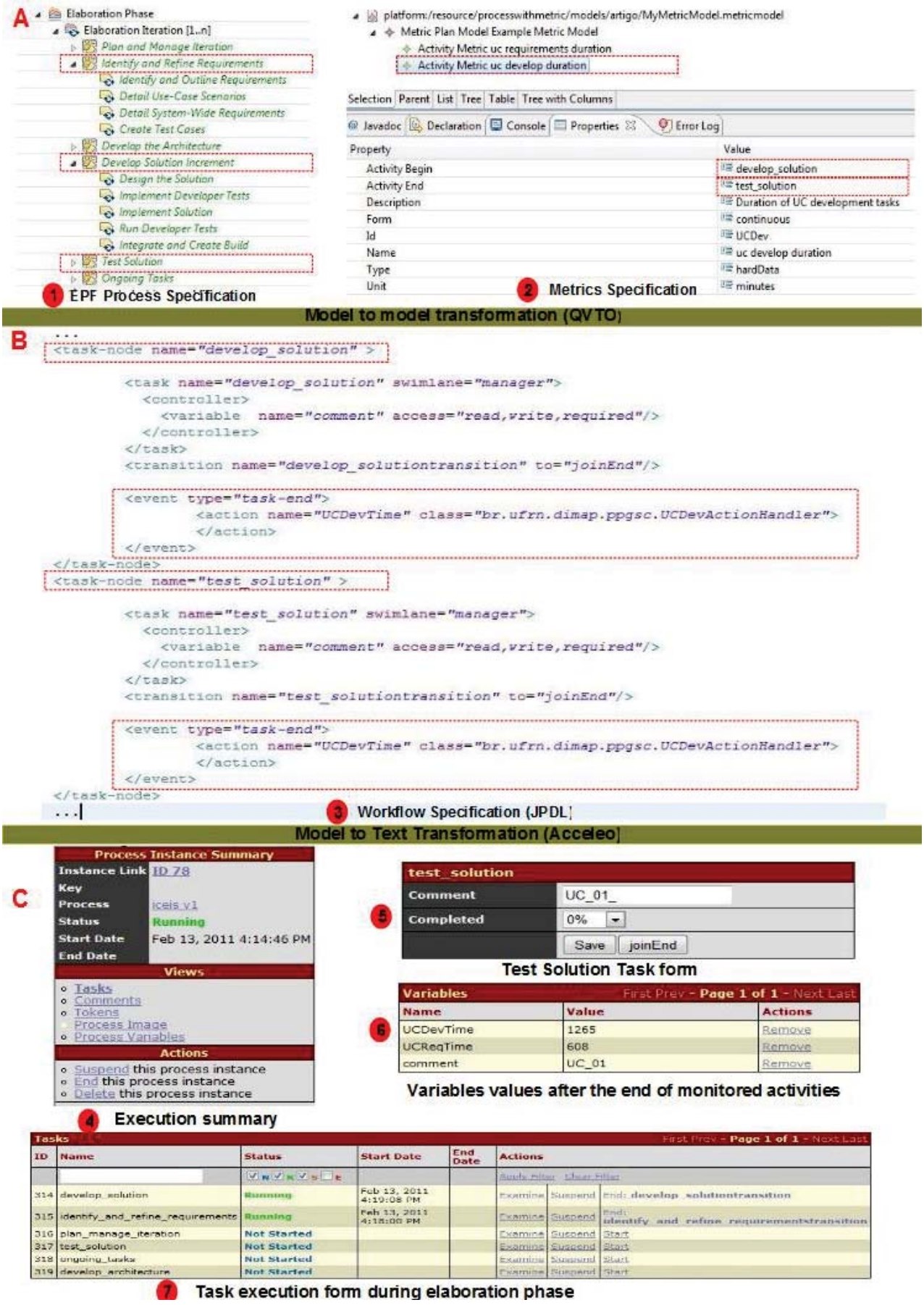


Figure 2 Approach in Action

Bendraou et al [3] presents a model-driven approach that includes the mapping between UML4SPM and WS-BPEL languages. Each language operates in a different domain: software process definition and process execution, respectively. The rationale for this choice was that both languages have their strengths in both the modeling and execution purposes. The transformation from process to workflow models has been defined by a program written directly in the Java language.

Maciel et al [5] also defines a model-based approach to software process modeling using MDA. The authors mention that the approach is totally based on OMG standards, such as SPEM 2.0, UML 2.0, MDA. However, the main focus of this work is on the specification of processes elements. The approach does not propose a complete transformation of software processes specification in order to enable their execution in workflow systems.

The approach proposed in this paper has in common with the works mentioned above the aim of addressing the process execution from existing process specification. However, our work differs from the most research carried out because: (i) our approach proposes automated model transformations between technologies of process definition (EPF) and execution (jBPM) widely used in industry and academia; (ii) it promotes the generation of workflow code assets (configuration files, JSF components and Java classes) that provides support to the effective processes execution and monitoring in workflow systems; and, last but not least, (iii) it enables the automatic collection of metrics during the execution of the process while minimizing the cost and cultural issues traditionally involved in manual collection of measures in software projects.

V. CONCLUSIONS

This paper presented an approach to the definition, deployment, execution and monitoring of software development processes that supports the automatic collection of quantitative measures during project execution. The approach was implemented and evaluated using existing model-driven technologies and platforms. Our implementation enables the transformation of process specifications from the Eclipse Process Framework (EPF) to jBPM workflow specifications.

Our approach also proposes extensions to the process modeling in order to support the collection of productivity metrics during the process workflow execution. Each new metric to be adopted in a software project must be specified in a specific model. When modeling each process metric, it is necessary to specify its points of intersection in the process. These join points can be the beginning and end of tasks/activities execution, or even the tasks/activities creation and deletion. The task and activity model elements from the process model act as join points, which can be intercepted to collect information of interest to be stored for the metrics. As described in the paper, the model-to-model transformation from the process and metric models to a jPDL workflow specification can be seen as a weaver that produces as output a process execution description with a set of actions responsible to automatically collect information during the process execution.

As future and ongoing work, several refinements of the approach can be done and some of them are already under development. Currently, an industrial case study in applying new metrics to measure productivity of a project is being modeled and implemented. New measures are being explored to extend our approach, such as the effort spent by team members on project tasks or activities, and the number of bugs or defects found, reported and corrected. In addition, we are integrating the approach proposed in this paper with variability management techniques and mechanisms previously proposed [8]. The main goal is to provide support to the variability modeling of process lines [9] and the automatic derivation, deployment and monitoring of software processes in workflow systems.

REFERENCES

[1] IBM. (2010) Rational Method Composer. [Online] . http://www-01.ibm.com/software/awdtools/rmc

[2] Eclipse Foundation. (2010) Eclipse Process Framework (EPF) Composer 1.0 Architecture Overview. [Online]. http://www.eclipse.org/epf/composer_architecture/

[3] Reda Bendraou, Andrey Sadovykh, and Marie-Pierre Gerva, "Software Process Modeling and Execution: The UML4SPM to WS-BPEL Approach," in *33rd EUROMICRO Conference SEAA*., 2007.

[4] Reda Bendraou, Jean-Marc Jézéquel, and Franck Fleurey, "Combining Aspect and Model-Driven Engineering Approaches for Software Process Modeling and Execution," in *International Software Process Conference (ICSP 2009)*, Vancouver, Canada, 2009, pp. 148-160.

[5] Rita Suzana Pitangueira Maciel, Bruno Carreiro da Silva, Ana Patrícia Fontes Magalhães, and Nelson Souto Rosa, "An Integrated Approach for Model Driven Process Modeling and Enactment," in *XXIII Simpósio Brasileiro de Engenharia de Software*, 2009.

[6] Capers Jones, *Applied Software Measurement* , 3rd ed.: McGraw-Hill , 2008.

[7] Marília Freire, Fellipe Aleixo, Uirá Kulezsa, Roberta Coelho, and Eduardo Aranha. Software Process Monitoring. [Online]. https://sites.google.com/site/softwareprocessmonitoring/

[8] Fellipe Aleixo, Marília Freire, Wanderson Câmara, and Uirá Kulezsa, "Automating the Variability Management, Customization and Deployment of Software Processes: A Model-Driven Approach," in *Enterprise Information Systems*, Joaquim Filipe and José Cordeiro, Eds.: Springer Berlin Heidelberg, 2011, pp. 372-387.

[9] Ove Armbrust et al., "Scoping software process lines," *Software Process: Improvement and Practice 14*, vol. 3, pp. 181-197, 2009.

FBDtoVerilog: A Vendor-Independent Translation from FBDs into Verilog Programs

Junbeom Yoo , Jong-Hoon Lee
Div. of Computer Science and Engineering
Konkuk University
Seoul, Republic of Korea
{jbyoo , kirdess}@konkuk.ac.kr

Sehun Jeong , Sungdeok Cha
Dept. of Computer Science and Engineering
Korea University
Seoul, Republic of Korea
{gifaranga , scha}@korea.ac.kr

***Abstract*—FBD (Function Block Diagram) is one of the widely used PLC (Programmable Logic Controller) programming languages in plant automation industry. Many vendors and products have their own forms and formats, which are not compatible with others. Formal verification techniques and tools for FBDs should have provided vendor- and product-specific versions. PLCopen, a vendor/product independent worldwide association, provides a standardized way to define FBDs in an XML format. This paper proposes a CASE tool, FBDtoVerilog, which translates the PLCopen-FBDs into Verilog programs. Verilog is an input programming language for formal verification tools such as VIS (Verification with Interaction and Synthesis). It had been efficiently used as an input front-end of formal verifications, when developing software controllers of nuclear power plants in Korea. We demonstrate its usefulness and effectiveness with a prototype version of FBDs which had developed for APR-1400 nuclear power reactor in Korea.**

Keywords-Translation; PLCopen; FBD; Verilog; CASE

I. INTRODUCTION

FBD is one of the five widely used PLC programming languages defined by International Electrotechnical Commission (IEC) [1]. It visually expresses PLC controller's behavior as sequentially interconnected function blocks. The KINCS project [2] developed a new RPS (Reactor Protection System) for Korean nuclear power plants and implemented its software in FBDs. Rigorous quality demonstration of RPS software was also required by the regulation agency (e.g., KINS [3] in Korea) prior to issuing operational approval. Automated and formal verification techniques such as model checking [4, 5] and equivalence checking [6] was applied to the FBDs in order to ensure adequate quality assurance.

Formal verification techniques have their own input front-ends. For example, the VIS verification system [7] needs Verilog program, while the SMV [8] model checker does SMV input program or Verilog program. Translation from FBDs into these front-ends is therefore the first step to applying various formal verification techniques into FBD programs. Our former researches on FBD verifications, '*FBD Verifier'* and '*PLC Verifier'* [9, 10] had to use a FBD format specific to POSCO ICT [11], which generated from its PLC engineering tool '*pSET*' [12]. Some changes in the format, however, made us difficult to keep consistency and correctness of the automatic translators and verification tools. This paper proposes a CASE tool, '*FBDtoVerilog*' translating FBDs into Verilog programs, but uses a de facto standard XML format of FBD, proposed by PLCopen [13]. PLCopen is a vendor- and product-independent worldwide association. *FBDtoVerilog* can translate into Verilog programs FBDs from any vendors complying with the association's standard.

We demonstrated correctness and effectiveness of the proposed translator through a case study, formal verification of FBD programs using the SMV and the VIS. We used a prototype version [14] of FBD programs which had developed for a nuclear reactor protection system in Korea. The remainder of the paper is as follows. Section 2 introduces the FBD and PLC open association briefly. It also introduces relevant features of Verilog programming language, which are pertinent to our discussion. Section 3 introduces the CASE tool *FBDtoVerilog*. Section 4 explains a case study of formal verification using the proposed tool. Section 5 concludes the paper.

II. BACKGROUND

A. Function Block Diagram

An FBD (Function Block Diagram) consists of an arbitrary number of function blocks, '*wired'* together in a manner similar to a circuit diagram. The international standard IEC 61131-3 defined 10 categories and all function blocks. For example, the function block ADD performs arithmetic addition of n+1 IN values and stores the result in OUT variable. Others are interpreted in a similar way.

Fig.1 shows a part of preliminary FBD programs for the KNICS RPS BP (Bistable Processor) logic. The former was generated mechanically [15] from a formal requirements specification [14], while the latter was developed by domain experts. Even though they look different in appearance, they show the same behavior. We used these FBDs as examples to keep consistent with our former work and aid understanding of FBD programs. These FBDs both creates a warning signal '*th_X_Pretrip*' when the pre-trip condition (i.e., reactor shutdown) remains true for *k_Trip_Delay* time units as implemented in the TOF function block. The number in parenthesis above each function block denotes its execution

order. The output '*th_Prev_X_Pretrip'* from MOVE stores current value of '*th_X_Pretrip'* in order to use in the next execution cycle. A large number of FBDs similar to Fig.1 and Fig.2 are assembled hierarchically and executed according to a predefined sequential execution order.

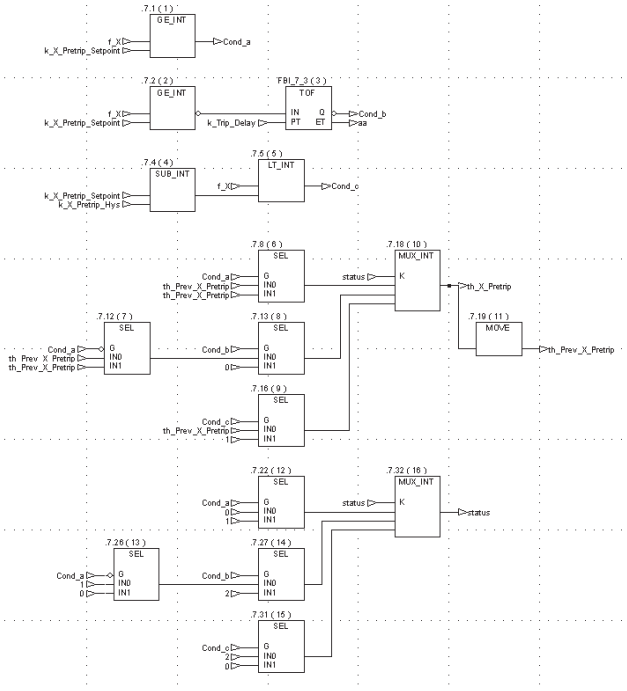


Figure 1. An FBD for *th_X_Pretrip* logic, generated mechanically

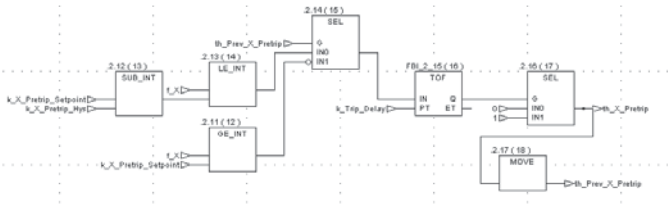


Figure 2. An FBD for *th_X_Pretrip* logic, developed by domain experts

B. PLC open

PLCopen [13] is a vendor- and product-independent worldwide association, aiming to resolve topics related to control programming and to support the use of international standards IEC 61131-3 [1]. A working group named TC6 for XML (eXtended Markup Language) in PLCopen has defined an open interface between all different kinds of software tools, which provides the ability to transfer one's information to other platforms. This paper used the XML specification defining FBD programming languages. The format unfortunately does not include all items which we need to translate FBDs into Verilog programs, so we used a few items in the specification for our specific purpose. The details will be introduced in Section 4.

C. Verilog Programming

Verilog is one of the most common Hardware Description Languages (HDLs) used by Integrated Circuit (IC) designers. Many verification and analysis techniques and tools widely use Verilog as an input programming language.

Fig.3 shows a Verilog program translated from the FBD described in Fig.2 according to the translation rules [15]. There are two inputs and two outputs. As input prefixes "*k_*" indicate constants variables. *th_Prev_X_Pretrip* is used as both input and output. Since it stores the value of *th_X_Pretrip* using the MOVE function block, we defined it as a *reg* variable in lines (8) and (32). The FBD's output is produced in the *assign* statements (12) ~ (18) by composing several function blocks in the FBD. It also uses the variable *timer* to emulate the TOF function block, which we emulate with *procedural* assignments using *always* statements (19) ~ (31). We restricted the number of TOF internal states to six in this example as defined in (1). In addition, we used the *clk* variable, reserved for simulation purposes in the VIS verification system, to simulate cyclic executions of PLCs.

```
(1)  typedef enum {T0, T1, T2, T3, T4, T5} timer_state;
(2)  `define k_X_Pretrip_Setpoint 30;
(3)  `define k_X_Pretrip_Hys 30;

(4)  module th_X_Pretrip(clk, f_X, th_X_Pretrip);
(5)      input clk;
(6)      input [0:6] f_X;
(7)      output th_X_Pretrip;

(8)      reg th_prev_X_Pretrip;
(9)      timer_state reg timer;

(10)     initial th_prev_X_Pretrip = 1;
(11)     initial timer = T0;

(12)     assign th_X_Pretrip =
(13)             ((th_prev_X_Pretrip == 0) && (f_X <= (k_X_Pretrip_Setpoint - k_X_Pretrip_Hys))) ? 1 :
(14)             ((th_prev_X_Pretrip == 0) && (f_X > (k_X_Pretrip_Setpoint - k_X_Pretrip_Hys)) && (timer == T5)) ? 0 :
(15)             ((th_prev_X_Pretrip == 0) && (f_X > (k_X_Pretrip_Setpoint - k_X_Pretrip_Hys)) && (timer != T5)) ? 1 :
(16)             ((th_prev_X_Pretrip == 1) && (f_X < k_X_Pretrip_Setpoint)) ? 1 :
(17)             ((th_prev_X_Pretrip == 1) && (f_X >= k_X_Pretrip_Setpoint) && (timer == T5)) ? 0 :
(18)             ((th_prev_X_Pretrip == 1) && (f_X >= k_X_Pretrip_Setpoint) && (timer != T5)) ? 1 : 0;

(19)     always @(posedge clk) begin
(20)             if(f_X >= k_X_Pretrip_Setpoint) begin
(21)                     case(timer)
(22)                             T0: timer = T1;
(23)                             T1: timer = T2;
(24)                             T2: timer = T3;
(25)                             T3: timer = T4;
(26)                             T4: timer = T5;
(27)                             T5: timer = T5;
(28)                     endcase
(29)             end
(30)             else
(31)                     timer = T0;

(32)             th_prev_X_Pretrip = th_X_Pretrip;
(33)     end

(34)  endmodule
```

Figure 3. A Verilog program translated from the FBD in Fig.3

III. FBDTOVERILOG

We have used the proposed, but not fully refined, FBD definition and translation rules [16] to formally verify FBD programs in the KNICS project. Fig.4 briefly shows how we have used them to verify the FBD programs with various verification techniques and tools. It is a part of PLC-based software development framework we proposed in [15]. We planned to apply two formal verification techniques into the FBDs, the model checking and the equivalence checking. While the former can prove mathematically whether the FBD satisfies important properties, the latter can conclude whether two different FBDs show the same behavior or not.

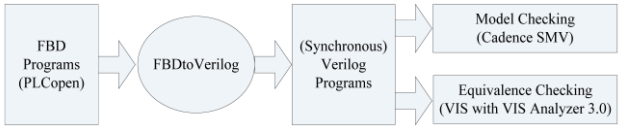


Figure 4. The use of the proposed translator in formal verifications

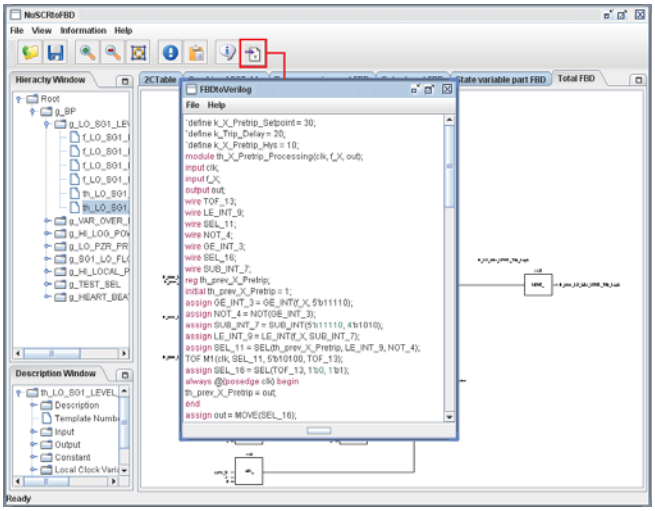


Figure 5. FBDtoVerilog v1.0 Screen-dump

We had developed automatic translator and verification assisting tool *FBD Verifier* [9], and applied them into the KNICS project in part [17]. However, our former work started with a specific version of FBDs specialized for POSCON ICT. In order to apply useful formal verification techniques with no hindrance from the compatibility problem, we decided to separate the translator from the specific FBD and used standard XML format of FBD. Fig.5 depicts a screen-dump of *FBDtoVerilog 1.0* CASE tool which we have developed. It is embedded in *NuSCRtoFBD 3.0* and reads standard FBDs of PLCopen and produces (synchronous) Verilog programs.

FBDtoVerilog used an *addData*, general-purpose element of the PLCopen XML specification [13]. *NuSCRtoFBD 3.0* generates PLCopen specific XML that every single function block element belongs to an externally visible output which *addData* element stores its name. Fig.6 shows *LE_INT* block cooperate with computing output *th_X_Pretrip*. *FBDtoVerilog* uses the information to translate an FBD's flat structure into a Verilog module's hierarchy structure.

```
<block instanceName="LE_INT" localId="9" typeName="LE_INT" exectionOrderId="15">
        <inputVariables>
                ...
        </inputVariables>
        <outputVariables>
                ...
        </outputVariables>
        <addData>
                <data name="th_X_Pretrip" handleUnknown="discard">
                        <BOOL/>
                </data>
        </addData>
</block>
```

Figure 6. Usage of *addData* element in *th_X_Pretrip* FBD specification

The current version of *FBDtoVerilog 1.0* has some room to improve. First, it produces incomplete Verilog code that requires manual post-process to supply variable size in bit vectors. Performing formal verification activities such as equivalence checking and model checking require complete size determination. Second, it translates every function block, even though they are too simple to be defined as a Verilog function. We suggest practically possible translation option in Table 1. These aspects will implement in next version of *FBDtoVerilog*.

Table 1. Alternative optimized function block translation rule

	Current rule	Optimized rule

SEL	var = SEL(a, b, c);
	…
	function SEL;
	  input in1;
	  input in2;
	  input in3;
	  begin
	  SEL = (in1 == 1) ? in3 :
	  in2;
	  end
	endfunction
		var = (a == 1) ? b : c;

ADD	var = ADD(a, b);
	…
	function [0:6] SUB_INT;
	  input [0:6] in1;
	  input [0:6] in2;
	  begin
	  SUB_INT = (in1 - in2);
	  end
	endfunction
		var = a + b;

IV. CASE STUDY

We performed a case study as described in Fig.7 to validate correctness of *FBDtoVerilog 1.0*. We translated the system FBD *g_LO_SG1_Level* depicted abstractly in Fig.1 and Fig.2 into Verilog programs. And we applied manual post-processing on the translated code with preserving its original semantic as we mentioned in Section 3 (see Fig.8). We had plan performing Cadence SMV model checking and the VIS equivalence checking against the Verilog program. When preparing the case study, we only focused on checking the validity of the CASE tool.

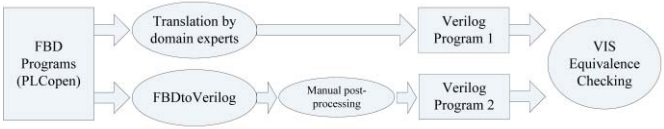


Figure 7. Case study plan

The VIS equivalence checking result shows "sequentially equivalent" message as we can see in Fig.9, which means two Verilog programs have same output behavior against same inputs. We also conducted flawless examination of two source codes to validate our tool's correctness, since source codes have quite different coding style. For example, original domain expert generated code doesn't contain user-define functions that our code has.

Cadence SMV model checker cannot read the Verilog program which the current version of *FBDtoVerilog* produced. We found out that the model checker forbid the reuse of functions such as *SEL* or *ADD* in our code. We are working on this issue with more refined translation rules. From the results, we can say that our proto-type *FBDtoVerilog* archived its main purpose at minimum that the translated Verilog code has same behavior with the original code developed and certified by domain experts.

```
module main(clk, f_X, th_X_Pretrip);
input clk;
input [0:6] f_X;
output th_X_Pretrip;

//wires and regs
wire tof_out;
wire Sel1;
wire Sel2;
…

reg th_prev_X_Pretrip;
initial th_prev_X_Pretrip = 1;

// assign temp wire varialbes
assign Ge_int = GE_INT(f_X, 7'b0011110);
assign Sel1 = SEL(th_prev_X_Pretrip, Le_int, Not);
…
TOF M1(clk, Sel1, 7'b0000101, tof_out);
assign th_X_Pretrip = MOVE(Sel2);

always @(posedge clk) begin
th_prev_X_Pretrip = th_X_Pretrip;
end

//function blocks
function GE_INT;
input [0:6] in1;
…
```

Figure 8. Translated Verilog code from the FBD in Fig.3

```
vis release 2.3 (compiled 2011 년 3 월 14 일 월요일 22 시 50 분 06 초 KST)
vis> read_blif_mv ./th_X_Pretrip_tr_v4.mv
Warning: Some variables are unused in model NOT.
Warning: Some variables are unused in model GE_INT.
Warning: Some variables are unused in model SEL.
Warning: Some variables are unused in model MOVE.
Warning: Some variables are unused in model LE_INT.
vis> init_verify
vis> seq_verify ./th_X_Pretrip_custom_v3.mv
Networks are sequentially equivalent.
```

Figure 9. VIS equivalence checking result

Our future work will focus on implementing next version of *FBDtoVerilog*. First issue is fully automatic Verilog code generation feature that includes variable size determination algorithm. Second issue is Cadence SMV compatible code generation feature. And we will plan the case study that verifies further correctness of the *FBDtoVerilog* through VIS equivalence checking and Cadence SMV model checker using all FBDs used in the KNICS project.

V. CONCLUSION

As safety critical systems are using FBD as standard representation of software design, software verification on FBDs becomes indispensable. Our former researches on FBD verifications used a vendor-specific format of FBD, and it made us difficult to keep consistency and correctness of the automatic translator and verification tools. This paper proposes a CASE tool, '*FBDtoVerilog*' translating FBDs into Verilog programs, but uses a de facto standard XML format of FBD, proposed by PLCopen. We demonstrated correctness and effectiveness of the assisting tool through a case study, formal verification of FBD programs using the VIS. We used a prototype version of FBD programs developed for a nuclear reactor protection system in Korea. The case study demonstrated that the CASE tool, *FBDtoVerilog* translates standard FBDs into Verilog programs correctly and efficiently.

ACKNOWLEDGMENT

This research was partially supported by the MKE (The Ministry of Knowledge Economy), Korea, under the ITRC (Information Technology Research Center) support program supervised by the NIPA (National IT Industry Promotion Agency (NIPA-2011-(C1090-1131-0008) and NIPA-2010-(C1090-1031-0003)). This research was also supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (2010-0002566).

REFERENCES

[1] IEC (International Electrotechnical Commission), International standard for programmable controllers: Programming languages: Part 3 (IEC 61131-3), 1993.

[2] KNICS (Korea Nuclear Instrumentation & Control System R&D Center), http://www.knics.re.kr/english/eindex.html.

[3] KINS (Korea Institute of Nuclear Safety), http://www.kins.re.kr.

[4] E. M. Clarke, E. A. Emerson, and A. P. Sistla. "Automatic verification of finite-state concurrent systems using temporal logic specifications," ACM Trans. Programming Languages and Sysems, Vol. 8, No. 2, pp.244-263, 1986.

[5] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled, Model Checking, MIT Press, 1999.

[6] Shi-Yu Huang and Kwang-Ting(Tim) Cheng, Fromal Equivalence Checking and Debugging, chapter 4, Kliwer Academic Publishers, 1998.

[7] Robert K. Brayton, Gary D. Hachtel, Alberto Sangiovanni-Vincentelli, Fabio Somenzi, Adnan Aziz, Szu-Tsung Cheng, Stephen A. Edwards, Sunil P. Khatri, Yuji Kukimoto, Abelardo Pardo, Shaz Qadeer, Rajeev K. Ranjan, Shaker Sarwary, Thomas R. Shiple, Gitanjali Swamy, and Tiziano Villa, "VIS : A system for verification and synthesis," In the Eighth International Conference on Computer Aided Verification, CAV '96, pages 428-432, 1996.

[8] K. L. McMillan. Symbolic Model Checking. Kluwer Academic Publishers, 1993.

[9] Eunkyoung Jee, Seungjae Jeon, Sungdeok Cha, Kwanyong Koh, Junbeom Yoo, Geeyong Park, and Poonghyun Seong, "FBD Verifier: Interactive and Visual Analysis of Counterexample in Formal Verification of Function Block Diagram," Journal of Research and Practice in Information Technology, Vol.42, No.3, pp.255-272, August, 2010.

[10] Junbeom Yoo, Sungdeok Cha, and Eunkyoung Jee, "Verificatin of PLC Programs written in FBD with VIS, Nuclear Engineering and Technology, Vol.41, No.1, pp.79-90, 2009.

[11] POSCO ICT, http://www.poscoict.co.kr.

[12] S. Cho, K. Koo, B. You, T.-W. Kim, T. Shim, and J.S. Lee, "Development of the loader software for PLC programming," In Conference of the Institute of Electronics Engineers of Korea, Vol.30, pp.959-960, 2007.

[13] PLCopen for efficiency in automation, http://www.plcopen.org.

[14] KAERI (Korea Atomic Energy Rearch Institute). Fromal SRS for Reactor Protection System, KNICS-RPS-SVR131-01 Rev.00, 2005.

[15] Junbeom Yoo, Sungdeok Cha, Chang Hwoi Kim, and Duck Yong Song, "Synthesis of FBD-based PLC Design from NuSCR Formal Specification," Reliability Engineering and System Safety, Vol.87, No.2, pp.287-294, 2005.

[16] Junbeom Yoo, Eunkyoung Jee, and Sungdeok (Steve) Cha, "Formal Modeling and Verification of Safety-Critical Software," *IEEE Software*, Vol.26, No.3, pp.42–49, May/June 2009.

Junbeom Yoo, EunKyoung Jee, and Sungdeok Cha, "A Verificatin Framework for FBD based Software in Nuclear Power Plants," In *The 15th Asia Pacific Software Engineering Conference (APSEC)*, pages 385–392, 2008

Modeling of Domain-Specific ECA Policies

Raphael Romeikat
University of Augsburg
Institute of Compter Science
Augsburg, Germany
romeikat@ds-lab.org

Bernhard Bauer
University of Augsburg
Institute of Compter Science
Augsburg, Germany
bauer@ds-lab.org

Henning Sanneck
Nokia Siemens Networks
Research
Munich, Germany
henning.sanneck@nsn.com

Abstract

Policy-based management is a flexible approach for the management of complex systems as policies make context-sensitive and automated decisions. For their effective development it is desired to specify policies at a high level of abstraction initially and to refine them until they are represented in a machine-executable way. We present an approach for the modeling of event-condition-action (ECA) policies at different levels of abstraction. A relational algebra is used to formally specify and validate the models at each level. Finally, executable policy code is generated from the low-level models. The approach is generic as it can be applied to any domain and supports a flexible number of abstraction layers. It is applied to the network management domain and demonstrated with policies for coverage optimization in a mobile network.

Keywords–policy-based management; model-driven engineering; network management

1. Introduction

Increasing complexity of information systems complicates their development and management. This evolution calls for changes in the way those systems are built. The aspect of development is addressed by the model-driven engineering (MDE) approach, which moves the focus from a code-centric to a model-centric point of view [1]. Models raise the level of abstraction and reduce complexity by separating different concerns of a system from each other. Models in MDE are no longer used for discussion and documentation purposes only, but they are used as primary artifacts from which implementations are generated [2]. Increasing complexity also affects the management of information systems at runtime. This issue is addressed by the idea of self-organizing systems. One example is the Autonomic Computing Initiative (ACI) by IBM, which proposes self-manageable systems in order to reduce human intervention necessary for performing administrative tasks [3].

Policies represent a promising technique for realizing autonomic capabilities within managed objects as they allow for a high level of automation and abstraction. Policy-based management has gained attention in research and industry as a management paradigm as it allows administrators to adapt the behavior of a system without changing source code or considering technical details. A system can continuously be adjusted to externally imposed constraints by changing the determining policies [4]. A well-known application domain is network management, where policies are widely used for performing configuration processes. The usage of policy-based systems for the management of mobile networks was recently considered in [5–10].

The event-condition-action (ECA) model is a common way to specify policies. ECA policies represent reaction rules that specify the reactive behavior of a system. An ECA policy correlates a set of events, a set of conditions, and a set of actions to specify the reaction to a certain situation. The conditions are evaluated on the occurrence of an event and determine whether the policy is applicable or not in that particular situation. The actions are only executed if the conditions are met. Multiple policy frameworks share this model as for example Ponder2 [11].

Policy-based management is a layered approach where policies exist at different levels of abstraction. The number of abstraction layers depends on the system to manage. Strassner defines a flexible number of abstraction layers as the Policy Continuum [5]. The idea is to specify and manage policies at each level in a domain-specific terminology, and to refine them from a business level down to a technical level. To support this we present a generic approach for the modeling of domain-specific ECA policies at different levels of abstraction. Policies are specified at the highest level initially and iteratively refined by re-writing them with the means of the lower levels until they are represented in a machine-executable way.

This paper is structured as follows. Section 2 describes how domain-specific policies are modeled at different levels of abstraction before executable code is generated. Section 3 provides an example from a case study. Related work is discussed in section 4. The paper concludes with a summary and future work in section 5.

2. Modeling

We use different models at different abstraction layers in order to specify policies as indicated in [12] and illustrated in figure 1. The domain model allows domain experts to specify concepts of a domain or system. The policy model allows policy experts to specify policies that are used to manage the system. The linking model allows policy and domain experts to link the policy model to the domain model in order to use the domain-specific concepts within the policies. These models are actually parts of one large model. For each of them a metamodel exists that defines the structure of the model. Two layers *i* and *j* are shown exemplarily in figure 1 with layer *i* providing a higher level and layer *j* providing a lower level of abstraction. Actually, the approach supports a flexible number of abstraction layers. The lowest layer represents the models such that executable policy code can be generated from them.

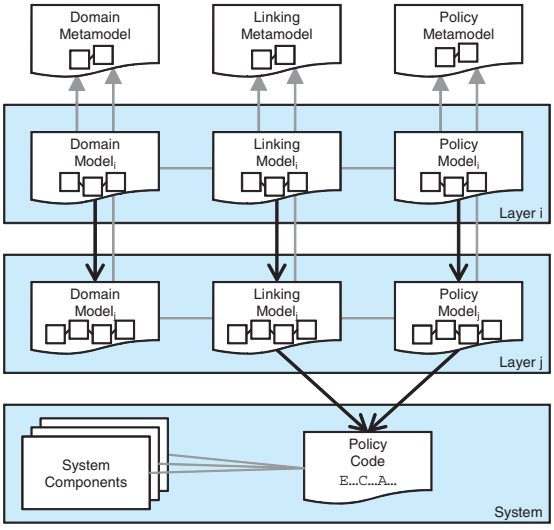


Figure 1. Overview

We are develpoing a relational algebra to formally specify the domain, policy, and linking metamodels, i.e. the abstract syntax of the models. The algebra will be used to validate whether model instances conform to the respective metamodel. For this purpose, a concrete syntax should provide a transformation into the relational algebra. Excerpts of the algebra are presented in this paper.

2.1. Domain Modeling

Different expert groups are involved in the management of a system such as business managers or system administrators. Depending on their focus and their background, members of an expert group have a particular view on the system and they use special terminology to describe their knowledge. The *domain* represents a common understanding of those expert groups and covers the context of a system at the different layers.

Any relevant information about the domain is covered by the *domain model*. The domain model covers the domain-specific concepts across all layers and specifies which concepts are available. Its purpose is to specify domain knowledge independently from any policies, which will later control a system in that domain. Thus it represents the basis for building policies, which will then use domain-specific concepts in their event, condition, and action parts. The domain model offers a particular view at any layer, which only contains the part of the domain model that is relevant at the respective layer. The domain model is an instance of the *domain metamodel*, which allows to specify domain models in a way that is more expressive than just a domain-specific vocabulary and close to the structure of an ontology. For this purpose, the metamodel represents domain-specific knowledge as shown in figure 2. It represents the abstract syntax of the domain, i.e. it defines the structure of the domain model.

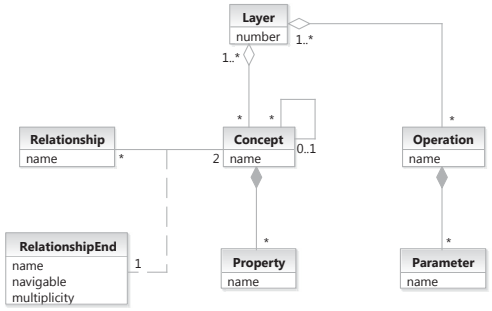


Figure 2. Domain metamodel

The domain metamodel contains the following entities. *Concepts* and *operations* in the domain metamodel have a name and are assigned to *layers*. The layers specify at which levels of abstraction concepts and operations are available. Each layer contains an arbitrary number of concepts and operations. Each concept and operation is assigned to one layer at least. A concept may have named *properties* assigned and each property belongs to one concept. An operation may have named *parameters* assigned and each parameter belongs to one operation. Any two concepts may be assigned to each other by named *relationships*.

The connection points between a relationship and a concept are called *relationship ends* and have a name, a navigability, and a multiplicity.

The domain metamodel allows to build a tree-like hierarchy of concepts. A concept may be assigned other concepts as subconcepts and is called their superconcept in this case. A subconcept inherits the properties and relationships from its superconcept. Thus any property and relationship of a superconcept is also available at its subconcepts. Of course, a subconcept may be assigned additional properties and relationships which are not assigned to its superconcept. A subconcept does not interit the layers from its superconcept. This allows to extend the concept hierarchy by specifying additional subconcepts at lower layers. If subconcepts also inherited the layers from their superconcept, they would be available at the higer layer as well, which is not intended.

2.2. Policy Modeling

Any information about the policies is covered by the *policy model*. The policy model offers a particular view at any layer, which only contains the part of the policy model that is relevant at the respective layer. The policy model is an instance of the *policy metamodel*, which contains the essential aspects required to specify ECA policies. It is shown in figure 3 and represents the abstract syntax of policies, i.e. it defines the structure of the policy model.

Various policy languages are available for different domains and application areas. A policy language usually provides some structured notation for policies and can be interpreted by an execution engine. Typically, such engines provide means to cope with priorities and conflict resolution, which reduces complexity for the developer. When developing a policy-based system, a decision for a policy language has to be made at some point. Unfortunately, no policy language is general and powerful enough to meet the requirements of any system in any domain. It is desired to specify policies independently from a particular policy language and to generate code for a particular language. For this purpose, we analyzed some well-known policy languages such as PonderTalk [11], KAoS [13], and Rei [14] and integrated them into the policy metamodel.

The policy metamodel contains the following entities. *Policies* in the policy metamodel have a name and are active or not. They are assigned to *layers* to specify at which levels of abstraction they are available. Each layer contains an arbitrary number of policies. A policy is assigned to one layer at least. A policy has at least one named *event* assigned. A policy optionally has a named *condition* assigned. A condition is one out of the following boolean expressions. A *binary expression* performs a comparison, which is one out of *equal*, *greater*, *greater or equal*, *lower*, and *lower or equal*. A *negation expression* negates another expression.

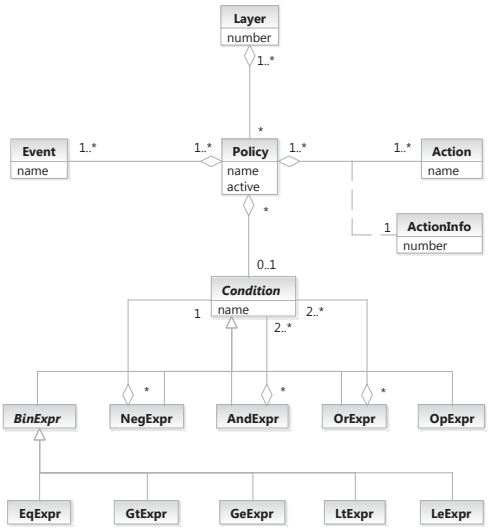


Figure 3. Policy metamodel

An *and expression* combines two other expressions as conjunction. An *or expression* combines two other expressions as disjunction. A special type of expression is the *operational expression*, which refers to some operation and uses its return value as the result of the expression. A policy also has at least one named *action* assigned. The sequence of action execution within that policy is specified by assigning a *number* to each action within that policy.

2.3. Domain-Specific Policy Modeling

Any information about how domain-specific information is used within the policies is covered by the *linking model*. It specifies how the domain and the policy model are linked to each other. For this purpose, it allows to create links from the entities in the policy model to the entities in the domain model at the respective layers. The linking model offers a particular view at any layer, which only contains the links that are relevant at the respective layer. The linking model is an instance of the *linking metamodel*, which provides means to create links from the policy model to the domain model as shown in figure 4. It represents the abstract syntax of the links, i.e. it defines the structure of the linking model.

An event in the policy metamodel corresponds to a concept in the domain metamodel that is used as event. The properties of that concept are regarded as the properties of the event. Multiple events may correspond to the same concept. A binary expression in the policy metamodel compares two arguments with each other. The simplest form of an argument is a literal value. A property in the domain metamodel whose concept is used as event of the respective policy may also be used as argument. This allows to use contextual information within the policy that was passed as

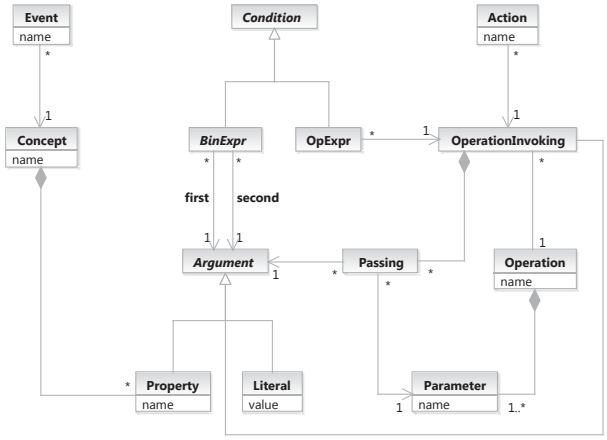


Figure 4. Linking metamodel

event parameter. Another form of an argument is the return value of an invoked operation in the domain metamodel. When invoking an operation, one argument may be passed to each parameter of the operation again. An operational expression in the policy metamodel invokes an operation in the domain metamodel and uses the return value of that operation as value of the expression. An action in the policy metamodel also invokes an operation in the domain metamodel.

The relational algebra imposes some restrictions on the models. One example is the usage of contextual information within a policy. Contextual information is usually passed to a policy via its events. The event properties contain information to be used within the policy and can be referenced in the policy condition and action. It is important that only properties are used within a policy that are visible for that policy. A property is visible for a policy if its concept is used as event of that policy. This restriction is covered by (1) to (3) in the relational algebra.

$$arg \in argumentsOfCondition(cd) \Rightarrow (arg \notin Properties \vee \forall po \in policiesOfCondition(cd). arg \in visibleProperties(po)) \quad (1)$$

$$arg \in argumentsOfAction(ac) \Rightarrow (arg \notin Properties \vee \forall po \in policiesOfAction(ac). arg \in visibleProperties(po)) \quad (2)$$

$$visibleProperties : Id_{Po} \rightarrow \mathcal{P}(Id_{Pr}) \quad po \mapsto \bigcup_{co \in visibleConcepts} propertiesOfConcept(co) \quad \text{with } visibleConcepts = \bigcup_{ev \in eventsOfPolicy(po)} conceptOfEvent(ev) \quad (3)$$

2.4. Code Generation

Refinement is performed by subsequently providing the models at the lower layers until they are represented in a machine-executable way. The domain model then represents the necessary concepts of the underlying system components and the policy and linking models represent policies that control the behavior of those components. Executable code in a policy language can now be generated from the models. This applies to any language that is able to express ECA policies as defined by the policy metamodel.

For this purpose, model transformations generate the policy code in a fully automated way. This involves model-to-model and model-to-text transformations, which first transform the policy and linking models into an intermediate model representation of the target language and then into executable policy code. Transformation is realized as proof of concept for the Ponder2 policy framework [11], which was developed at Imperial College over a number of years. Details of the transformation are presented in [15].

3. Case Study

Management of information systems is a complex task especially in the management of mobile networks. Complexity arises from the distributed architecture of the underlying cellular network with its high number of network elements (NEs) to be deployed and managed and from interdependencies between their configurations.

Operation, administration, and maintenance (OAM) of a mobile network is usually based on a centralized information system which is organized in different management domains. Configuration management (CM) deals with a consistent configuration of all NEs, performance management (PM) analyzes the efficiency of the current network configuration and seeks a more efficient one, and fault management (FM) detects and resolves errors that occur in the network. Any management domain focusses on different aspects and has a special view onto the network and the configurations of the NEs.

3.1. Policy-Based Coverage Optimization

One important management task within any cellular network is coverage optimization. The cells of the network should always provide a complete coverage without areas having no coverage at all. The detection and resolution of converage holes is a complex task as any of the CM, PM, and FM domains are affected. The coverage area of each cell is determined through multiple factors.

- **Position**: Location and direction of an antenna are the main determining factors for the coverage area of a cell. These parameters are preplanned and are almost

not adjustable after deployment as this would require expensive human on-site intervention.

- **Transmission power**: Later adaptations of the transmission power (TXP) have a direct impact on the size of a cell's coverage area. Adaptations can be performed remotely through the operation and maintenance system.
- **Antenna tilt**: The tilt describes the angle between the antenna and the ground. A rough mechanical adjustment is done when the antenna is deployed. Adaptations of the antenna tilt are performed through remote electrical tilt changes (RET) within the antenna.

In order to optimize coverage within the network, usually a sequence of power and tilt adaptations is used since they can be performed remotely. After each change the situation is re-evaluated by analysis of measurement reports. The results of this evaluation are used to determine whether additional adaptions are required or not. Due to the cellular structure of the network, power and tilt adaptations have a strong impact. Adaptions at a single cell may have impact on adjacent cells and may result in a conflict and thus undesired state of the network.

The dependencies between power and tilt adaptions require subsequent adaptions to be coordinated with each other in order to ensure that no logical errors, oscillations, or even deadlocks occur in the configuration. For this purpose, a policy-based coordination mechanism was developed that takes decisions in an automated way [9, 10]. The coordination policies express the decision logic to determine if a change request should be acknowledged, rejected, or rescheduled, and if previously executed requests should be rolled back. The policies are represented at multiple abstraction layers. The highest layer represents a management point-of-view whereas the lowest layer represents a specific implementation for the underlying management system [16] that uses Ponder2 [11] as policy framework.

3.2. Policy Modeling and Code Generation

The behavior of coverage optimization is controlled by a set of ECA coordination policies. For this purpose, a domain model is specified that covers the relevant concepts for coverage optimization. This happens at a high layer, i.e. from a functional point of view initially. At the same layer the necessary coordination policies are specified as a policy model and they are linked to the domain model with a linking model. Figure 5 shows a coordination policy that reschedules tilt change requests if a power change is already active. A simplified graphical notation with UML models and textual annotations is used as concrete syntax.

In order to enable a technical view, a refined domain model is specified at a lower layer that represents the underlying management system. A refined policy and a refined linking model is specified at the same layer in accordance with the refined domain model. Figure 6 shows the refined coordination policy, which provides the same functionality but uses the technical concepts of the underlying management system. Through refinement, some high-level entities must be replaced with the respective low-level ones, e.g. the *powerChangeRequest* and *tiltChangeRequest* concepts with the *reconfRequest* one or the *isPowerChangeActive.cell* and *isTiltChangeActive.cell* parameters with the *isReconfActive.property* one. Furthermore, information is added by means of the additional parameter *isReconfActive.type* and the passed value *"TXP"* in the invoking of the *isReconfActive* operation. The policy is now represented in a machine-executable way.

Finally, the refined models are automatically transformed into the executable Ponder2 code shown in listing 1. The respective model transformations are provided in [15]. The resulting code is directly used in the underlying management system.

```
1 policy := root/factory/ecapolicy create.
2 policy event: root/event/reconfRequest;
3        condition: [ :reconfRequest.id :reconfRequest.
              type :reconfRequest.property :reconfRequest.
              changeValue | root/op isReconfActive:"TXP"
              property:reconfRequest.property ];
4        action: [ :reconfRequest.id :reconfRequest.type :
              reconfRequest.property :reconfRequest.
              changeValue | root/op
              rescheduleReconfRequest:reconfRequest.id ].
5 root/policy at:"coordinationPolicy" put:policy.
6 policy active: true.
```

Listing 1. Generated Ponder2 code (excerpt)

4. Related Work

This section summarizes comparable approaches for the modeling of policies. All approaches are compared to each other with regard to their features and outlined in table 1. A + in the table indicates that a feature is available in an approach, a *o* indicates a limited feature, and a - indicates that a feature is not available at all.

The authors of [17] present a model-driven approach to design policies and integrate them into the software development process. The approach is based on MDE concepts and uses a UML profile for modeling policies. The general policy modeling language (GPML) supports ECA policies amongst different types of policy. The ability to define a particular vocabulary allows to adapt policies to different domains. Policies are modeled at a low level of abstraction and cannot be refined. Model transformations are used to map GPML policies via an interchange format to existing policy languages. No formal specification is provided.

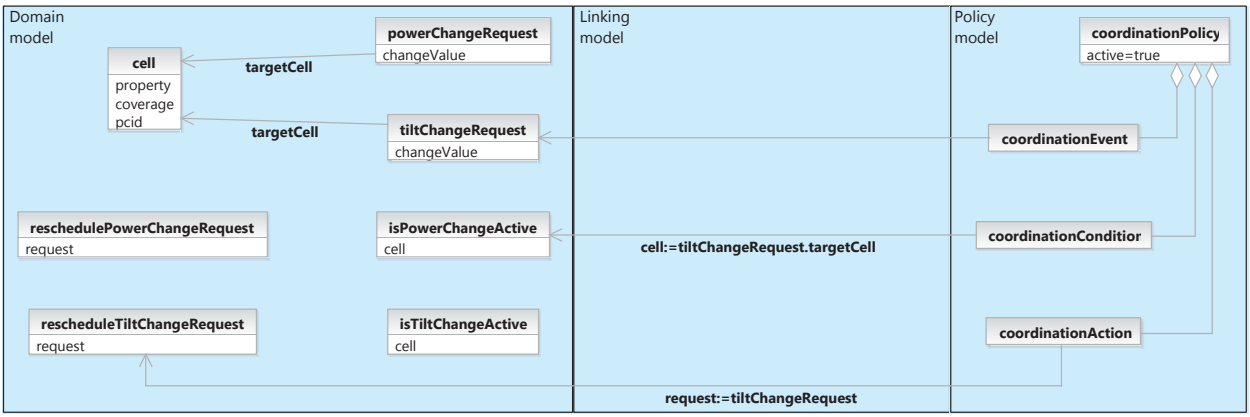


Figure 5. High-level model (excerpt)

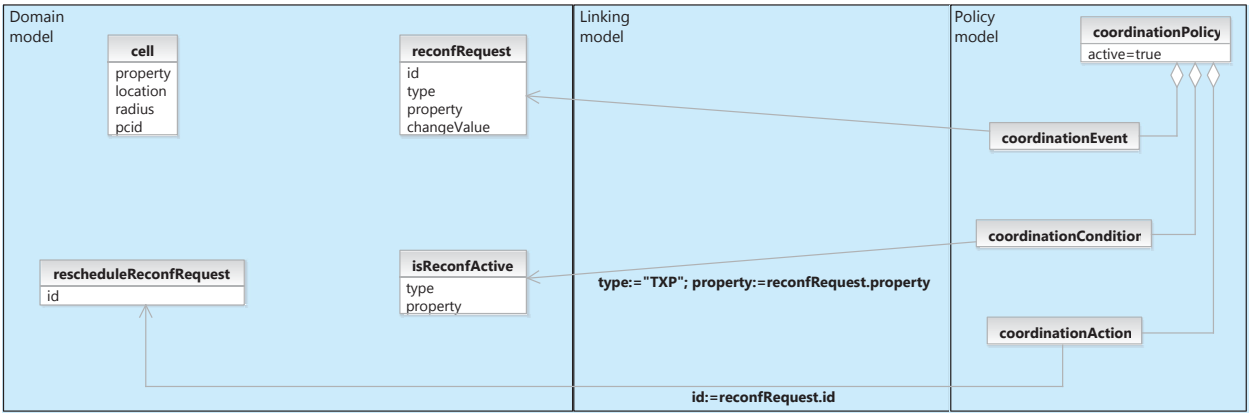


Figure 6. Low-level model (excerpt)

The CIM Policy Model [18] by the Distributed Management Task Force (DMTF) addresses the management of complex multi-vendor environments with a huge number of heterogeneous devices. Policies are specified in a language-independent way and abstract from hardware characteristics. A UML profile is provided for the graphical representation of policies. The CIM Policy Model is a domain-specific model with a focus on network management. Different abstraction levels and policy refinement are not supported. The approach does not address code generation for existing policy languages. A formal specification is not provided.

DEN-ng [6] by the TM Forum provides an information model for system entities and policies to manage those entities. It allows entities and policies to be modeled in an implementation-independent way while omitting technical details. ECA policies are modeled in a UML diagram. DEN-ng allows to specify a domain and resources within that domain on which policies operate. DEN-ng is based on the Policy Continuum and considers policies at different levels of abstraction. Specification of policies is addressed but policies cannot be refined automatically into a lower level of abstraction, nor transformed into an existing language. No formal specification is provided.

5. Conclusion

A model-based approach to the specification of ECA policies was presented in this paper. The usage of models allows to specify policies at a high level of abstraction initially and avoids the direct implementation of policies at a technical level. The approach is novel as it is generic with respect to the domain, to the language, and to the number of abstraction levels and nevertheless allows to generate executable code. The separation of knowledge into different models or model parts allows for an effective collaboration of domain and policy experts. The usage of different abstraction layers faciliates the collaboration of business and technical experts. A relational algebra is being developed to precisely define the abstract syntax of the models and allow

	GPML [17]	CIM Policy Model [18]	DEN-ng [6]	Modeling of Domain-Specific ECA Policies
ECA policies	+	o	+	+
Graphical modeling	+	+	+	+
Language-independent	+	+	+	+
Customizable domain	+	-	+	+
Abstraction levels	-	-	+	+
Formal specification	-	-	-	o
Automated refinement	-	-	-	-
Code generation	+	-	-	+

Table 1. Comparison of related work

for their validation, which is a prerequisite for the transformation into executable code.

The possibility to generate code eliminates the dependency from a particular policy language as the same models can be used to generate code for various languages. A prototype of a graphical policy editor that supports code generation for Ponder2 has already been developed [19]. The case study showed that a purely graphical syntax for the models might be confusing as diagrams take a lot of space in complex scenarios. An effective textual syntax is subject to future work. A representation of the models in the relational algebra for purposes of validation should be generated automatically. Automation of the refinement process is also subject to future work. Currently, the lower-level models are created manually based on the higher-level ones. However, it should be sufficient to specify the refinement of the domain model once and then apply that refinement to the policy and linking models in order to generate their refined representation whenever they are created or modified. In the end the objective is to automatically reflect changes of the high-level models in their low-level implementation.

References

[1] J. Bézivin, "On the Unification Power of Models," *Software and Systems Modeling*, vol. 4, no. 2, pp. 171–188, May 2005.

[2] D. W. Flater, "Impact of Model-Driven Standards," in *35th Annual Hawaii International Conference on System Sciences (HICSS)*. IEEE CS, January 2002, p. 285.

[3] J. O. Kephart and D. M. Chess, "The Vision of Autonomic Computing," *Computer*, vol. 36, no. 1, pp. 41–50, January 2003.

[4] N. Damianou, N. Dulay, E. Lupu, and M. Sloman, "The Ponder Policy Specification Language," in *2nd Workshop on Policies for Distributed Systems and Networks (POLICY)*. Springer LNCS, January 2001, pp. 18–38.

[5] J. Strassner, *Policy-Based Network Management: Solutions for the Next Generation*. San Francisco, CA, USA: Morgan Kaufmann Publishers, 2003.

[6] J. Strassner, "DEN-ng: Achieving Business-Driven Network Management," in *8th Network Operations and Management Symposium (NOMS)*. IEEE CS, April 2002, pp. 753–766.

[7] S. van der Meer, A. Davy, S. Davy, R. Carroll, B. Jennings, and J. Strassner, "Autonomic Networking: Prototype Implementation of the Policy Continuum," in *1st International Workshop on Broadband Convergence Networks (BcN)*, April 2006, pp. 1–10.

[8] T. Bandh, H. Sanneck, L.-C. Schmelz, and G. Carle, "Automated Real-time Performance Management in Mobile Networks," in *1st WoWMoM Workshop on Autonomic Wireless AccesS (IWAS)*. IEEE CS, June 2007, pp. 1–7.

[9] R. Romeikat, B. Bauer, T. Bandh, G. Carle, H. Sanneck, and L.-C. Schmelz, "Policy-driven Workflows for Mobile Network Management Automation," in *6th International Wireless Communications and Mobile Computing Conference (IWCMC)*. ACM, June 2010, pp. 1111–1115.

[10] T. Bandh, R. Romeikat, and H. Sanneck, "Policy-based Coordination and Management of SON Functions," in *12th International Symposium on Integrated Network Management (IM)*. IEEE ComSoc, May 2011, to be published.

[11] K. Twidle, E. Lupu, N. Dulay, and M. Sloman, "Ponder2 - A Policy Environment for Autonomous Pervasive Systems," in *9th Workshop on Policies for Distributed Systems and Networks (POLICY)*. IEEE CS, June 2008, pp. 245–246.

[12] R. Romeikat and B. Bauer, "Specification and Refinement of Domain-Specific ECA Policies," in *4th International Workshop on Domain-specific Engineering (DsE@CAiSE)*. Springer LNBIP, June 2011, to be published.

[13] A. Uszok, J. M. Bradshaw, and R. Jeffers, "KAoS: A Policy and Domain Services Framework for Grid Computing and Semantic Web Services," *Lecture Notes in Computer Science*, vol. 2995, pp. 16–26, January 2004.

[14] L. Kagal, T. Finin, and A. Joshi, "A Policy Language for a Pervasive Computing Environment," in *4th International Workshop on Policies for Distributed Systems and Networks (POLICY)*, June 2003, pp. 63–74.

[15] R. Romeikat, M. Sinsel, and B. Bauer, "Transformation of Graphical ECA Policies into Executable PonderTalk Code," in *3rd International Symposium on Rule Interchange and Applications (RuleML)*. Springer LNCS, November 2009, pp. 193–207.

[16] T. Bandh, H. Sanneck, and R. Romeikat, "An Experimental System for SON Function Coordination," in *International Workshop on Self-Organizing Networks (IWSON)*. IEEE VTS, May 2011, to be published.

[17] N. Kaviani, D. Gasevic, M. Milanovic, M. Hatala, and B. Mohabbati, "Model-Driven Engineering of a General Policy Modeling Language," in *9th Workshop on Policies for Distributed Systems and Networks (POLICY)*. IEEE CS, June 2008, pp. 101–104.

[18] Distributed Management Task Force, "CIM Policy Model White Paper," DSP0108, June 2003.

[19] University of Augsburg, "PolicyModeler," http://policymodeler.sf.net, August 2009.

A Software Engineering Approach to User-Driven Control of the Microgrid

Mark Allison, Andrew A. Allen, Zhenyu Yang and Peter J. Clarke
School of Computing and Information Sciences
Florida International University
Miami, FL 33199, USA
Email: {malli002, aalle004, yangz, clarkep}@cis.fiu.edu

***Abstract*—**
The smart grid has been proposed as the panacea to address systemic challenges of the over fifty year old legacy electrical grid, the single largest machine on the planet. A core component central to realizing the smart grid concept is the *microgrid*. The microgrid is a self-sustaining entity, capable of data interchange and real-time monitoring and control of its distributed generation, storage and load components.

In this paper we introduce ongoing research that uses a software engineering approach to user-driven control of the microgrid. Our approach uses a domain-specific modeling language (DSML), *MGridML*, and a virtual machine, *MGridVM*, which interprets user-defined models representing domain-level abstractions of the microgrid. MGridML captures high-level representations of pertinent domain features, based on a centralized hierarchical model of demand side energy management. A metamodel for MGridML and a prototype of the MGridVM are presented to show the feasibility and practicality of our approach.

***Index Terms*—Smart Grid, Microgrid, Domain-Specific Modeling Languages, Model-Driven Engineering.**

I. Introduction

The changes in energy consumption patterns are being dictated by rising energy costs and higher demand while increasingly being aware that thes unsustainable energy sources near depletion. These changes manifest themselves as additional requirements of the legacy electrical grid. The United States Department of Energy and similar entities across the globe have been tasked to upgrade the single largest interconnected machine on the planet, the electrical grid [1]. The existing grid, heavily reliant on fossil fuels, has effectively remained unchanged since the early twentieth century and is reaching its functional limits. The *smart grid* is seen as the successor to the lagacy grid and a core component central to realizing the smart grid concept is the *microgrid*. The microgrid is a self-sustaining entity, capable of data interchange and real-time monitoring and control of its distributed generation, storage and load components.

This paper introduces ongoing research that uses a software engineering approach to user-driven control of the microgrid. Our approach uses a domain-specific modeling language (DSML), *MGridML*, and a virtual machine, *MGridVM*, which interprets user-defined models representing domain-level abstractions of the microgrid. A metamodel for MGridML was developed based on a feature analysis of the microgrid and will be briefly described in the paper. To demonstrate the feasibility and practicality of our software engineering approach to energy management within the microgrid we provide a discussion of our prototype. The prototype includes the MGridML modeling environment, the MGridVM and a low voltage laboratory testbed which captures the essence of the microgrid.

This paper is organized as follows: Background technologies to provide context are reviewed in the next section. Section III provides our motivation for embarking on this research direction. The modeling language and virtual machines are described in Sections IV and V respectively. We place this work in perspective by presenting related work in Section VII. Finally we discuss future directions of this work and conclude in Section VIII.

II. Background

In this section we introduce the concepts and technologies required for our research. This background material include energy management within the microgrid and the use of domain-specific modeling languages (DSMLs) to model and realize applications in a specific domain.

A. The Microgrid Concept

The smart grid is the conceptualized solution to the ailing aspects of our existing grid infrastructure. The smart grid is challenged with balancing demand and supply through real-time communication and distributed generation accommodating renewable sources with self-healing capabilities [2]. All of this while reducing the carbon footprint and still maintaining the affordability of energy to the consumer [2]. It is expected that the smart grid will achieve much of the proposed goals through the integration of the microgrid concept.

Lasseter et al. [3] define the Consortium for Electric Reliability Technology Solutions (CERTS) concept of a *microgrid* as an aggregation of loads and microsources operating as a single system providing both power and heat. It is further stated that the majority of the microsources must be power electronic based. The key features of the microgrid include power electronics, control and communications capabilities that allows it to function as a semi-autonomous power system. The microgrid is expected to maintain stable operation and perform as a good citizen of the grid [4].

The production of electrical energy within the current electrical grid is demand based whereby the energy consumed drives the amount of energy being generated as storage is generally infeasible. Additionally, the means of energy production is centralized and far removed from the area of demand leading to heavy energy losses [5]. The microgrid advantage is in the proliferation of distributed energy resources (DERs) and energy storage and is the focus of our research. To further assist readers who maybe unfamiliar with the microgrid concept, we define the following terms:

- *Islanding* - In this state the microgrid operates independently of an external source.
- *Load* - A device which consumes electrical power.
- *Point of Common Coupling (PCC)* - The point at which the utility may be disconnected from the microgrid.
- *MGCC* - The Microgrid Central Controller, responsible for centrally directing energy flow within the microgrid.

B. Domain Specific Modeling Languages

Domain specific languages (DSL's) or little languages [6], are so called as they are usually small with well focused expressive power. The benefits of the DSL approach has been extensively researched. Sirer et al. [7], states that DSLs bring the solution domain closer to the problem domain, improve testability and simplifies maintenance. Empirical data from [8] suggests that the use of DSL's increases reliability, usability and flexibility. Bell Labs using the Family-Oriented Abstraction, Specification and Translation (FAST) approach to domain engineering reports at least a factor of four increase in productivity. [9]. DSLs were originally text-based, however there has been an increase focus on graphical DSLs or *domain-specific modeling languages* (DSMLs) since there are now tools to easily create modeling environments for DSMLs [10].

In order for this solution to apply, the problem domain has to be suitably qualified. According to [11], candidate domains should be reasonably stable and economically viable. We argue that the scope of our domain, microgrid energy management, falls within these parameters due to its maturity and its relevance within the global energy vision. Stahl et al. [12] outline the steps required to create a DSML, these include (1) an analysis of the domain, (2) the creation of a meta-model consisting of the abstract syntax of the language and the static semantics, (3) the concrete syntax for the language, (4) the dynamic semantics of the language. We applied Stahl et al. approach to define and design the MGridML language.

III. Motivation

The motivation for initially developing the MGridML and MGridVM was partially the result of observing the shortcomings of implementing a distributed generation and storage functionality within a local single family dwelling to facilitate islanding. We observed several issues which plague the realization of a robust microgrid capable of responding to instability in power distribution or the users economic preference. State of the art control systems typically require domain expertise far beyond that of the end-user of the system and are not programmable by the end-user. In the sequel we describe a scenario that typifies the challenges of an end-user faces when implementing a mircogrid.

A. Motivating Scenario

The primary actor in the scenario is Mary and the equipment in the microgrid comprises of distributed generation in the form of a photovoltaic generator, a storage system (batteries), and three loads, one of which is considered to be critical. The microgrid is connected to the local utility via a point of common coupling (PCC) which is capable of isolating or islanding the microgrid.

Microgrid Scenario: Our scenario begins when the utility grid is experiencing power fluctuations as observed in Mary's flickering lights and the erratic behavior of her appliances. Mary is not aware of how long this situation will last and is afraid of damage to costly equipment so she decides to remove herself from the grid and use stored energy. Mary is unsure of how long her energy storage will last. Mary, using a hardware solution, manually disconnects the microgrid at the PCC then engages the storage. Power is restored but Mary is left with an uneasy feeling. How long before she will have to switch to her batteries and how long will those last? Maybe she should turn off the non-critical appliances. She looks worryingly at the window at her neighbors house to see if power is restored.

Although this scenario is based on a manual approach, it raises critical questions as to the base requirements and issues yet to be addressed by alternate approaches in the context of user facilitated energy management. Our methodology conceives the solution as a black box then describe its core functionality and desired qualities. We postulate that the black-box requires realtime data and component state information from sources internal and external to the microgrid. It should conceptually describe Mary's directives in an intuitive manner and act accordingly within a reasonable time-frame, keeping her updated of the systems current state and near future projections.

Storage depletion estimates at current usage levels during islanding would assist the user, Mary, in modifying her load profile in the short term. It should alert her whenever the utility power level is stable as defined by some threshold and have the ability to automate a sequential restoration to pre-islanding operation. There is the need for a planning mechanism to optimize performance; the blackbox should be capable planning according to load and source forecasting and not depend on the lay user to perform complex calculations to facilitate optimization.

Should the user require the capacity of optimizing for economic purposes, the blackbox solution necessitates real-time monitoring of tariff information from the smart grid as in the case where Mary chooses to disconnect from the utility if the rates become too high and use stored energy if she anticipates sunshine the next day. With the advent of time-varying pricing, flattening the demand curve is a paramount consideration. Should additional hardware be intro-

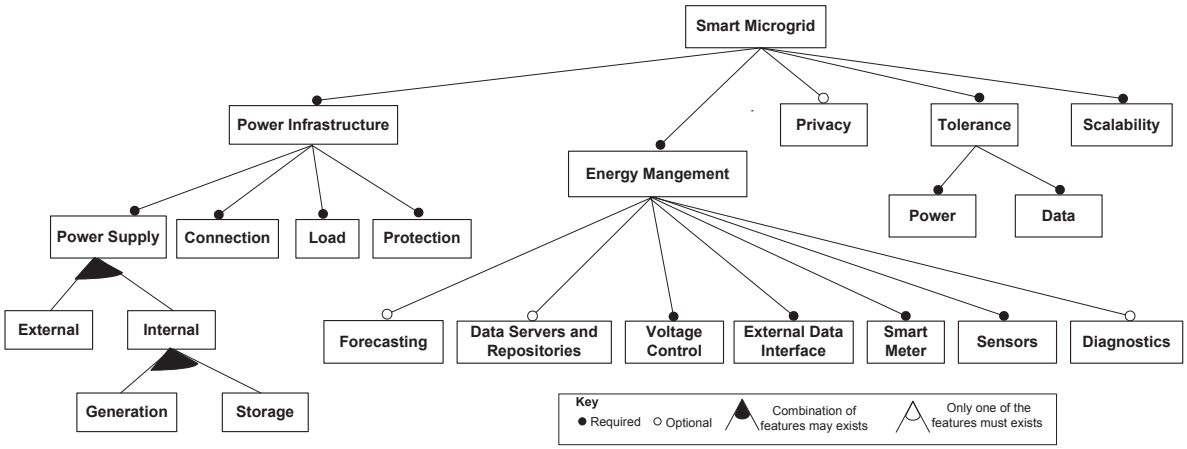


Fig. 1. Feature diagram for the microgrid.

duced within the microgrid, the black box should absorb and compensate. Lastly, thresholds, which if exceeded could result in equipment damage or raise safety concerns, will need to be identified and anticipated during operation. The consolidation of these considerations become the basis of our feature model addressed next.

B. Domain Analysis

A crucial aspect within the development of any DSML is a detailed and methodological introspection of the application domain to ascertain the predominant features and concepts [13]. Feature analysis is at times an ad-hoc learning process requiring constant and consistent refinement as domain objects are identified, abstracted and structured [13]. Following the Feature Oriented Domain Analysis methodology (FODA) [14], introspection begins by defining the scope of our domain.

The developmental methodologies engaged within the research of the microgrid required a familiarity with the domain terminology, to communicate effectively with domain experts for the purposes of verifying the abstraction of the essence of the domain. The primary artifact of analyzing the microgrid is the feature diagram [15], as seen in Figure 1, which shows the composite and atomic features of the microgrid.

Figure 1 shows the feature diagram for the mictrogrid. The feature diagram shows that a microgrid must have a power infrastructure, an energy management component, provide tolerance and be scalable. From our analysis privacy is currently an optional feature. Using the key for the symbols, shown to the lower right of the diagram, the remainder of the diagram can be interpreted. Note that power supply may have an external power supply, internal power supply, or both.

IV. The Microgrid Modeling Language

The development of the graphical DSML, MGridML was accomplished utilizing Microsoft DSL tools, a part of the the Visual Studio SDK [16]. The DSML is defined by its metamodel which comprises of the abstract syntax and static semantics. Figure 2 shows the partial abstract syntax for MGridML. A model of the microgrid is referred to as an

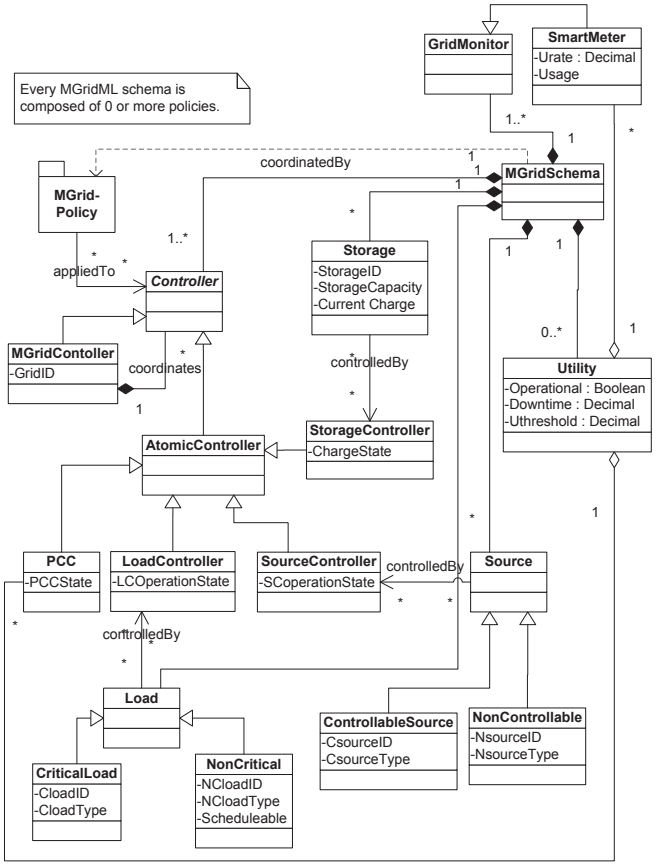


Fig. 2. Partial abstract syntax for MGridML.

MGrid schema. The MGrid schema is composed of one or more grid monitors, one or more controllers, one or more storage units, one or more loads, one or more sources and zero or more utilities. The schema may also contain zero or more MGrid policies not describe in this paper. We use the polices to constrain the behavior of the system based on the environment or end-user preferences.

The CERTS microgrid concept provides the basis for the

architecture and acceptable behavior of elements within our models. Key to the structure of the modeling language is its mapping to unique critical functions. Figure 3 shows an MGrid schema using a concrete syntax produced using the matamodel in Figure 2. Due to space limitations we do not define the concrete syntax for MGrdiML. The computer shape in the middle of the model represents the MGrid controller that coordinates the activities of loads (lower left of the figure), the sources (right of the figure) and the PCC shown as the smart meter. Policies are attached to the various entities in the model. For example one policy that is defined for safety is that a storage source cannot be charging and discharging at the same time. This policy is shown in the lower right corner of the model. Note that the model shown in Figure 3 would be created by an expert, such as a technician.

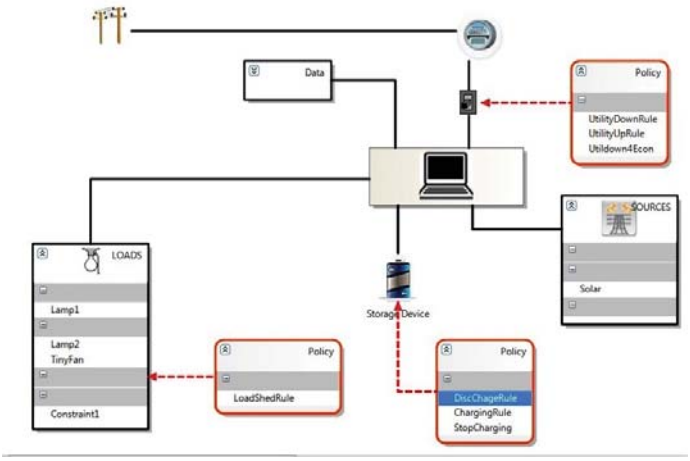


Fig. 3. MGrid schema for the scenario.

V. The Virtual Machine

In this section we describe the MGridVM which follows a similar structure to the Communication Virtual Machine (CVM) developed by Deng et al. [17]. The MGridVM uses a four layered architecture similar to CVM.

Figure 4 shows the structure of the MGridVM. The key on the right side of the figure uses a color code to identify with components belong to which layer. The layers of the MGridVM are described as follows.

Microgrid Management Interface (MMI) - Gives the user the ability to conceive, describe and obtain the feasibility of desired behaviors and contingencies of the underlying physical components of the microgrid through MicroGridML, the DSML described earlier. Facilitation is at an abstract level to be intuitive yet expressive enough to describe most of the configurations and functionality of the microgrid. The output to the underlying layer is a feasible schema or instantiation of the grid model with artifacts which describe possible transformations. The user may chose to to run a simulation based on the current hardware configuration over a specified period of time. In this case the hardware negotiation layer is disconnected and a mockup based on repository data is used as

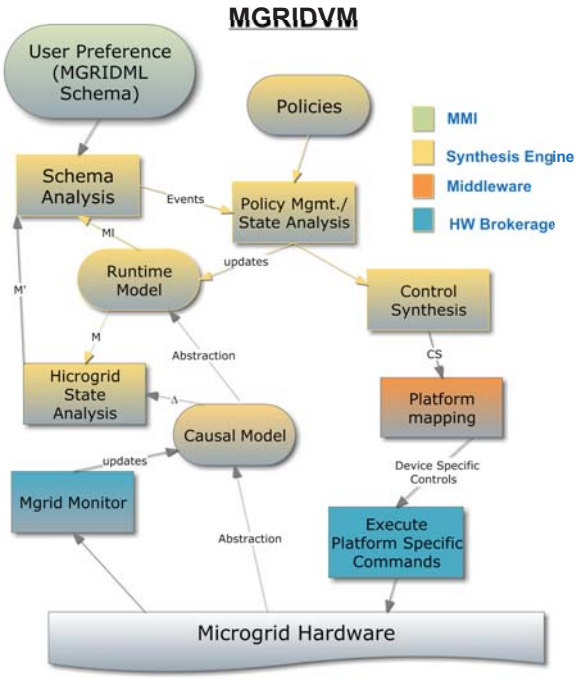


Fig. 4. The MGrid Virtual Machine

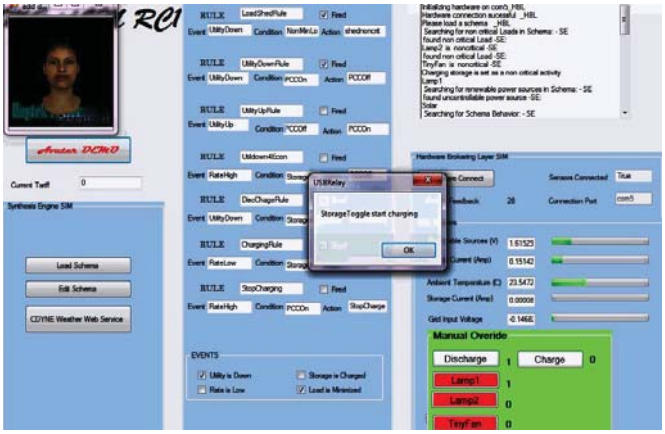


Fig. 5. MGridVM Management Interface.

data-source. Figure 5 shows the user interface for our current prototype for the MMI.

Synthesis Engine (SE) - Transforms the behaviors and contingencies in the declarative MicroGridML into an executable MicroGrid Control Script (MGCS). The actions and commands inherent within the MGCS, serve as directives to manage the physical grid components. This layer contains a runtime model which reflects the current state of the microgrid. Changes to the runtime model are either user initiated, whereby the user sends a new model to be implemented, or by changes in the microgrid hardware states. A schema analyzer transforms the difference between the new model and the executing model into system events. We defined our policies in terms of a targeted triple of Events, Conditions and Actions; system events determine which policies become active, but only after policy manager scrutiny. The final component of

this layer is a state analyzer which updates the runtime model to be consistent with the state of the hardware.

Grid Control Middleware - Transforms the platform independent control script into hardware specific commands. It is at this layer whereby the virtual machine may amalgamate diverse devices with distinct commands.

Hardware Brokerage - This layer is primarily responsible for executing device specific commands upon the physical hardware below and monitoring the status of its devices.

VI. Validation

To validate the efficacy and feasibility of our approach, we developed a microgrid testbed (see Figure 7). The testbed was based on the running scenario outline earlier in Section III-A, but the abstraction took into consideration the critical features of a microgrid. The testbed operates on approximately 5V DC, eliminating the need for DC/AC PWM inversion for the PV minisource. The MGridVM serially communicates with the testbed via two USB interfaces. The first, a sensory interface, monitors voltage and current levels at specific points in the testbed. The second, a relay bank, actuates component switching. Storage is in the form of one Lithium Ion battery producing at 4.75V and the LV grid source is represented by a 5.0V AC/DC adapter rated at 100mA. Figure 6 shows a schematic drawing of the testbed used to validate the feasibility of our approach.

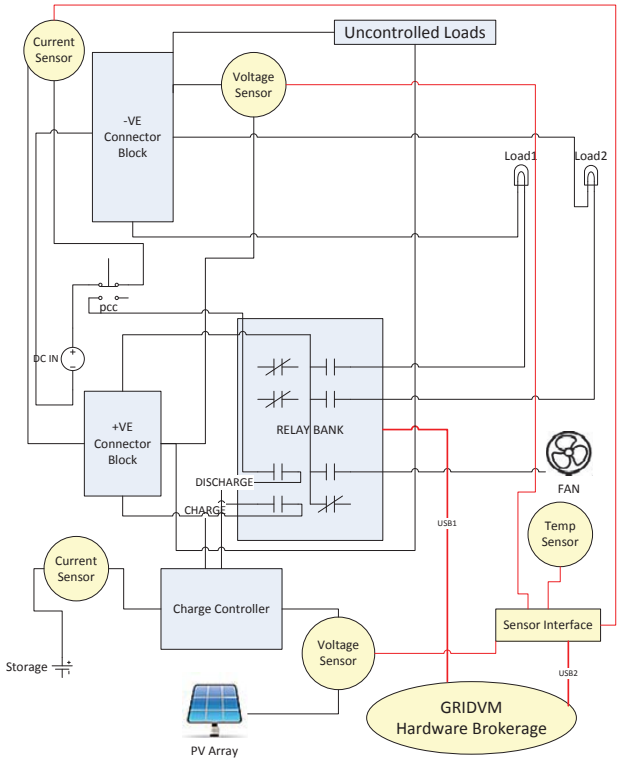


Fig. 6. Schematic of the testbed.

Voltage and current levels within the testbed are continuously monitored by sensors according to a defined data acquisition rate and sent to the sensory interface. The hardware brokerage layer of the MGridVM receives these monitored updates and analyzes the values to determine changes in component states. Values that fall outside the scope of the defined threshold levels indicate an important state change and an appropriate event will be generated. For example, should the grid source voltage level drop to 2.0V, the brokerage layer would receive the new voltage value, compare the value to the predefined threshold of 4.0V and generate a *UtilityDown* event. Based on the policies in effect that are driven by the *UtilityDown* event, the appropriate control script would be generated. Assuming there exists a policy P_n such that:

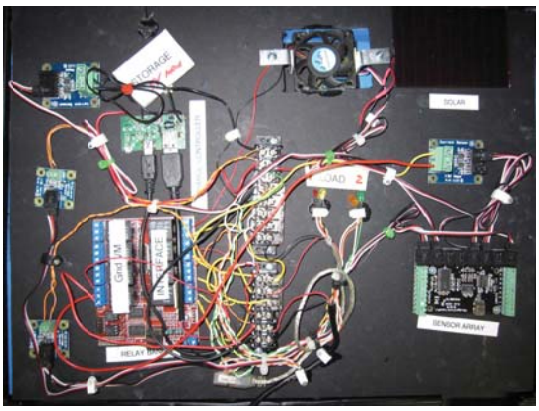


Fig. 7. Hardware implementation for the testbed.

$$P_n = \langle event \rangle\ UtilityDown$$
$$\langle condition \rangle\ StoreLevel > StoreThreshold$$
$$\langle action \rangle\ DischargeBattery()$$

In our scenario P_n would cause a control script to be generated for the storage to be discharged into the mircogrid, assuming the current condition of the storage level exceeds that of the storage threshold level. The round trip is completed when the control script is sent down to the hardware brokerage level to be executed via the USB connection to the relay bank on the testbed itself.

VII. Related Work

There has been significant effort in the research community to address energy management. In general, the challenge of operating a microgrid lies in the fact that at all times the balance between generation and demand has to be maintained. Much of the work in the area of microgrids tend to focus on the electrical aspects such as efficient designs and hierarchical integration [18], [19], [20] of generation and load into existing electric power distribution infrastructure. This work compliments much of the aforementioned research by providing orthogonally, a simplified software engineering approach to support the management of the microgrid through models.

Other researchers have also investigated software approaches to support energy management. Zaidi et al. [21] proposes an approach for self-configuring microgrids that focuses on the demand side management. Individual load are

assigned to intelligent control nodes containing switching and power measuring features that are remotely accessible through wireless radios. Control algorithms are used to identify the type of loads by looking at their power consumption profiles, which are measured by the control nodes and communicated to the micgrogrid controller. Similar to Zaidi's work, our approach leverages intelligent controller to support demand side energy management. Unlike Zaidi's work, where loads are assigned predefined priorities and load isolation decisions are based on the predefined priorities, our work provides a user-centric focus to energy management. End users define load priorities and preferences as user defined policies which influences the weighting of the load isolation algorithms.

Livengood et al. [22], proposes a software energy management systems, an energy box, capable of optimizing residential energy consumption through a suite of stochastic dynamic programming algorithms centered around demand pricing. Livengood et al.'s approach however introduces some complexity for the creation and modification of the management algorithms. Our approach differs in the use of a DSL and an intuitive modeling environment that supports the manipulation of management rules. MgridML captures domain expertise within the language enabling the end user to make fine grained adjustments to the microgrid activities, while still being sheltered from its complexity.

The DSL approach is utilized in Habitation [23], a domain specific language for home automation system design. Similar to our work, a model-driven paradigm is employed, providing a higher level of abstraction to the user of the tool. the Habitation language however, targets the representation and manipulation of loads. MGridML is designed to address the complete energy system with algorithms concerned with the balancing of energy between loads and sources. Additionally, Habitation uses a code generation methodology while MGridVM uses a runtime model interpretation technique to support dynamic reconfiguration of the microgrid.

VIII. Concluding Remarks

In this paper we presented a model-driven technique to address the need for robust, user-driven software overlay for energy management. This approach provides for energy management of the microgrid and is furthermore an enabling technology for automated demand response. Our user-driven approach to energy management considers the role of the microgrid user as critical to overall consumption reduction. Future directions of this research will be geared towards load and source forecasting and extending the virtual machine to facilitate mobility. We will further fine tune our human computer interface according to design dimensions with Embodied Conversational Agent feedback.

ACKNOWLEDGEMENTS

The authors would like to acknowledge support in part by a Florida International University Dissertation Year Fellowship - Andrew Allen, and the NSF under grant HRD-0833093.

References

[1] Litos Strategic Communication, "The smart grid: An introduction," US Department of Energy, Tech. Rep., 2008, http://www.oe.energy.gov/DocumentsandMedia/DOE_SG_Book_Single_Pages(1).pdf (March 2011).

[2] A. Vojdani, "Smart integration," *Power and Energy Magazine, IEEE*, vol. 6, no. 6, pp. 71–79, 2008.

[3] R. Lasseter, A. Akhil, C. Marnay, J. Stephens, J. Dagle, R. Guttromson, A. S. Meliopoulous, R. Yinger, and J. Eto, "White paper on integration of distributed energy resources. the certs microgrid concept," Consortium for Electric Reliability Technology Solutions, prepared for the U.S. Department of Energy, Tech. Rep., April 2002.

[4] N. Hatziargyriou, H.Asano, R. Iravani, and C. Marnay, "Microgrids," *IEEE Power & Energy Magazine*, vol. july, pp. 78–94, 2007.

[5] R. Lasseter and P. Piagi, "Microgrid: A conceptual solution," in *IEEE Power Electronics Specialists Conference*, vol. 6. Citeseer, 2004, pp. 4285–4291.

[6] A. Van Deursen and P. Klint, "Little languages: Little maintenance?" *Journal of Software Maintenance: Research and Practice*, vol. 10, no. 2, pp. 75–92, 1998.

[7] E. Sirer and B. Bershad, "Using production grammars in software testing," in *Proceedings of the 2nd conference on Domain-specific languages*. ACM, 1999, pp. 1–13.

[8] R. Kieburtz, L. McKinney, J. Bell, J. Hook, A. Kotov, J. Lewis, D. Oliva, T. Sheard, I. Smith, and L. Walton, "A software engineering experiment in software component generation," in *Proceedings of the 18th international conference on Software engineering*. IEEE Computer Society, 1996, p. 552.

[9] D. Weiss, "Creating domain-specific languages: The fast process," in *First ACM-SIGPLAN Workshop on Domain-Specific Languages; DSL*, vol. 97, 1997.

[10] D. Forum, "Domain specific modeling," 2011, http://www.dsmforum.org/(March).

[11] M. Simos, "Organization domain modeling (odm): Formalizing the core domain modeling life cycle," *ACM SIGSOFT Software Engineering Notes*, vol. 20, no. SI, pp. 196–205, 1995.

[12] T. Stahl, M. Voelter, J. Bettin, A. Haase, S. Helsen, and K. Czarnecki, *Model-Driven Software Development: Technology, Engineering, Management*, 1st ed. John Wiley & Sons, 2006.

[13] R. Prieto-Diaz, "Domain analysis: an introduction," *ACM SIGSOFT Software Engineering Notes*, vol. 15, no. 2, p. 54, 1990.

[14] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature-oriented domain analysis," CMU, Tech. Rep. CMU/SEI-90-TR-21, Nov 1990.

[15] K. Czarnecki, K. Osterbye, and M. Voelter, "Generative programming," in *Object-Oriented Technology ECOOP 2002 Workshop Reader*. Springer, 2002, pp. 15–29.

[16] S. Cook, G. Jones, S. Kent, and A. Wills, *Domain-specific development with visual studio DSL tools*. Addison-Wesley Professional, 2007.

[17] Y. Deng, S. M. Sadjadi, P. J. Clarke, C. Zhang, V. Hristidis, R. Rangaswami, and N. Prabakar, "A communication virtual machine," in *COMPSAC '06: Proceedings of the 30th Annual International Computer Software and Applications Conference*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 521–531.

[18] Z. Jiang and R. Dougal, "Hierarchical microgrid paradigm for integration of distributed energy resources," in *Power and Energy Society General Meeting - Conversion and Delivery of Electrical Energy in the 21st Century, 2008 IEEE*, July 2008, pp. 1 –8.

[19] J. Driesen and F. Katiraei, "Design for distributed energy resources," *Power and Energy Magazine, IEEE*, vol. 6, no. 3, pp. 30 –40, May 2008.

[20] F. Katiraei, R. Iravani, N. Hatziargyriou, and A. Dimeas, "Microgrids management," *Power and Energy Magazine, IEEE*, vol. 6, no. 3, pp. 54 –65, May 2008.

[21] A. Zaidi and F. Kupzog, "Microgrid automation - a self-configuring approach," in *Multitopic Conference, 2008. INMIC 2008. IEEE International*, Dec 2008, pp. 565 –570.

[22] D. Livengood and R. Larson, "The energy box: Locally automated optimal control of residential electricity usage," *Service Science*, vol. 1, no. 1, pp. 1–16, 2009.

[23] M. Jimenez, F. Rosique, P. Sanchez, B. Alvarez, and A. Iborra, "Habitation: A domain-specific language for home automation," *IEEE Software*, vol. 26, pp. 30–38, 2009.

A Comparative Study of Different Strategies for Predicting Software Quality

Taghi M. Khoshgoftaar
Florida Atlantic University
Boca Raton, Florida 33431
khoshgof@fau.edu

Kehan Gao
Eastern Connecticut State University
Willimantic, Connecticut 06226
gaok@easternct.edu

Amri Napolitano
Florida Atlantic University
Boca Raton, Florida 33431
amrifau@gmail.com

***Abstract*—Various methods have been developed for improving the quality of a software product, especially for high-assurance and mission-critical software systems. One commonly used approach is software quality modeling, in which software practitioners utilize software metrics and defect data collected during the software development process to build defect prediction models that will help to find poor-quality program modules. Those modules predicted to be fault-prone will receive more inspection and testing, thereby improving their quality. Efficacy of defect prediction models is influenced by relevance between software metrics and fault data. Usually not all software metrics in data repositories contribute equally to the occurrence of faults. Choosing the most important metrics (features) prior to the model training process is needed to improve the effectiveness of defect predictors. In this paper, we study 18 filter-based feature selection techniques and evaluate their effectiveness through a case study performed on 16 different software data sets. Among the 18 techniques, six of them are standard filter-based methods, while 11 of them are threshold-based feature selection (TBFS) techniques proposed by our research team recently. The last one is based on signal to noise ratio (S2N), which is a widely used concept in electrical and communication engineering, but which is rarely used in feature selection. The experimental results demonstrate that the TBFS techniques perform similarly to the standard techniques and the S2N technique shows significantly better performance than the other 17 approaches.**

***Index Terms*—threshold-based feature selection, filter-based feature ranking techniques, software defect prediction, software metrics, software quality, signal to noise ratio.**

I. Introduction

One effective method for improving the quality and reliability of a high-assurance software project/product is to detect and correct software defects (bugs) early enough during the software development process and prior to system deployment and operation, since for such high-assurance systems certain defects or failures can have disastrous consequences. Various techniques and approaches have been created for this purpose. Among them, software quality modeling is an effective and attractive approach. In software quality assurance, practitioners often use software metrics (attributes or features) gathered during the software development process and various data mining techniques to build classification models for predicting whether a given program module (instance or example) is in the fault-prone (*fp*) or not fault-prone (*nfp*) class [1]. Such a strategy allows practitioners to intelligently allocate project resources and focus more on the potentially problematic modules.

However, in the practice of software quality estimation, we find that not all collected software metrics are useful or make equally important contributions to classification results. Simply training a defect prediction model using the available set of software metrics without regard to the quality of the underlying software measurement data is not recommended. Selecting a subset of features that are most relevant to the class attribute is necessary and may result in better predictions.

In this paper, we investigate 18 filter-based feature ranking techniques to select subsets of features (metrics or predictors). Among the 18 techniques, six of them are commonly used, namely, chi-square (CS), information gain (IG), gain ratio (GR), symmetrical uncertainty (SU), and ReliefF (two types, RF and RFW) [2]. Eleven of them are threshold-based feature selection techniques (TBFS) proposed by our research team; a preliminary investigation of one of these 11 TBFS techniques has been reported in our recent work [3]. The remaining technique is based on signal to noise ratio (S2N), which is a widely used concept in electrical and communication engineering, but which has only been used in data mining research very recently.

Our proposed threshold-based feature selection (TBFS) techniques substantially extend the FAST algorithm proposed by Chen and Wasikowski [4]. FAST is based on the area under a ROC (Receiver Operating Characteristic) curve generated by moving the decision boundary of a single feature classifier with thresholds placed using an even-bin distribution. This means that they calculated a ROC curve by discretizing the distribution, while our technique is much more general than their work. Our technique does not require discretization, making it more precise and avoiding the determination of how wide the bins should be. Further, there are 11 different versions of TBFS which are based on 11 different classifier performance metrics for feature ranking. Another nice property of our technique is that it can be extended to incorporate additional metrics.

In order to evaluate the effectiveness of the 18 filter-based feature selection methods, we perform a case study on 16 software data sets. The case study data consists of software measurement and defect data from three real-world software projects, including four data sets from a very large high-assurance telecommunications software system (LLTS) [5], three data sets from NASA software project KC1 [6], and nine data sets from the Eclipse project [7].

Following feature selection, defect prediction models are constructed using the Naïve Bayes classifier with the training data consisting of the software metrics selected by the 18 different approaches. The empirical results demonstrate that our proposed TBFS techniques have similar performance to the six commonly used feature selection approaches. In addition, the S2N technique shows significantly better performance than the other 17 techniques (6 standard methods + 11 TBFS methods). Moreover, some techniques exhibit more stable performance than other techniques with respect to different data sets. One thing we would like to emphasize is that from a software engineering point of view, a practitioner appreciates working with a smaller set of metrics for defect prediction than analyzing a large number of metrics.

The remainder of the paper is organized as follows: Section II summarizes some key related work. Section III describes the 18 filter-based feature ranking techniques and other methodologies used in this study. Section IV presents an empirical case study, including

software measurement data, results and analysis. Section V draws the conclusion of the paper and provides some directions for future work.

II. Related Work

One of the challenging problems in the data mining process is high dimensionality of data, which not only requires extensive computation during the learning process but also may deteriorate learning results. Feature selection (or attribute selection) is an effective method for handling high dimensional data. In the selection process, the most important and relevant features will be selected to build classifiers. Feature selection techniques are broadly categorized into two groups: *wrapper*-based and *filter*-based. The wrapper-based approach involves training a learner during the feature selection process, while a filter-based approach uses the intrinsic characteristics of the data for feature selection and does not rely on training a learner. The primary advantage of the filter-based approach over the wrapper-based approach lies in its faster computation.

Hall and Holmes [8] investigated six attribute selection techniques that produce ranked lists of attributes and applied them to several data sets from the UCI machine learning repository. Ilczuk et al. [9] investigated the importance of feature selection in judging the qualification of patients for cardiac pacemaker implantation. Forman [10] studied a few number of filter-based feature ranking techniques in the context of text mining.

Although feature selection has been widely applied in various data mining problems, its application in software quality and reliability engineering has been rather limited. Rodríguez et al. [11] applied feature selection to five software engineering data sets using three filter-based models and two wrapper-based models. The authors stated that the reduced data sets maintained the prediction capability of the original data sets while using fewer attributes. Chen et al. [12] have studied feature selection using wrappers in the context of software cost/effort estimation. In their study, serval COCOMO-I and COCOMO-II data sets were used. They concluded that the reduced data set could improve the estimation and recommended feature selection in cost modeling, particularly when dealing with very small data sets. Pizzi et al. [13] described a stochastic metric selection method to identify the software metric subset that is most effective at predicting software module complexity. This classification method was empirically evaluated and validated against three benchmark approaches. The main advantage of this research, claimed by the authors, is that a project manager or software architect could utilize the predictions to identify highly complex modules for review and possible revision. In a recent study [14], we investigated feature selection techniques using the filter-based method for software defect prediction. It was concluded that the performances of the classification models either improved or were not affected when only 15% of the original features were used.

III. Methodology

A. Standard Filter-Based Feature Ranking Techniques

The procedure of feature ranking is to score each feature according to a particular method, allowing the selection of the best features. The six commonly used filter-based feature ranking techniques examined in this work include [2]: chi-square (CS), information gain (IG), gain ratio (GR), two types of ReliefF (RF and RFW), and symmetrical uncertainty (SU).

The chi-square (χ^2) statistic is used to evaluate the distribution of the class as it relates to the values of the target feature. The values of the features must be discretized into a number of intervals before performing the test. The χ^2 statistic is defined as follows:

$$\chi^2 = \sum_{i=1}^{I} \sum_{j=1}^{n_c} \frac{(O_{i,j} - E_{i,j})^2}{E_{i,j}}$$

where I is the number of different values (or intervals) of the feature, n_c is the number of classes (n_c=2 for a binary classification problem), $O_{i,j}$ and $E_{i,j}$ are the observed number and the expected number of instances corresponding to value (or interval) i and class j. The larger the χ^2 statistic, the more likely it is that the distribution of values and classes are dependent; that is, the feature is relevant to the class.

Information gain, gain ratio, and symmetrical uncertainty are measures commonly used in the field of information theory [2]. Information gain (IG) is the information provided about the target class attribute Y, given the value of another attribute X. IG measures the decrease of the weighted average impurity of the partitions compared to the impurity of the complete set of data. IG tends to prefer attributes with a larger number of possible values. If one attribute has a larger number of values, it will appear to gain more information than those with fewer values, even if it is actually no more informative. One strategy to solve this problem is to use the gain ratio (GR), which penalizes multiple-valued attributes. Symmetrical uncertainty (SU) is another way to overcome the problem of IG's bias toward attributes with more values, doing so by dividing by the sum of the entropies of X and Y.

Relief is an instance-based feature ranking technique introduced by Kira and Rendell [15]. It measures the importance of features by considering how much their values change when comparing a randomly chosen instance with its nearest hit (an instance from the same class) and its nearest miss (one from a different class). ReliefF is an extension of the Relief algorithm that can handle noise and multiclass data sets, and is implemented in the WEKA tool [2]. When the `WeightByDistance` (weight nearest neighbors by their distance) parameter is set as default (false), the algorithm is referred to as RF; when the parameter is set to 'true', the algorithm is referred to as RFW.

Algorithm 1: Threshold-Based Feature Selection

input :
1. Data set D with features $F^j, j = 1, \ldots, m$;
2. Each instance $\mathbf{x} \in D$ is assigned to one of two classes $c(\mathbf{x}) \in \{P, N\}$ where $P = fp$ and $N = nfp$;
3. The value of attribute F^j for instance $\mathbf{x}$ is denoted $F^j(\mathbf{x})$;
4. Metric $\omega \in$ {FM, OR, PO, PR, GI, MI, KS, DEV, GM, AUC, PRC}.

output: Ranking $\mathbb{R} = \{r^1, \ldots, r^m\}$ where r^j represents the rank for attribute F^j, i.e., the r^j-th most significant attribute as determined by metric ω.

for $F^j, j = 1, \ldots, m$ **do**
- Normalize $F^j \mapsto \widehat{F}^j = \frac{F^j - \min(F^j)}{\max(F^j) - \min(F^j)}$;
- Calculate metric ω using attribute $\widehat{F}^j$ and class attribute $\{c(\mathbf{x}) | \mathbf{x} \in D\}$, $\omega(\widehat{F}^j)$; (The detailed formula of each metric ω is provided in Sections III-B1 through III-B11.)

Create attribute ranking $\mathbb{R}$ using $\omega(\widehat{F}^j)$ $\forall j$

B. Threshold-Based Feature Selection

The threshold-based feature selection (TBFS) technique was proposed by our research team and implemented within WEKA [2]. The procedure is shown in Algorithm 1. Each independent attribute works individually with the class attribute, and that two-attribute data set is evaluated using different performance metrics. More specifically,

the TBFS procedure includes two steps: (1) normalizing the attribute values so that they fall between 0 and 1; and (2) treating those values as the posterior probabilities from which to calculate performance metrics. Note that no classifiers were built during the feature selection process.

Analogous to the procedure for calculating rates in a classification setting with a posterior probability, the true positive (TPR), true negative (TNR), false positive (FPR), and false negative (FNR) rates can be calculated at each threshold $t \in [0,1]$ relative to the normalized attribute $\widehat{F}^j$. Precision $PRE(t)$ is defined as the number of positive examples with $\widehat{F}^j > t$ divided by the total number of examples with $\widehat{F}^j > t$. The feature rankers we propose utilize these five rates as described below. The value is computed in both directions: first treating instances above the threshold (t) as positive and below as negative, then treating instances above the threshold as negative and below as positive. The better result is used. Each of the 11 metrics are calculated for each attribute individually, and attributes with higher values for FM, GM, PR, PO, AUC, PRC, MI, KS and OR and lower values for GI and DEV are determined to better predict the class attribute. In this manner, the attributes can be ranked from most to least predictive based on each of the 11 metrics.

1) F-Measure: The F-measure (FM) is derived from *recall* (or true positive rate) and *precision*.

$$\text{FM} = \max_{t\in[0,1]} \frac{2 \times TPR(t) \times PRE(t)}{TPR(t) + PRE(t)}.$$

Recall and precision are calculated at each point along the normalized attribute range of 0 to 1. The maximum F-measure obtained by each attribute represents how strongly that particular attribute relates to the class, according to the F-measure.

2) Odds Ratio: The odds ratio (OR) is defined as:

$$\begin{aligned}\text{OR} &= \max_{t\in[0,1]} \frac{TPR(t)(1-FPR(t))}{(1-TPR(t))FPR(t)} \\ &= \max_{t\in[0,1]} \left(\frac{TPR(t)}{FPR(t)}\right)\left(\frac{TNR(t)}{FNR(t)}\right)\end{aligned}$$

OR is the maximum value of the ratio of the product of correct to incorrect predictions.

3) Power: Power (PO) is defined as:

$$\begin{aligned}\text{PO} &= \max_{t\in[0,1]} \left((1-FPR(t))^k - (1-TPR(t))^k\right) \\ &= \max_{t\in[0,1]} \left((TNR(t))^k - (FNR(t))^k\right)\end{aligned}$$

for some integer $k \geq 1$. Note that if $k = 1$, Power is equivalent to KS (described in Section III-B7). In this work, we use $k = 5$ as in Forman [10].

4) Probability Ratio: The probability ratio (PR) is defined as:

$$\text{PR} = \max_{t\in[0,1]} \frac{TPR(t)}{FPR(t)}$$

5) Gini Index: The Gini index (GI) was first introduced by Brieman et al. [16] within the CART algorithm. For a given threshold t, let $S_t = \{\mathbf{x} \mid \widehat{F}^j(\mathbf{x}) > t\}$ and $\bar{S}_t = \{\mathbf{x} \mid \widehat{F}^j(\mathbf{x}) \leq t\}$. The Gini index is calculated as:

$$\begin{aligned}\text{GI} &= \min_{t\in[0,1]} \left[1 - \left(P^2(TP(t) \mid S_t) + P^2(FP(t) \mid S_t)\right)\right] \\ &\quad + \left[1 - \left(P^2(TN(t) \mid \bar{S}_t) + P^2(FN(t) \mid \bar{S}_t)\right)\right] \\ &= \min_{t\in[0,1]} \left[2PRE(t)(1-PRE(t)) + 2NPV(t)(1-NPV(t))\right]\end{aligned}$$

where $TP(t)$ is the number of true positives given threshold t (and similarly for $TN(t)$, $FP(t)$ and $FN(t)$). NPV or *negative predictive value* represents the percentage of examples predicted to be negative that are actually negative and is very similar to the precision — in fact, it is often thought of as the precision of instances predicted to be in the negative class. The Gini index for the attribute is then the minimum Gini index at all decision thresholds $t \in [0,1]$.

6) Mutual Information: Let $c(\mathbf{x}) \in \{P, N\}$ denote the actual class of instance $\mathbf{x}$, and let $\hat{c}^t(\mathbf{x})$ denote the predicted class based on the value of the attribute F^j and a given threshold t. The mutual information (MI) computes the criterion with respect to the number of times a feature value and a class co-occur, the feature value occurs without the class, and the class occurs without the feature value. The MI metric is defined as:

$$\text{MI} = \max_{t\in[0,1]} \sum_{\hat{c}^t\in\{P,N\}} \sum_{c\in\{P,N\}} p(\hat{c}^t, c) \log \frac{p(\hat{c}^t, c)}{p(\hat{c}^t)p(c)}$$

where

$$\begin{aligned}&p(\hat{c}^t = \alpha, c = \beta) = \frac{\mid \{\mathbf{x} \mid (\hat{c}^t(\mathbf{x}) = \alpha) \cap (c(\mathbf{x}) = \beta)\} \mid}{\mid P \mid + \mid N \mid}, \\ &p(\hat{c}^t = \alpha) = \frac{\mid \{\mathbf{x} \mid \hat{c}^t(\mathbf{x}) = \alpha\} \mid}{\mid P \mid + \mid N \mid}, \\ &p(c = \alpha) = \frac{\mid \{\mathbf{x} \mid c(\mathbf{x}) = \alpha\} \mid}{\mid P \mid + \mid N \mid}, \\ &\alpha, \beta \in \{P, N\}.\end{aligned}$$

7) Kolmogorov-Smirnov Statistic: The Kolmogorov-Smirnov Statistic (KS) measures the maximum difference between the cumulative distribution functions of examples in each class based on the normalized attribute $\widehat{F}^j$. The distribution function $F_c(t)$ for a class c is estimated by the proportion of examples $\mathbf{x}$ from class c with $\widehat{F}^j(\mathbf{x}) \leq t$, $t \in [0,1]$. In a two class setting with $c \in \{N, P\}$, KS is computed as

$$\text{KS} = \max_{t\in[0,1]} |F_P(t) - F_N(t)|.$$

The larger the KS value, the better the attribute is able to separate the two classes, and hence the more significant the attribute is. The range of KS is between 0 and 1.

8) Deviance: The deviance (DEV) is the minimum residual sum of squares based on a threshold t. It measures the sum of the squared errors from the mean class given a partitioning of the space based on the threshold t. As it represents to total error found in the partitioning, lower values are preferred.

9) Geometric Mean: The geometric mean (GM) is the square root of the product of the true positive rate and true negative rate. GM ranges from 0 to 1, and an attribute that is perfectly correlated to the class provides a value of 1. GM is a useful performance measure since it is inclined to maximize the true positive rate and the true negative rate while keeping them relatively balanced. GM is calculated at each value of the normalized attribute range, and the maximum value of GM is used as a measure of attribute strength.

10) Area Under the ROC Curve: Receiver Operating Characteristic, or ROC, curves graph true positive rate on the y-axis versus the false positive rate on the x-axis. The resulting curve illustrates the trade-off between true positive rate and false positive rate. In this study, ROC curves are generated by varying the decision threshold t used to transform the normalized attribute values into a predicted class. In other words, the true positive and false positive rates are calculated as the threshold for the normalized attribute varies from 0 to 1. The area under the ROC curve (AUC) is used to provide a

single numerical metric for comparing the predictive power of each attribute. This definition is different than the one used by Chen and Wasikowski [4], which consider only a small subset of the possible threshold values when calculating the true positive and false positive rates.

11) Area Under the Precision-Recall Curve: The area under the precision-recall curve (PRC) is a single-value measure that originated from the area of information retrieval. A precision-recall curve is generated by varying the decision threshold t from 0 to 1 and plotting the recall (y-axis) and precision (x-axis) at each point in a similar manner to the ROC curve. The area under the PRC ranges from 0 to 1, and an attribute with more predictive power results in an area under the PRC closer to 1.

C. Signal to Noise Ratio Technique

Signal to noise ratio (S2N) [17] is a simple univariate ranking technique which defines how well a feature discriminates two classes in a two class problem. S2N, for a given feature, separates the means of the two classes relative to the sum of their standard deviation. The equation to calculate S2N is

$$\text{S2N} = \frac{(\mu_P - \mu_N)}{\sigma_P + \sigma_N}$$

where μ_P and μ_N are the mean values of a particular attribute for the samples from class P and class N, and σ_P and σ_N are the corresponding standard deviations.

D. Classification Algorithm

The software defect prediction models are built using the naïve Bayes (NB) [2] algorithm. This learner was selected for two key reasons: (1) it does not have a built-in feature selection capability, and (2) it is commonly used in both the software engineering and data mining domains. Prior research has shown that naïve Bayes classifiers often perform well, even on real-world data where the variables are related [18]. The WEKA data mining tool [2] is used to implement the classifier and the default parameter settings are adopted.

E. Performance Evaluation

The Area Under the ROC (Receiver Operating Characteristic) curve, abbreviated as AUC, is used for evaluating the defect prediction models in this study. The AUC metric is illustrated earlier in the paper. It is worthwhile to mention that AUC is commonly used to evaluate software defect prediction models [1]. An ROC curve illustrates the classifier's performance across all decision thresholds, i.e., a value between 0 and 1 that theoretically separates the *fp* and *nfp* modules. The AUC values range from 0 to 1, where a perfect classifier provides an AUC value of 1 [2].

IV. Case Study

A. Software Measurement Data

Experiments conducted in this study used software metrics and defect data collected from three real-world software projects, including a very large high-assurance telecommunications software system (denoted as LLTS) [5], NASA software project KC1 [6], and the Eclipse project [7].

LLTS consists of 42 software metrics, including 24 product metrics, 14 process metrics, and four execution metrics. The dependent variable is the class of the program module. A module with one or more faults is considered *fp*, and *nfp* otherwise. The LLTS software system consists of four successive releases labeled SP1, SP2, SP3, and SP4, where each release is characterized by the same number and type of software metrics, but has a different number of instances (program modules). The SP1, SP2, SP3, and SP4 data sets consist of 3649, 3981, 3541, and 3978 program modules, respectively.

TABLE I
Software Data Set Characteristics

	Data	#Attri.	#Inst.	#fp	%fp	#nfp	%nfp
LLTS	SP1	42	3649	229	6%	3420	94%
	SP2	42	3981	189	5%	3792	95%
	SP3	42	3541	47	1%	3494	99%
	SP4	42	3978	92	2%	3886	98%
NASA	KC1-5	62	145	36	25%	109	75%
	KC1-10	62	145	21	14%	124	86%
	KC1-20	62	145	10	7%	135	93%
Eclipse	E2.0-10	208	377	23	6%	354	94%
	E2.0-5	208	377	52	14%	325	86%
	E2.0-3	208	377	101	27%	276	73%
	E2.1-5	208	434	34	8%	400	92%
	E2.1-4	208	434	50	12%	384	88%
	E2.1-2	208	434	125	29%	309	71%
	E3.0-10	208	661	41	6%	620	94%
	E3.0-5	208	661	98	15%	563	85%
	E3.0-3	208	661	157	24%	504	76%

The original NASA project KC1 [6] includes 145 instances, each containing 94 independent attributes. After removing 32 Halstead derived measures, we have 62 attributes left. We used three different thresholds to define defective instances, thereby obtaining three versions of the preprocessed KC1 data set. The thresholds are 20, 10, and 5, indicating instances with number of defects greater or equal to 20, 10, or 5 belong to the *fp* class, or the *nfp* class otherwise.

From the PROMISE data repository [7], we obtained the Eclipse defect counts and complexity metrics data set. The original data for the Eclipse packages consists of three releases denoted 2.0, 2.1, and 3.0, respectively. We chose three post-release defects thresholds (*thd*) to determine the defective instances for each release. Similar to the NASA KC1 data sets, a program module with thd or more post-release defects is labeled *fp*, while those with fewer than thd defects are labeled *nfp*. In our study, we use *thd* ϵ {10, 5, 3} for release 2.0 and 3.0, while we use *thd* ϵ {5, 4, 2} for release 2.1. All nine derived data sets contain 208 independent attributes. Releases 2.0, 2.1, and 3.0 contain 377, 434, and 661 instances, respectively. The sixteen data sets used in this work reflect software projects of different sizes with different proportions of *fp* and *nfp* modules. Table I lists the characteristics of the sixteen data sets utilized in this work.

B. Experiments

Before using a filter-based feature ranking technique, the practitioner must choose how many features to select. To our knowledge, no guidance is provided in related literature on the appropriate number of features to select. A recent study [19] recommended using $\lceil \log_2 n \rceil$ features (n is the total number of the independent attributes) to build Random Forests learners for binary classification for imbalanced data sets. Moreover, a preliminary investigation showed that $\lceil \log_2 n \rceil$ is also appropriate for various learners. Consequently, we still choose $\lceil \log_2 n \rceil$ attributes that have the highest scores. That is, for the four LLTS data sets, $\lceil \log_2 n \rceil = 6$, where $n = 42$; for the three NASA KC1 data sets $\lceil \log_2 n \rceil = 6$, where $n = 62$; and for the nine Eclipse data sets, $\lceil \log_2 n \rceil = 8$, where $n = 208$.

1) Results of the Feature Selection Techniques: Following the feature selection algorithms, the NB classification models are constructed with data sets containing only the selected attributes. The defect prediction models are evaluated with respect to the AUC performance metric.

TABLE II
CLASSIFICATION PERFORMANCE IN TERMS OF AUC

Filters	LLTS				NASA			Eclipse									Avg.
	SP1	SP2	SP3	SP4	KC1-5	KC1-10	KC1-20	E2.0-10	E2.0-5	E2.0-3	E2.1-5	E2.1-4	E2.1-2	E3.0-10	E3.0-5	E3.0-3	
CS	0.7846	0.8108	0.8184	0.7696	0.7484	0.7513	0.8525	0.7904	0.8421	0.7963	0.8419	0.8226	0.7536	0.8742	0.8866	0.8130	0.8098
GR	0.7346	0.7613	0.7808	0.7519	0.7489	**0.7729**	0.8669	0.8074	0.8078	0.7458	0.7919	0.7917	0.7614	0.8463	0.8785	0.7974	0.7903
IG	0.7831	0.8081	0.8118	0.7795	0.7438	0.7546	0.8569	0.8070	0.8562	0.8002	0.8547	0.8281	0.7542	0.8963	0.8851	0.8122	0.8145
RF	0.7879	0.8053	**0.8305**	0.7731	**0.7990**	0.7585	0.8296	0.8455	0.8617	0.7857	0.8022	0.7688	0.7993	0.8044	0.8481	0.7789	0.8049
RFW	0.7882	0.8081	0.8190	0.7735	0.7832	0.7639	**0.8987**	0.8107	0.8607	**0.8107**	0.8188	0.7302	0.7966	0.8101	0.8732	0.8024	0.8093
SU	0.7865	0.7729	0.7882	0.7592	0.7468	0.7719	0.8532	0.8158	0.8464	0.7940	0.8269	0.8170	0.7551	0.8540	0.8817	0.8072	0.8048
FM	0.7822	0.8074	0.8176	0.7731	0.7479	0.7119	0.8519	0.8463	0.8588	0.7868	0.8491	0.8131	0.7619	0.8936	0.8863	0.8129	0.8125
OR	0.7405	0.8060	0.7181	0.7558	0.7364	0.7700	0.8541	0.8147	0.8567	0.8001	0.8225	0.8109	0.7554	0.8606	0.8830	0.8092	0.7996
PO	0.7891	0.8071	0.8141	0.8023	0.7358	0.7627	0.8405	0.8398	0.8464	0.7985	0.8126	0.8050	0.7522	0.8966	0.8885	0.8163	0.8130
PR	0.7345	0.7963	0.7179	0.7605	0.7667	0.7641	0.8378	0.8116	0.8081	0.7963	0.7543	0.7841	0.7034	0.8408	0.8810	0.8107	0.7855
GI	0.7341	0.7982	0.7678	0.6997	0.7671	0.7645	0.8398	0.8116	0.8079	0.7963	0.7539	0.7858	0.7056	0.8439	0.8813	0.8108	0.7855
MI	0.7739	0.8010	0.8119	0.7788	0.7602	0.7413	0.8556	0.8335	0.8558	0.7897	0.8531	0.8234	0.7542	0.8715	0.8845	0.8124	0.8125
KS	0.7723	0.7750	0.8125	0.7588	0.7754	0.7169	0.8574	0.8243	0.8533	0.7914	0.8510	0.8238	0.7594	0.8713	0.8862	0.8109	0.8087
DEV	0.7821	0.8099	0.8163	0.7874	0.7559	0.7480	0.8504	0.8350	0.8558	0.7866	0.8563	0.8240	0.7563	0.8986	0.8856	0.8140	0.8164
GM	0.7716	0.7740	0.8165	0.7586	0.7665	0.7137	0.8588	0.8168	0.8423	0.7914	0.8464	0.8187	0.7627	0.8622	0.8858	0.8123	0.8061
AUC	0.7685	0.8072	0.7947	0.7683	0.7682	0.7066	0.8404	**0.8763**	0.8588	0.7932	0.8507	0.8163	0.7544	0.8988	0.8847	0.8101	0.8123
PRC	0.7885	0.8131	0.8120	0.7953	0.7147	0.7349	0.8300	0.8405	0.8474	0.7917	0.8506	0.8245	0.7569	0.8936	0.8852	0.8106	0.8118
S2N	**0.7995**	**0.8142**	0.8067	**0.8130**	0.7847	0.7508	0.8646	0.8623	**0.8767**	0.8003	**0.8681**	**0.8560**	**0.8063**	**0.9175**	**0.9099**	**0.8759**	**0.8379**

TABLE III
ONE-WAY ANOVA

Source	Sum Sq.	d.f.	Mean Sq.	F	p-value
Techniques	0.4039	17	0.0238	10.38	0
Error	6.5513	2862	0.0023		
Total	6.9552	2879			

The classifier performance results are presented in Table II. In our experiments, ten runs of five-fold cross-validation were performed for model training. The values presented in the tables represent the average AUC for every classification model constructed over the ten runs of five-fold cross-validation. All the results of 18 feature selection techniques and over 16 different software data sets are reported. The best feature selection technique in terms of classification performance (AUC) for each data set (column) is highlighted in **bold**. We also summarize the average performance (last column of the table) for each feature selection technique across the 16 data sets. The results demonstrate that S2N outperformed the other feature selection techniques for 10 out of 16 data sets and also on average. For the other six data sets the best feature selection techniques are scattered at GR(1), RF(2), RFW(2) and AUC(1), where the value in parenthesis represents the number of best cases.

We also performed a one-way ANalysis Of VAriance (ANOVA) F-test on the classification performance for each technique across all the data sets to examine the significance level of the performance differences. The underlying assumptions of ANOVA were tested and validated prior to statistical analysis. The main factor of our ANOVA experiment is the 18 feature ranking techniques. The null hypothesis for the ANOVA test is that all the group population means are the same, while the alternate hypothesis is that at least one pair of means is different. Table III shows the ANOVA results. The p-value is less than the typical cutoff of 0.05, implying that for the main factor, the alternate hypothesis is accepted, namely, at least two group means are significantly different from each other. We continued our statistical validation by performing a multiple comparison test on the main factor with Tukey's Honestly Significant Difference (HSD) criterion. Note that for both ANOVA and multiple comparison tests, the significance level was set to 0.05.

The multiple comparison results, as shown in Figure 1, display a graph with each group mean represented by a symbol ($\circ$) and the 95% confidence interval as a line around the symbol. Two means

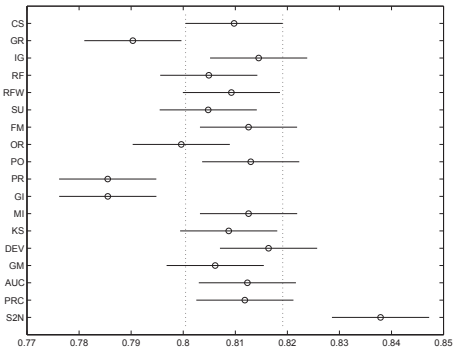


Fig. 1. Multiple Comparison

are significantly different if their intervals are disjoint, and are not significantly different if their intervals overlap. Based on the multiple comparison results, we can conclude the following points:

- For the six standard filter-based feature selection techniques, GR performed worst, IG performed best and the other four methods sit in between.
- For the 11 proposed threshold-based feature selection techniques, PR and GI performed poorly, DEV, PO, FM, MI, AUC, and PRC performed relatively better than other techniques, and OR, KS, and GM performed moderately.
- Overall, the proposed threshold-based feature selection techniques had fairly similar performance to the six standard filter-based feature selection methods. However, the S2N technique showed significantly better performance than the other 17 techniques.
- Moreover, some techniques demonstrated relatively stable performance with respect to different data sets such as MI, KS, S2N and SU, while other techniques, such as RF and RFW, showed more fluctuational performance with respect to different data sets.

2) Discussion on Selected Software Metrics: From a software engineering point of view, a discussion on which software metrics were selected is warranted. Due to paper size limitations, we only present the results on LLTS SP1 (see Table IV). Table IV presents the six selected software metrics for each of the top three feature selection techniques (S2N, PO and PRC). It is found that the selected

TABLE IV
SOFTWARE METRICS SELECTED FOR SP1

Top 3 Filters	Ranked attributes ID 1	2	3	4	5	6
S2N	19	31	32	36	27	21
PO	13	11	37	39	23	17
PRC	16	29	36	42	32	11

TABLE V
SOFTWARE METRICS FOR LLTS DATA SETS

ID	Symbol	Description
11*	UNQ_DES	# of different designers making changes.
13	LO_UPD	# of updates to this module by designers who had between 11 and 20 total updates in entire company career.
16	CALUNQ	# of distinct procedure calls to others.
17	CAL2	# of second and following calls to others.
19	CNDSPNMX	Maximum span of branches of conditional arcs.
21	FILINCUQ	# of distinct include files.
23	LOC	# of lines of code.
27	NDSENT	# of entry nodes.
29	NDSPND	# of pending nodes (i.e., dead code segments).
31	LGPATH	Base 2 logarithm of the number of independent paths.
32*	STMCTL	# of control statements.
36*	VARSPNSM	Total span of variables.
37	VARSPNMX	Maximum span of variables.
39	VARUSD2	# of second and following uses of variables. VARUSD2=VARUSD-VARUSDUQ where VARUSD is the total # of variable uses.
42	TANCPU	Execution time (microseconds) of an average transaction on a tandem system.

* the metrics selected by two of the top three methods (S2N, PO and PRC)

attributes were spread over 15 different software metrics, and there were very few overlaps of the selected metrics between the top three methods, only three metrics {11, 32, 36} selected twice. Table V lists more detailed information of the 15 metrics.

Our recent work [20] has shown that classification models built on smaller subsets of attributes via the six commonly used filter-based feature selection techniques had similar or better performances than those built with a complete set of attributes. Thus, we did not present the results for full data sets in this paper.

V. CONCLUSION

In the software quality modeling process, one problem often encountered by software practitioners is the presence of excessive metrics in a training data set. In this study, we presented 18 filter-based feature ranking techniques to choose the most important software metrics. Among the 18 techniques, six are standard filter-based techniques, while 11 are the threshold-based techniques we proposed, and the remaining one is the signal to noise ratio (S2N) technique rarely used in feature selection. We used the naïve Bayes classifier to build classification models with data sets containing only the selected attributes. The experiments were performed on 16 different data sets obtained from three types of software projects. The key conclusions include: (1) our proposed threshold-based feature selection techniques performed similarly to the standard feature selection techniques; (2) the S2N technique showed significantly better performance than the other 17 techniques; and (3) some techniques (such as MI, KS, S2N, and SU) demonstrated more stable performance than other techniques (such as RF and RFW) with respect to different data sets. Future work will include more case studies with software measurement data sets of other software systems. In addition, evaluation of the classification models using different learners such as multilayer perceptron, support vector machine, and *k*-nearest-neighbors will be examined.

REFERENCES

[1] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking classification models for software defect prediction: A proposed framework and novel findings," *IEEE Transactions on Software Engineering*, vol. 34, no. 4, pp. 485–496, July-August 2008.

[2] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd ed. Morgan Kaufmann, 2005.

[3] T. M. Khoshgoftaar and K. Gao, "A novel software metric selection technique using the area under roc curves," in *Proceedings of the 22nd International Conference on Software Engineering and Knowledge Engineering*, San Francisco, CA, July 1-3 2010, pp. 203–208.

[4] X.-w. Chen and M. Wasikowski, "Fast: a roc-based feature selection metric for small samples and imbalanced data classification problems," in *KDD '08: Proc. 14th ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining*. New York, NY, USA: ACM, 2008, pp. 124–132.

[5] T. M. Khoshgoftaar, L. A. Bullard, and K. Gao, "Attribute selection using rough sets in software quality classification," *International Journal of Reliability, Quality and Safty Engineering*, vol. 16, no. 1, pp. 73–89, 2009.

[6] A. G. Koru, D. Zhang, K. E. Emam, and H. Liu, "An investigation into the functional form of the size-defect relationship for software modules," *IEEE Transactions on Software Engineering*, vol. 35, no. 2, pp. 293–304, 2009.

[7] T. Zimmermann, R. Premraj, and A. Zeller, "Predicting defects for eclipse," in *Proceedings of the 29th International Conference on Software Engineering Workshops*. Washington, DC, USA: IEEE Computer Society, 2007, p. 76.

[8] M. A. Hall and G. Holmes, "Benchmarking attribute selection techniques for discrete class data mining," *IEEE Transaction on Knowledge and Data Engineering*, vol. 15, no. 6, pp. 1437–1447, Nov/Dec 2003.

[9] G. Ilczuk, R. Mlynarski, W. Kargul, and A. Wakulicz-Deja, "New feature selection methods for qualification of the patients for cardiac pacemaker implantation," *Computers in Cardiology*, vol. 34, no. 2-3, pp. 423–426, 2007.

[10] G. Forman, "An extensive empirical study of feature selection metrics for text classification," *Journal of Machine Learning Research*, vol. 3, pp. 1289–1305, March 2003.

[11] D. Rodriguez, R. Ruiz, J. Cuadrado-Gallego, and J. Aguilar-Ruiz, "Detecting fault modules applying feature selection to classifiers," in *Proceedings of 8th IEEE International Conference on Information Reuse and Integration*, Las Vegas, Nevada, August 13-15 2007, pp. 667–672.

[12] Z. Chen, T. Menzies, D. Port, and B. Boehm, "Finding the right data for software cost modeling," *IEEE Software*, vol. 22, no. 6, pp. 38–46, November 2005.

[13] N. J. Pizzi, A. B. Demko, and W. Pedrycz, "The analysis of software complexity using stochastic metric selection," *Journal of Pattern Recognition Research*, vol. 1, pp. 19–31, 2011.

[14] K. Gao, T. M. Khoshgoftaar, H. Wang, and N. Seliya, "Choosing software metrics for defect prediction: an investigation on feature selection techniques," *Software: Practice and Experience. Special Issue: Practical Aspects of Search-Based Software Engineering*, vol. 41, no. 5, pp. 579–606, April 2011.

[15] K. Kira and L. A. Rendell, "A practical approach to feature selection," in *Proceedings of 9th International Workshop on Machine Learning*, 1992, pp. 249–256.

[16] L. Breiman, J. Friedman, R. Olshen, and C. Stone, *Classification and Regression Trees*. Boca Raton, FL: Chapman and Hall/CRC Press, 1984.

[17] L. Goh, Q. Song, and N. Kasabov, "A novel feature selection method to improve classification of gene expression data," in *Proceedings of the Second Conference on Asia-Pacific Bioinformatics*, Dunedin, New Zealand, 2004, pp. 161–166.

[18] P. Domingos and M. Pazzani, "On the optimality of the simple bayesian classifier under zero-one loss," *Machine Learning*, vol. 29, no. 2-3, pp. 103–130, 1997.

[19] T. M. Khoshgoftaar, M. Golawala, and J. Van Hulse, "An empirical study of learning from imbalanced data using random forest," in *Proceedings of the 19th IEEE International Conference on Tools with Artificial Intelligence*, vol. 2, Washington, DC, USA, 2007, pp. 310–317.

[20] H. Wang, T. M. Khoshgoftaar, K. Gao, and N. Seliya, "Mining data from multiple software development projects," in *Proceedings of 2009 IEEE Intl. Conf. on Data Mining Workshops*, Miami, FL, 2009, pp. 551–557.

Criteria of Human Software Evaluation: Feature Selection Approach

Marek Z. Reformat
Electrical and Computer Engineering
University of Alberta
Edmonton, AB, Canada
Marek.Reformat@ualberta.ca

Sonal Patel
Electrical and Computer Engineering
University of Alberta
Edmonton, AB, Canada
sonal1@ualberta.ca

***Abstract*—Software maintenance includes activities related to keeping already developed software up to the needs and wishes of its users. In order to accomplish that there is a need to understand how this software works. This task can be simplified when the software is built using simple, easy to use and maintain components.**

In order to learn more about software artifacts programmers estimate their quality. They inspect the code, and based on their experience they offer their judgment and provide a score in a specific scale. An important aspect of that process is to understand how programmers perform such evolution. What software measures are subconsciously used during the evaluation? What kinds of measures are used to evaluate different software quality attributes?

In order to answer some of those questions we present a procedure suitable for estimating importance of software metrics and their subconscious utilization by programmers. The proposed approach is based on feature selection techniques.

Keywords: software maintenance, usability, and complexity; feature selection techniques; software evaluation process; software metric-based representation of programmers, tag-clouds

I. INTRODUCTION

Among all phases of a software life cycle the maintenance phase seems the most often forgotten. Yet at the same time it determines a long-term success of a software system. The activities related to maintenance depend on the level of understanding of software artifacts that constitute the software. This, on the other hand, depends on quality of artifacts – how they are design and implemented.

Quality of software artifacts is the subject to many research activities. Many aspects of that quality – especially reliability and defectiveness – are topics of numerous publications [1-4]. An important facet of defining and identifying software quality is related to a process of human evaluation of software objects. This process raises a number of questions: What does it mean for a programmer that a piece of code is of good quality, i.e., that it is usable, simple and easy to understand, or maintainable? Is a programmer consistent with his/her evaluations across multiple artifacts? Can we see some patterns among programmers – do they recognize the same software measures as important?

In order to identify important metrics for a programmer during an evaluation process, we propose a simple feature selection-based approach. The approach is suitable for finding important software measures that "describe" and "represent" an individual preforming a software object evaluation process. The proposed process uses a number of feature selection techniques, i.e., techniques that identify the most suitable software data features for representing software artifacts. The obtained results are combined together in order to find a single view characterizing a programmer performing evaluation. We also use tag-clouds [6] for visualization purposes.

We illustrate application of the proposed approach on a software engineering data. The data represents 366 software object described with 64 software metrics. Three programmers examined each object, and evaluated it from the point of view of complexity, usability and maintainability.

II. FEATURE SELECTION METHODS

Feature selection also known as attribute selection is an important pre-processing step in most data mining and machine learning problems. Apart from improving the quality of the machine learning data set, the main goal of a feature selection process is to look for a subset of features that is equivalent to the set of all features for a specific task [7].

There are two categories of feature selection methods: *wrappers* and *filters*. Wrappers are algorithms that use feedback from a learning algorithm in order to determine which feature(s) are needed in order to build a classification model. Filters are algorithms in which a feature subset is selected without any involvement of a learning algorithm (classifier). An additional categorization leads to feature selection techniques and feature ranking techniques. Feature selection methods select subsets of attributes that have good predictive power when used together. Feature ranking techniques ranks the attributes according to their individual predictive power [7].

A. Software for feature selection

There are several data analysis software systems already equipped with feature selection algorithms, such as MATLAB, SciLab, NumPy, and the R language. Apart from these, there are software systems that have been customized specifically for feature-selection tasks: Weka, RapirMiner, and Orange, to name just a few [8].

In Weka [9], feature selection can be performed in three different ways: 1) using selection methods directly; 2) using a meta classifier; or 3) using a filter. In the paper, the attribute selection has been done using two types of techniques: *SubSet evaluators*, and *Attribute (Ranking) evaluators*.

B. SubSet Evaluators

In the type of *SubSet Evalutors* we have applied three different methods.

ConsistencySubsetEval: The value of a subset of attributes is assessed by the level of consistency in the class values when the training instances are projected onto the subset of attributes. The common practice is to use it in conjunction with a Random or Exhaustive search which looks for the smallest subset with consistency equal to that of the full set of attributes [10].

CfsSubsetEval: A subset of features is evaluated by taking into consideration the individual predictive capability of each feature along with the degree of redundancy between them. Subsets of features which are highly correlated with the class while having low inter-correlation are preferred.

FilteredSubsetEval: A method for running a random subset evaluator on data that has been passed through an arbitrary filter. The structure of the filter is entirely based on the training data.

Those methods are combined with different serach techniques. A list of all applied combinations is included in Table I.

TABLE I. SUBSET EVALUATORS

Subset Evaluator:	Search Method:
ConsistencySubsetEval	BestFirst
CfsSubsetEval	GenericSearch
ConsistencySubsetEval	GenericSearch
FilteredSubsetEval	GreedyStepwise
ConsistencySubsetEval	LinearForwardSelection
CfsSubsetEval	RankSearch
CfsSubsetEval	ScatterSearchV1
CfsSubsetEval	SubsetSizeForwardSelection

C. Attribute Evaluators (Ranking Methods)

Weka provides a number of Attribute Evaluators. Among them we have used the following ones.

FilteredAttributeEval: Similar to FilteredSubsetEval.

GainRatioAttributeEval: Assesses the value of an attribute by measuring the gain ratio with respect to the class.

ChiSquaredAttributeEval: Assesses the importance of an attribute by calculating the value of the chi-squared statistic with respect to the class.

InfoGainAttributeEval: Assesses the importance of an attribute by measuring the information gain with respect to the class.

OneRAttributeEval: Assesses the value of an attribute by using the OneR classifier.

ReliefFAttributeEval: Assesses the value of an attribute by recurrently sampling an instance, and taking into consideration the value of the given attribute for the nearest instance of the same and different class.

SVMAttributeEval: An SVM Classifier is used to assessed the value of an attribute. Attributes are ranked by the square of the weight assigned by the SVM. For multiclass problems, attribute selection is performed by ranking attributes for each class independently using a one-vs-all method, and then processing the top of each pile to determine a final ranking.

SymmetricalUncertAttributeEval: Assesses the value of an attribute by measuring the symmetrical uncertainty with respect to the class.

All of these methods are run with Ranker as a selection method.

III. MULTI-METHOD ALGORITHMS

The process of identifying the most important criteria used subconsciously by programmers evaluating quality of the software artifacts is replaced by selection of the most important features of data generated via the human evaluation process. The process embraces number of selection techniques (Section III) and a simple procedure of fusing the results obtained from the selection processes.

A. Fusing Results of SubSet Evaluators

Every time we run a *SubSet Evaluator* we obtain a subset of features that according to the method we ran is the most discriminative. This means that using multiple selection methods we obtain multiple subsets of "super" features.

There are a number of ways in which we could combine these subsets – where the simplest ones are to do a union, or perform an intersection. However, we would like to make the merge in more conscious manner, and take advantage of utilization of multiple selection methods. A very simple algorithm applied for selecting the most significant features based on multiple different *SubSet Evalators* is presented in Figure 1.

```
// initialization
create counter for each feature
initialize counters to 0

// feature evaluation
for each subset of features
  for each feature from the subset
      increase the counter of that feature by 1
  rof
rof
sort features based on the value of their counters

// selection process
calculate a threshold value
      = 0.75*no_SubSet_methods
select features with values above the threshold
normalization of counters of the selected features
```

Figure 1. **Algorithm SF_SubSetE**:
Selection of Features from SubSet Evaluators

The algorithm SF_SubSetE is applied after all eight *SubSet Evaluators* (Table I) are run on the data. The result of that is a

single subset of best features. The example of its utilization is shown in Section VI.

B. Fusing Results of Attribute Evaluators

As for the *SubSet Evaluators,* a similar algorithm has been constructed for identification of the best features obtained from *Attribute Evaluators*, Figure 2. Its application to the software data is also presented in Section VI.

```
// initialization
create counter for each feature
initialize counters to 0

// feature evaluation
for each ranking
  for each feature from the ranking
      increment the counter of that feature by a number
      representing the position of the feature in
      the ranking
  rof
rof
sort features based on the value of their counters

// selection process
calculate a threshold value
      = 0.25*no_Rank_methods*no_features
select features with values below the threshold
normalization of counters of the selected features
```

Figure 2. **Algorithm SF_AttrE**:
Selection of Features from Attribute Evaluators

IV. Data Description

An experiment has been performed to generate data required for illustration of the proposed approach to support quality analysis. In the experiment, objects of the system EvIdent [5] have been independently analyzed and ranked by three programmers based on three quality attributes: complexity, maintainability and usability. Independently, quantitative software measures of these objects have been compiled.

A. Exeprimantal Setup

EvIdent is a user-friendly, algorithm-rich, graphical environment for the detection, investigation, and visualization of novelty/change in a set of images as they evolve in time or frequency. It is written in Java and C++ and is based upon VIStA, an application-programming interface (API) developed at the National Research Council, Figure 3.

The VIStA API is written in Java and offers a generalized data model, an extensible algorithm framework, and a suite of graphical user interface constructs. Using the Java Native Interface, VIStA allows seamless integration of native C++ algorithms. Figure 3 shows the class model and relationships between the three main Java packages used in VIStA.

B. Data Set

All Java-based EvIdent/VIStA objects have been used in this study (C++-based objects have been excluded). For each of the 366 software objects, three programmers, called '#1', '#2' and '#3', were asked to independently rank objects' maintainability, complexity and usability in the scale from 1 to 5 (where 1 means VERY POOR, 2 means POOR, 3 means OKAY, 4 means GOOD, and 5 means VERY GOOD). The programmers determined maintainability, complexity and usability of objects based on their own judgment and experience.

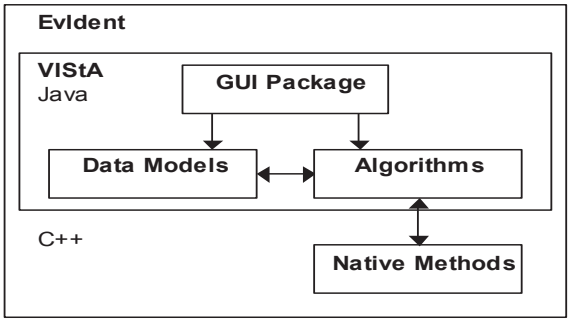


Figure 3. EvIdent software model with VIStA

At the same time, a set of 64 software metrics was calculated for each object. As the result, the collected data set consists of 366 data points represented by a set of 64 software metrics and three value assigned to each point by three programmers. A list of the extracted metrics can be found in Appendix.

V. Software Data Analysis

The available software data (Section IV) is processed using described feature selection methods (Section III). The whole procedure for identifying the best features is as follow:

StepA: the *SubSet Evaluators* are applied to the data; the results are processed using the algorithm SF_SubSetE

StepB: the *Attribute Evaluators* are applied to the data, and the algorithm SF_AttrE is used to identify the most significant features

StepC: the *correlation* between each pair: feature-quality attribute is calculated; a simple threshold (value of 0.7 is used here) is applied to select sets of most related features

StepD: the obtained three sets of features are fused using an arithmetic average of their scores; the features in the **upper quartile** (the one with the score above 0.75) are selected

The procedure is repeated for each programmer and each quality attribute. In total there are nine sets of best features selected for each combination of programmer-quality attribute. The following subsections illustrate details of the procedure for the **programmer_#1** and the quality attribute **complexity**.

A. SubSet Evalutor Results

In the first step – **StepA** – *SubSet Evaluators* are applied to the software data. The algorithm for fusing the results presented in Section IV has been applied next. The overall results are presented in Table II. As it can be seen in the table, the application of the **StepA** leads to the sequence of nine software metrics. We can say that these metrics represent the

programmer_#1 during evaluation of complexity of software objects. In other words, we can assume that these metrics are "important" indicators of complexity for the **programmer_#1**.

TABLE II. FEATURES OBTAINED USING SUBSET EVALUATORS FOR: PROGRAMMER_#1 – COMPLEXITY

LOC
WDC
CYCO
HLUR
MIC
TOK
MNL1
HLDF
HLVL

B. Attribute Evaluator Results

The *Attribute Evaluators* and the algorithm SF_AttrE have been also used to process the software data, **StepB**. The results are shown in Table III.

TABLE III. FEATURES OBTAINED USING ATTRIBUTE EVALUATORS FOR: PROGRAMMER_#1 – COMPLEXITY

HLPL
HLUR
HLOR
HLEF
HLON
WDC
HLDF
HLVL
HLVC
TOK
HLUN
DEC

These eleven metrics also represent the **programmer_#1** as the evaluator of **complexity** of software objects.

C. Correlation Results

The third set of processing methods is a simple calculation of correlations, **StepC**. It is performed for each pair software – complexity. The results are in Table IV.

TABLE IV. FEATURES OBTAINED USING CORRELATION FOR: PROGRAMMER_#1 – COMPLEXITY

HLUR
HLDF
HLVC
HLUN
CBO
FNOT
WMC1
CYCO

The set of metrics obtained using correlations is the third representation of the **programmer_#1**'s view at the **complexity** of objects.

D. Fusion of Results

The results obtained using **StepA**, **StepB** and **StepC** are fused together in order to achieve a single representation of the **programmer_#1**. As it has been indicated earlier, an arithmetic average has been used to combine scores of the metrics, and then all metrics with the combined score below 0.75 are rejected. The obtained set of the metrics characterizes the **programmer_#1**, Table V.

TABLE V. FEATURES REPRESENTING PROGRAMMER_#1 – COMPLEXITY

HLUR
LOC
WDC
HLDF
CYCO
HLVL
TOK
HLOR

The above described process has been applied to all available data, i.e., other quality attributes and other programmers. The final result – the metrics representing programmers as th evaluators of different quality attributes are included in Table VI.

TABLE VI. COMPLEXITY, MAINTAINABILITY AND USABILITY OBTAINED USING A FUSION OF SUBSET AND ATTRIBUTE EVALUATORS AND CORRELATION FDSFDS

programmer_#1	**programmer_#2**	**programmer_#3**
complexity		
HLUR	HLDF	LOC
LOC	ATOK	HLDF
WDC	CBO	WDC
HLDF	HLVC	CYCO
CYCO	HLVL	HLVL
HLVL	DEC	HLPL
TOK		
HLOR		
maintainability		
HLON	HLDF	HLVC
HLUR	RFO	DAC
LOC	DEC	DEC
HLVC	WDC	CBO
FNOT	DAC	REMM
	HLVL	HLUN
	LOC	
usability		
WMC2	CYCO	CBO
OPER	MEMB	HLPL
CYCO	OPER	FNOT
HLUR	LOC	DAC
WMC1	ADDM	MIC
LOC		LOC
		IMST

VI. ANALYSIS OF RESULTS

A. Single Quality Attribute Across Multiple Programmers

At first, we look at "the description" of all three programmers when they evaluated objects from the point of view of different quality attributes. Figure 4 shows sets of mutually overlapping features that represent the programmers when they evaluate complexity of objects. We could say that each of them "uses" Halstead program volume (HLVL). There are also features that are "shared" between pairs of

programmers. A single-metric-overlap exists between the **programmer_#1** and the **programmer_#2**, while a large overlap is between the **programmer_#1** and the **programmer_#3**. There is no overlapping between **programmer_#2** and **_#3**.

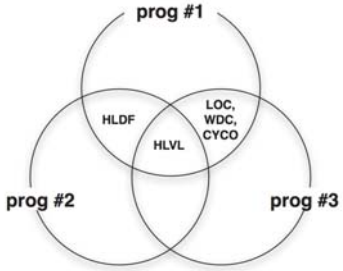


Figure 4. Complexity

The diagram in Figure 4 gives an interesting insight into a practice of describing, characterizing and evaluating programmers. We could hypothesize that **programmer_#1** and **_#3** could think in a similar way when they estimate complexity. Another interesting conclusion is that lines of code (LOC) and number of decisions (WDC) and pathways (CYCO) are important factors influencing estimation of complexity.

Similar diagrams are presented for the other two quality attributes: maintainability (Figure 5), and usability (Figure 6).

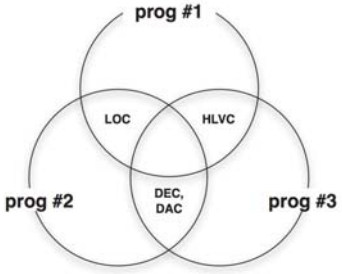


Figure 5. Maintainability

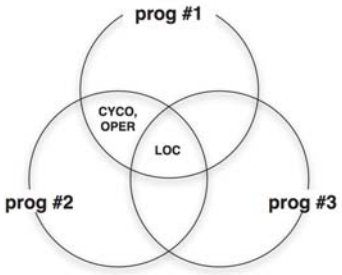


Figure 6. Usability

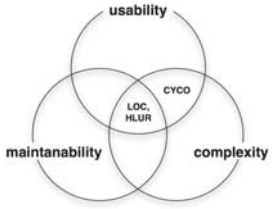
It seems that for maintainability, Figure 5, the programmers share at least one "common" metric describing maintainability. We see that lines of code (LOC), sum of operators and operands (HLVC), and number of decision keywords (DEC) and number of reference types (DAC) influence programmers' evaluation processes.

For the case of usability, all programmers where recognizing lines of code (LOC) as an important judging criteria. Additionally, for both **programmers_#1** and **_#2** number of loops (CYCO) and number of operations (OPER) are deciding factors about usability of objects.

B. Single Individual Across Multiple Quality Attribues

Let us take a look how each programmer has evaluated software objects when he/she was asked to appraise complexity, usability, and maintainability of objects.

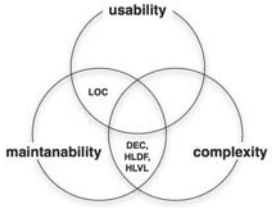
The **programmer_#1** consistently "use" two factors for all three evaluations, Figure 7 (a). These two factors are lines of code (LOC) and number of unique operators (HLUR). A number of pathways (CYCO) is an important factor in estimating usability and complexity. There is no common metric describing evaluation of usability and maintainability. Figure 7 (b) represents a tab-cloud [6] that characterizes the programmer. The tag-cloud has been created based on metrics obtained during StepA, StepB and StepC of the data analysis process (Section VI). The size of the fonts indicates importance of shown matrics. Once again we see HLUR, LOC, CYCO, HLVC as important "labels" describing the programmer.



(a) (b)

Figure 7. Programmer_#1: overlapping metrics (a), and tag-cloud (b)

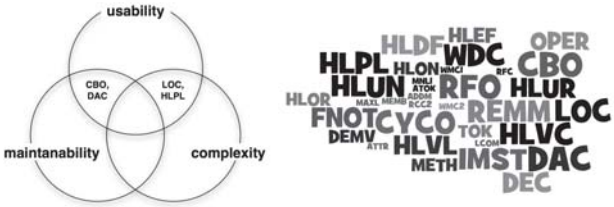
The **programmer_#2** is "shown" in Figure 8. Once again we see lines of code (LOC) as an important factor together with number of decision keywords (DEC), Halstead difficulty (HLDF), and program volume (HLVL). It seems that the **programmer_#2** thinks differently when evaluating usability and complexity/maintainability. The tag-cloud, Figure 8 (b), confirms importance of LOC, HLVC, HLUN, and CBO.



(a) (b)

Figure 8. Programmer_#2: overlapping metrics (a), and tag-cloud (b)

The factors influencing evolution of objects preformed by the **programmer_#3** are shown in Figure 9. It seems that once again some of the metrics we have already seen in the case of the other two programmers are also important for the **programmer_#3**: lines of codes (LOC) and Halstead program length (HLPL) for usability and complexity; number of reference types (DAC) and coupling (CBO) for usability and maintainability. The tag-cloud "adds" HLUN, WDC, RFO, CYCO and REMM.



(a) (b)

Figure 9. Programmer_#3: overlapping metrics (a), and tag-cloud (b)

VII. CONCLUSIONS

The importance of quality of software attributes is unquestionable. It is essential to understand what programmers mean when they say that an object is of high quality, and to know what aspects of software component influence their behavior. The approach we proposed uses multiple feature selection techniques for identifying a small set of software metrics describing a human evaluator. Some metrics representing programmers and relationships among them have been determined. To verify these findings we need to confront the obtained results with the programmers.

REFERENCES

[1] K. El-Emam, W. Melo, and J.C. Machado, "The Prediction of Faulty Classes Using Object-Oriented Design Metrics," J. Systems and Software, 56(1), 2001, pp 63-75.

[2] N. Fenton and M. Neil, "A Critique of Software Defect Prediction Models," IEEE Trans. Software Eng., 25(3), 1999, pp 675-689.

[3] T.M. Khoshgoftaar and N. Seliya, "Comparative Assessment of Software Quality Classification Techniques: An Empirical Case Study," Empirical Software Eng., 9(3), 2004, pp 229-257.

[4] I. Myrtveit, E. Stensrud, and M. Shepperd, "Reliability and Validity in Comparative Studies of Software Prediction Models," IEEE Trans. Software Eng., 31(5), 2005, pp. 380-391.

[5] N.J. Pizzi, R.A. Vivanco, and R.L. Samorjai, "EvIdent: a functional magnetic resonance image analysis system", Artificial Intelligence in Medicine, 21, 2001, pp. 263-269.

[6] G. Smith, Tagging: People-powered Metadata for the Social Web, New Riders Press, 2008.

[7] H. Wang, T.M. Khoshgoftaar, and K. Gao, "High-Dimensional Software Engineering Data and Feature Selection", ICTAI 2009, pp 83-90.

[8] http://en.wikipedia.org/wiki/Feature_selection

[9] http://www.cs.waikato.ac.nz/ml/weka/

[10] http://weka.sourceforge.net/packageMetaData/consistencySubsetEval/index.html

APPENDIX

The attributes of the data used in this paper are briefly described in the Table III.

TYPE	Type: GUI (=1), Data Model (2), Algorithm (3), Other (4)
METH	Number of methods
LOC	Number of lines of code
ALOC	Mean LOC per method
MLOC	Median LOC per method
RCC1	Ratio of comment lines of code to total lines of code including whitespaces and comments
RCC2	True comment ratio
TOK	Number of tokens
ATOK	Mean TOK per method
MTOK	Median TOK per method
DEC	Number of decisions: for, while, if, switch, etc.
ADEC	Mean DEC per method
MDEC	Median DEC per method
WDC	Weighted number of decision based on nesting level
AWDC	Mean WDC per method
MWDC	Median WDC per method
INCL	Number of inner classes
DINH	Depth of inheritance
CHLD	Number of children
SIBL	Number of siblings
FACE	Number of implemented interfaces
RCR	Code reuse: ratio of overloaded inherited methods to methods that are not overloaded
CBO	Coupling between objects
LCOM	Lack of cohesion of methods
RFO	Response for an object (response set contains the methods that can be executed in response to a message bing received by the object
RFC	Response for class
MNL1	Maximum method name length
MNL2	Minimum method name length
MNL3	Mean method name length
MNL4	Median method name length
ATCO	Attribute complexity
CYCO	Cyclomatic complexity
DAC	Data abstraction coupling
FNOT	Fan out
HLDF	Halstead difficulty
HLEF	Halstead effort
HLPL	Halstead program length
HLVC	Halstead program vocabulary
HLVL	Halstead program volume
HLON	Halstead number of operands
HLOR	Halstead number of operators
HLUN	Halstead number of unique operands
HLUR	Halstead number of unique operators
MIC	Method invocation coupling
MAXL	Maximum number of levels
MAXP	Maximum number of parameters
MAXO	Maximum size operations
ATTR	Number of attributes
ADDM	Number of added methods
CLAS	Number of classes
CHCL	Number of child classes
CONS	Number of constructors
IMST	Number of import statements
MEMB	Number of members
OPER	Number of operations
OVRM	Number of overridden methods
REMM	Number of remote methods
PKGM	% of package members
PRVM	% of private members
PROM	% of protected members
PUBM	% of public members
DEMV	Violations of Demeters Law
WMC1	Weighted methods per class (each method weighted by is CYCO)
WMC2	Weighted methods per class (inherited methods excluded)

A Dual Clustering Approach to the Extract Class Refactoring

Keith Cassell, Peter Andreae, Lindsay Groves
School of Engineering and Computer Science
Victoria University of Wellington
Wellington, New Zealand
{*kcassell, pondy, lindsay*}*@ecs.vuw.ac.nz*

***Abstract*—Large classes typically have many internal interactions between their members, making them difficult to understand and expensive to maintain. When large classes are split into smaller, more cohesive classes, maintainability costs can be reduced; however, the very complexity that makes the classes costly to maintain also makes them difficult to split. Our ExtC tool uses clustering techniques to help solve this problem. By clustering based on the structural characteristics of the class, followed by additional clustering based on semantics, ExtC divides the members of large classes into groups. These groups provide the basis for refactoring the large classes into smaller, more cohesive ones.**

***Keywords*-Refactoring; betweenness; clustering; cohesion; extract class; graph theory; maintainability; object-oriented.**

I. Introduction

Code maintenance is expensive. We propose a solution to a common maintenance problem in object-oriented systems – the presence of large, complex classes, sometimes known as *god classes* [1].

In his book on refactoring [2], Fowler identifies a large class as being a "bad smell" that may be a symptom of an underlying software problem, and he recommends the *Extract Class* refactoring as a potential remedy for the problem. One of the challenges in performing this refactoring is determining how to split a large class, i.e. how to group the original class's attributes and methods into revised classes.

Our novel contribution is the sequential use of two complementary clustering techniques to determine how to revise Java classes. We first use a divisive structural clustering technique to separate a class's members into groups based on statically determined dependencies between the members. This typically breaks a large class into multiple groups, some of which may consist of only one or two members. Then we use an agglomerative "semantic"[1] clustering technique to combine the smaller groups with the larger ones. In many cases, these groups can be used to form the basis of two nontrivial classes, both of which are more cohesive than the original.

[1]Some "semantic" techniques, including the vector space model we use, are based on statistics. However, when given sufficient data, some statistical techniques approximate the results given by knowledge-based semantic techniques. We use "semantic" in this paper for convenience and to be consistent with other researchers [3], [4], [5].

II. Refactoring Approach

To help show how the structure of a class relates to its amenability to Extract Class, consider the portion of the intraclass dependency graph shown in Figure 1[2]. Intraclass dependency graphs are concise representations of single classes built from a static analysis of source code. The nodes of a graph represent a class's members – circles for methods and stars for attributes, and each edge represents a method calling a method or a method accessing an attribute.

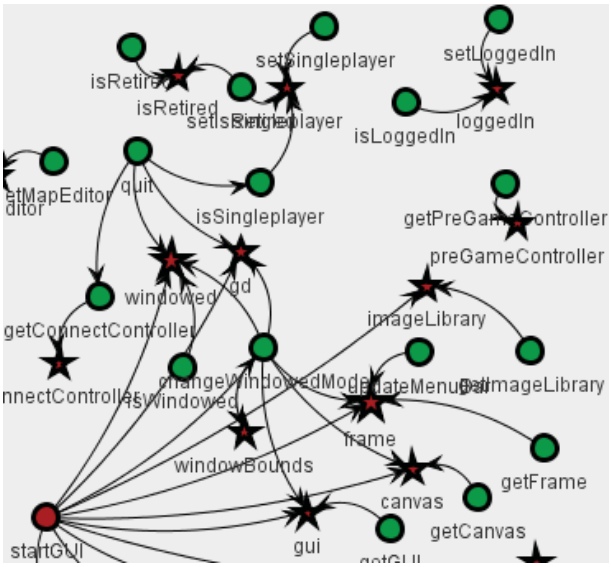


Figure 1. FreeColClient Dependency Graph

The disconnected nature of the example graph shows that this class has several loosely related responsibilities. For example, the members represented by the three nodes in the upper right (`loggedIn`, `isLoggedIn`, and `setLoggedIn`) are not accessed by any other code in the class.

A. Goals

When refactoring to improve maintainability, it is useful to keep as much of the existing structure of the code as possible, while addressing the potential maintainability problems

[2]generated from code at https://svn.freecol.org/svnroot/freecol/freecol/trunk:/src/net/sf/freecol/client/FreeColClient.java

of large classes. Extract Class breaks a class into two classes. We refer to the class that gets refactored as the *original class*, the post-refactoring class that is most like the original class as the *modified class*, and the other post-refactoring class as the *extracted class*. Following refactoring, we want the modified and extracted classes to be smaller and more cohesive than the original class, and the modified class should implement the same interfaces as the original class.

In addition, we want to avoid situations where we replace one maintenance problem with another. For example, in breaking up a large class, we do not want to create trivial classes that do too little (the *lazy class* smell [2]). For example, although it would be cohesive, we would not create a new class from `loggedIn`, `isLoggedIn`, and `setLoggedIn`, because such a class would do too little.

B. Design

Unsupervised clustering techniques group together entities that are highly related, so clustering algorithms should be useful for identifying cohesive new classes, given information about existing classes [6]. Our initial approach to extracting classes [7] used a structural divisive clustering algorithm. By using an algorithm that creates clusters by splitting an intraclass dependency graph, we can be conservative and group members together while keeping existing inter-member relationships intact.

However, while inspecting the dependency graphs of hundreds of large classes, we observed that a significant percentage of the graphs were composed of disconnected subgraphs, and many of these subgraphs consisted of three or fewer members. Our divisive clustering technique does not assist in determining how those isolated members should be assigned to classes, so our initial implementation kept the members represented by the small subgraphs with the original class.

While we still feel that the intraclass structural information should be the primary guide for refactoring, there is additional information available that we had not used. Semantic information embedded in identifier names and comments has been used to measure cohesion [4], to identify topics in source code [5], [8], and to help refactor [3], [4]. We decided to use semantic information as a secondary information source for clustering. Because semantic clustering techniques partially rely on meaningful terms being embedded in the identifiers, we feel they are less reliable than structural information.

Our current dual clustering approach consists of divisive clustering based on structural information, followed by agglomerative clustering based on semantic information. Divisive clustering splits the members of the class into at least two groups, and the agglomerative clustering connects the small groups of members with the larger groups. The two resultant groups serve as inputs for the Extract Class refactoring.

1) Divisive clustering based on structure: We use a structural divisive clustering algorithm known as betweenness clustering [9] for our initial step of breaking apart a large class. Betweenness clustering is a general graph-based technique that has been applied to many domains, including software refactoring [7]. For refactoring large classes, the graph represents the class, the nodes represent the class's members, and the edges represent a dependency between the members, e.g. a method calling a method or a method accessing an attribute. The betweenness value of an edge is the number of shortest paths between pairs of nodes that pass through it. The high betweenness edges tend to be within the centrally located "thin areas" of a graph.

Betweenness clustering operates by iteratively removing the edge with the maximum betweenness value. After each edge removal, the algorithm recalculates the betweenness values of the edges prior to the next edge removal. The successive removal of edges will disconnect the connected portions of the graph and form new clusters.

We run betweenness clustering on undirected versions of intraclass dependency graphs and stop the clustering once the first new cluster is formed. The disconnected subgraphs represent the members of the class that have high interconnectivity (cohesion) amongst themselves relative to members outside the cluster. The two largest clusters serve as seeds for agglomerative clustering.

2) Agglomerative clustering based on semantics: Agglomerative clustering is an iterative process of combining clusters of one or more entities – it starts with seed clusters and adds closely related clusters to them until some stopping criterion is reached. For the purpose of restructuring classes, the entities to be clustered are attributes and methods. The determination of the clusters to combine is done using a *similarity function* or *distance function* that calculates how similar (or distant) two entities are based on the characteristics (or *feature set*) of the entities. Because a similarity function can be considered a kind of distance function (two similar things are less distant conceptually), we generally use the term "distance function" in this paper.

Several researchers [3], [10], [11] have used structural information to create a feature set for use in agglomeratively clustering class members. Because we already make use of structural information in betweenness clustering, we use semantic information instead.

Consider the `retire` method in Figure 2. The only class member it references directly is the `client` attribute; however, there is a wealth of other information available. Some of this is structural information pertaining to other classes, for instance, it calls the `Message` class's `createNewRootElement` method. There is also semantic information encoded in variable names, e.g. `reply` and in string constants, e.g. "highScore".

We use document clustering techniques and treat the class's methods and attributes as "documents". The docu-

```
public boolean retire() {
 Element retireElement =
  Message.createNewRootElement("retire");
 Element reply=client.ask(retireElement);
 boolean result = reply != null
  && "true".equals(
      reply.getAttribute("highScore"));
  return result;
}
```

Figure 2. FreeColClient's retire method

ments' contents are the words present in the identifiers and constants (but not words within comments). Code adapted from TopicXP [8] extracts this information. The words stored include the stemmed form of the full identifier, e.g. `createnewrootel`, and the stemmed form of its components, e.g. `creat` and `root`. Certain common words are filtered out, including Java reserved words like `new` or `boolean`. Words are stored together with their number of occurrences in the document.

We use UCLA's S-Space package [12] to create a *vector space model* of these documents. In a vector space model, documents are represented as vectors, where each element in the vector corresponds to a unique word in the corpus (the collection of documents). We can then compute the distance between two documents (class members) based on the cosine similarity [13] of their vectors.

III. Extract Class Experiments

A. Experimental Design

To test the hypothesis that dual clustering provides an effective way of determining how to split large Java classes, we ran experiments consisting of the following steps:

1) Software metrics identify large noncohesive classes from within open-source software projects.
2) Clustering algorithms form two clusters from each class's members.
3) Using the two clusters as input, an automated Extract Class refactoring separates each of the original classes into two classes.
4) Software metrics quantify the changes caused by the refactoring.

Most of the analysis and clustering in these experiments is done using a series of plug-ins for the Eclipse platform [14]. We use the open-source Metrics2 [15] Eclipse plug-in to collect metric data. Our open-source ExtC (*Ext*ract *C*lass) Eclipse plug-in [16] provides capabilities for identifying god classes using metrics, visualizing those classes, and proposing class splits based on clustering.

1) Identifying large, noncohesive classes: Our god class query checks the following conditions:

1) number of instance methods (NOM) > 20
2) weighted method count (WMC) [17] ≥ 47.0
3) tight class cohesion (TCC) [18] < 0.34
4) depth in the inheritance hierarchy = 1.0

The first three conditions specify the class's size, cohesion, and complexity. The fourth simplifies the analysis by eliminating the effects of inherited members. We use Metrics2 to collect the measurements and store them in a database.

We ran this query on data gathered from four mature open-source projects – FreeCol[3] 0.94, Heritrix[4] 1.8.0, Jena[5] 2.5.5, and Weka[6] 3.6.3, covering a variety of domains. Thirty classes matched the query.

2) Clustering the members: Semantic clustering requires a document corpus from which similarity can be determined. We use code adapted from TopicXP [8] to extract semantic information from the source code and to create documents from the classes' members. We store a corpus of documents for each project. Then, we use UCLA's S-Space package [12] to build vector space models for each of these corpora, which are saved and used later in the clustering step.

ExtC provides a batch mode of operation to cluster the members of the classes that match the god class query. The results determine whether the classes are good candidates for an Extract Class refactoring. If the clustering steps produce two clusters of seven or more members, we refactor. We use a cluster size of seven as a threshold to avoid producing new classes that do too little (lazy classes). (Seven is approximately the average number of methods per class [1].) The dual clustering step does not take long – on average less than seven seconds per class when run on a low end PC.

3) Refactoring the large classes: We use an automated Extract Class refactoring to reduce the variability of how the classes are refactored. This helps reduce the risk of the refactoring and makes evaluation and comparison of the refactoring results more consistent.

Because Eclipse does not have an Extract Class capability capable of moving methods into a new class, we use IntelliJ IDEA's [19] Extract Class refactoring to split off a new class. We manually input the results of the clustering into the IDEA GUI. IDEA does the code manipulation necessary to form the new class and to modify other classes that might be affected by the refactoring.

4) Measuring the modified software: We use Metrics2 to collect measurements on the modified software. The data collected includes the number of instance methods, weighted method count, and tight class cohesion.

B. Data Collected

We collected connectivity data on the intraclass dependency graphs of the matching classes after removing some "special" methods (e.g. constructors and *toString*) that might

[3] http://www.freecol.org/
[4] http://crawler.archive.org/
[5] http://jena.sourceforge.net/
[6] http://www.cs.waikato.ac.nz/ml/weka/

link together functionally unrelated members. The following observations are particularly interesting:

1) Not one of the 30 classes had a completely connected dependency graph.
2) Multiple large groups are relatively rare. Only 20% of the classes had more than one large group, and none had more than two large groups.
3) Multiple small groups are extremely common, with 50% of the classes having seven or more.

These observations lend support to our belief that divisive clustering techniques alone do not adequately address the allocation of all of the class members to revised classes.

Based on the results of the clustering, seven of the 30 classes did not warrant refactoring, because they did not produce two clusters with at least seven members each. Four of those classes did not even produce two clusters of at least four members.

Table I summarizes the results for the 22 classes[7] where a class of seven or more members could be extracted based on the clustering results. The letter in parentheses after each class name indicates the project from which the class came. Following a column that indicates the change in cohesion values (ΔTCC), there are three groupings of columns. The *Original Class* columns contain the measurements for the class before refactoring, while the *Modified Class* columns contain the measurements for that class after a class was extracted from it. The *Extracted Class* columns contain the measurements for the class that was extracted from the original. The following measurements are shown:

- *#A* - number of attributes.
- *#M* - number of instance (non-static) methods.
- *TCC* - Tight Class Cohesion ranges from 0 (least cohesive) to 1 (most cohesive).
- *WMC* - Weighted Methods per Class ranges from 0 (least complexity) upwards.

The classes are sorted according to the change in TCC values between the original class and the modified class.

There is a peculiarity in the table. In the first row, the extracted class for `Rule` has no instance attributes or methods and has a TCC of 0. This class is entirely composed of static methods. Because TCC is calculated based on the connections between methods and attributes (of which there are none), the TCC score is 0, even though the methods of the class call each other.

C. Analysis of Results

In general, there were slight improvements in both cohesion (TCC) and complexity (WMC). 55% of the 22 classes showed cohesion increasing or staying the same for both the modified and extracted classes, and an additional 41% showed cohesion increasing for either the modified or the extracted class.

The modified classes that had a poor ΔTCC generally had an associated extracted class with a high TCC score. All told, 45% of the 22 extracted classes had a TCC of at least 0.5, indicating that these classes are relatively cohesive. Two of these classes had the maximal score of 1.0.

It is not surprising that most improvements in TCC are small. TCC measures the proportion of connected methods to the maximum possible number of connected methods. The large classes that were selected by our god class query had low TCC scores largely because they had many disconnected members, as indicated by the large number of disconnected subgraphs in the dependency graphs. Splitting such a class can eliminate some of the disconnectedness; however, many of the small, structurally disconnected portions of the original class become small, structurally disconnected portions of either the modified class or the extracted class.

One class, Weka's `Node`, showed decreased cohesion for both the modified and extracted classes. It is interesting to see why. `Node`'s intraclass dependency graph contains 11 disconnected subgraphs, most of which consist of a single field and its accessors. Only one of the subgraphs has seven members or more, and this larger subgraph is relatively cohesive. However, due to how our algorithm works, the large group gets broken apart to form seeds and the smaller groups agglomerate with them. This contrasts with the more typical case of `FreeColClient` (Figure 1), where the initial seeds were not derived from the same cohesive group.

One result that may be surprising is that the modified class often has about the same number of methods as the original. One might expect that the count would have decreased by the number of methods that had migrated to the extracted class. This "surprise" is due to the conservative nature of IDEA's Extract Class refactoring, which maintains the public interface of the class by using delegation. The logic of the original method is moved to the extracted class, while the class from which it was extracted has a proxy method that calls the moved method. This conservative approach is necessary to avoid potentially breaking an external client's code. If it is known that there are no external clients, then the delegation could be removed in many cases, because the calling code of the internal clients could be adjusted at the time of the refactoring.

As expected, almost all classes showed a decrease in complexity as measured by WMC. Because WMC is not a normalized metric, removing methods from a class generally decreases the class's WMC score.

IV. Comparison with Related Work

A variety of techniques have been proposed for making large classes more maintainable. In this section, we discuss those that concentrate on refactoring large classes using

[7]IDEA's Extract Class failed to operate on one of the classes to be refactored.

Table I
REFACTORING RESULTS

Class Name	Δ TCC	Original Class				Modified Class				Extracted Class			
		#A	#M	TCC	WMC	#A	#M	TCC	WMC	#A	#M	TCC	WMC
Rule (J)	0.15	6	32	0.17	109	6	32	0.32	76	0	0	0.00	33
Node (J)	0.14	1	30	0.08	51	1	30	0.21	44	2	15	0.30	22
FreeColClient (F)	0.13	26	46	0.05	94	17	46	0.18	89	10	19	0.11	24
CandidateURI (H)	0.12	10	54	0.13	84	11	56	0.25	81	1	18	0.01	23
ParserBase (J)	0.11	18	34	0.06	93	16	34	0.17	44	3	12	0.17	61
BruleEngine (J)	0.09	8	21	0.26	57	7	21	0.35	57	2	10	0.50	10
N3JenaWriterC... (J)	0.06	19	40	0.22	136	17	42	0.28	119	4	16	0.07	35
FreeColServer (F)	0.06	18	48	0.14	181	13	48	0.20	166	6	12	0.09	27
SettingsHandler (H)	0.05	4	25	0.32	60	4	25	0.37	48	2	16	0.87	28
Specification (F)	0.05	36	86	0.14	155	34	86	0.18	139	8	40	0.51	73
RegOptimizer (W)	0.04	20	22	0.29	48	5	24	0.33	36	17	28	0.21	42
ImageLibrary (F)	0.04	1	61	0.22	109	2	6 1	0.26	93	1	19	0.02	35
CommandLine (J)	0.02	8	36	0.22	66	8	36	0.25	59	1	16	0.87	23
DatabaseUtils (W)	0.01	17	44	0.26	194	16	44	0.26	190	2	7	0.60	11
FreeColObject (F)	0.01	2	51	0.11	99	2	52	0.12	95	1	15	1.00	29
Script (W)	0.00	6	31	0.30	72	9	39	0.31	82	2	7	0.73	9
CrawlController (H)	0.00	38	98	0.02	247	37	98	0.20	230	2	7	0.67	24
ResultMatrix (W)	0.00	37	127	0.13	305	36	127	0.13	303	2	9	0.71	11
Heritrix (H)	-0.01	7	48	0.20	293	7	48	0.20	291	2	11	1.00	13
LPBRuleEngine (J)	-0.02	8	26	0.32	47	6	27	0.30	43	4	13	0.41	18
Node (W)	-0.05	15	29	0.12	63	14	30	0.07	39	3	8	0.04	33
XMLDocument (W)	-0.07	7	35	0.27	61	7	3 6	0.20	52	2	8	0.47	18
Average	0.04	13.6	44.5	0.18	114.1	12.0	45.3	0.22	103	3.35	13.3	0.41	26
Max	0.15	38	127	0.32	305	37	127	0.37	303	17	40	1.00	73
Min	-0.07	1	21	0.05	47	1	21	0.07	36	0	0	0	9

clustering, rather than on those that do more extensive reorganizations of class hierarchies [20].

Other researchers have used agglomerative clustering to determine how to refactor. Serban and Czibula's CASYR algorithm [11] creates feature sets for classes, methods and attributes based on structural information about methods calling other methods or methods accessing attributes. For example, a method's feature set will include the names of the method itself and the attributes accessed by it, and an attribute's feature set will include the names of the attribute itself and the methods accessing it. Serban and Czibula use a *Jaccard similarity* function to determine the similarity between feature sets, where the similarity score is equal to the number of common features divided by the total number of features for the entities being compared. The JDeodorant Eclipse plug-in uses a similar agglomerative technique to extract classes [10] but uses a slightly different feature set.

We contend that betweenness clustering using an intra-class dependency graph has advantages over agglomerative clustering using local structural information. Betweenness clustering's primary advantage is that it uses extended structural information in addition to local information – the overall relationship of a member to other members is embedded in the intraclass dependency graph on which the betweenness algorithm works, and this information is used when determining clusters. Furthermore, due to how it operates, betweenness clustering retains most existing intraclass member relationships. The edges removed from the graph to form a cut set are relationships that will be re-established between the members of the evolved classes.

Researchers based predominately at the University of Salerno [3], [4] are the only ones that we know of who use both structural and semantic information to cluster members and extract classes; however, their approach is significantly different from ours. Their most recent approach [3] combines structural and semantic information to calculate pairwise similarities between all of a class's members. Their similarity function has three weighted terms. Two pertain to structural characteristics of the code, while the third uses latent semantic indexing to compute the semantic similarity between members.

Using these similarity measures, they create a graph whose nodes are the class's members and whose weighted edges contain the similarity scores between the connected nodes, so initially, the graph is fully connected. Then, all edges with a weight below a certain threshold are removed, which disconnects the graph. In their second step, they combine the small groups with the larger ones using the same similarity function discussed above.

We believe that betweenness clustering on the actual intraclass dependency graph is superior to their first step of creating a disconnected graph whose nodes are linked by similarity scores. Because their graph is built using similarity scores, it does not necessarily retain existing call relationships between members within the class. Betweenness does retain most of the existing intraclass relationships, so using

it as the basis for new class formation should generally entail less change, and therefore, less risk.

V. Conclusions

In this paper, we advocate using clustering techniques to determine how to refactor large classes. Our approach to identifying extract class opportunities goes beyond previous attempts by using two distinct clustering phases. The first uses structural information to split the members of a large class into groups. The second uses semantic information to combine the smaller groups with the larger ones.

To test the hypothesis that our clustering techniques offer useful advice for splitting classes, we ran the algorithm against 30 open source classes and collected metric data both on the original classes and on the refactored classes. Based on this data, it appears that our clustering techniques can effectively identify how to split some large classes, but not others. For 40% of the 30 classes, we were able to create two more cohesive classes from the original class.

There is much potential future work. There are many uncertainties about how best to apply clustering techniques to software, including a need for additional study of semantic clustering. While semantics have an intuitive appeal, it is not clear how they can best be applied to software engineering in general, and refactoring in particular. Besides the techniques themselves, there are also questions about the "data" on which they should operate, for example, whether both code and comments should be considered as input.

While there is still much work to do, we feel that clustering techniques already provide benefit for a number of software engineering tasks. In particular, this paper has demonstrated how complementary clustering techniques can be used to refactor large classes to make them measurably more more cohesive and less complex.

References

[1] M. Lanza and R. Marinescu, *Object-Oriented Metrics in Practice*. Springer-Verlag New York, Inc., 2006.

[2] M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts, *Refactoring : Improving the Design of Existing Code*. Boston: Addison-Wesley, 1999.

[3] G. Bavota, A. D. Lucia, A. Marcus, and R. Oliveto, "A two-step technique for extract class refactoring," in *Proc. of the IEEE/ACM International Conf. on Automated Software Engineering*. Antwerp, Belgium: ACM, 2010, pp. 151–154.

[4] A. D. Lucia, R. Oliveto, and L. Vorraro, "Using structural and semantic metrics to improve class cohesion," in *IEEE International Conf. on Software Maintenance, 2008.*, Beijing, Sep. 2008, pp. 27 – 36.

[5] A. Marcus, "Semantic driven program analysis," in *Proc. of the 20th IEEE International Conf. on Software Maintenance*. IEEE Computer Society, 2004, pp. 469–473.

[6] N. Anquetil, C. Fourrier, and T. C. Lethbridge, "Experiments with clustering as a software remodularization method," in *Proc. of the Sixth Working Conf. on Reverse Engineering*. IEEE Computer Society, 1999, p. 235.

[7] K. Cassell, P. Andreae, L. Groves, and J. Noble, "Towards automating class-splitting using betweenness clustering," in *24th IEEE/ACM International Conf. on Automated Software Engineering*, Auckland, NZ, Nov. 2009, pp. 595–599.

[8] T. Savage, B. Dit, M. Gethers, and D. Poshyvanyk, "TopicXP: exploring topics in source code using latent dirichlet allocation," in *Proc. of 26th IEEE International Conf. on Software Maintenance*, Timioara, Romania, Sep. 2010.

[9] M. Girvan and M. Newman, "Community structure in social and biological networks." *Proc Natl Acad Sci U S A*, vol. 99, no. 12, Jun. 2002.

[10] M. Fokaefs, N. Tsantalis, A. Chatzigeorgiou, and J. Sander, "Decomposing object-oriented class modules using an agglomerative clustering technique," in *IEEE International Conf. on Software Maintenance*. Los Alamitos, CA, USA: IEEE Computer Society, 2009, pp. 93–101.

[11] G. Serban and I. Czibula, "Object-oriented software systems restructuring through clustering," in *Artificial Intelligence and Soft Computing - ICAISC 2008*. Berlin / Heidelberg: Springer-Verlag, 2008, pp. 693–704.

[12] D. Jurgens and K. Stevens, "The S-Space package: An open source package for word space models," in *System Papers of the Association of Computational Linguistics*, Uppsala, Sweden, Jul. 2010, pp. 30–35.

[13] A. Strehl, J. Ghosh, and R. Mooney, "Impact of similarity measures on web-page clustering," *Workshop on Artificial Intelligence for Web Search (AAAI 2000)*, pp. 58—64, 2000.

[14] S. Shavor, J. D'Anjou, S. Fairbrother, D. Kehn, J. Kellerman, and P. McCarthy, *The Java(TM) Developer's Guide to Eclipse*. Addison-Wesley Professional, May 2003.

[15] F. Sauer and G. Boissier, "Eclipse metrics plugin continued," http://metrics2.sourceforge.net/, 2010.

[16] K. Cassell, C. Anslow, L. Groves, and P. Andreae, "Visualizing the refactoring of classes via clustering," in *Thirty-Fourth Australasian Computer Science Conf. (ACSC 2011)*, Perth, Australia, Jan. 2011, pp. 63–72.

[17] S. Chidamber and C. Kemerer, "A metrics suite for object oriented design," *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476–493, 1994.

[18] J. M. Bieman and B. Kang, "Cohesion and reuse in an object-oriented system," *SIGSOFT Softw. Eng. Notes*, vol. 20, no. SI, pp. 259–262, 1995.

[19] S. Saunders, D. K. Fields, and E. Belayev, *IntelliJ IDEA in Action*. Manning Publications, Mar. 2006.

[20] G. Snelting and F. Tip, "Reengineering class hierarchies using concept analysis," in *Proc. of the 6th ACM SIGSOFT international symposium on Foundations of software engineering*. Lake Buena Vista, Florida, USA: ACM, 1998, pp. 99–110.

An Empirical Study of Software Metrics Selection Using Support Vector Machine

Huanjing Wang
Western Kentucky University
Bowling Green, Kentucky 42101
huanjing.wang@wku.edu

Taghi M. Khoshgoftaar
Florida Atlantic University
Boca Raton, Florida 33431
khoshgof@fau.edu

Amri Napolitano
Florida Atlantic University
Boca Raton, Florida 33431
amrifau@gmail.com

***Abstract*—The objective of feature selection is to identify irrelevant and redundant features, which can then be discarded from the analysis. Reducing the number of metrics (features) in a data set can lead to faster software quality model training and improved classifier performance. In this study we focus on feature ranking using linear Support Vector Machines (SVM) which is implemented in WEKA. The contribution of this study is to provide an extensive empirical evaluation of SVM rankers built from imbalanced data. Should the features be removed at each iteration? What should the recommended value be for the tolerance parameter? We address these and other related issues in this work.**

I. Introduction

Software quality models are trained using software measurement data (metrics) and various data mining techniques to get useful information that can be used to find higher quality software production. The characteristics of software metrics (a.k.a. features or attributes) influence the performance and effectiveness of the quality model. Previous studies [1], [2], [3] have shown that the performance of software quality models are improved when irrelevant and redundant features are removed before modeling. In this study, we investigated feature selection by the means of a feature ranking method using linear support vector machines (SVM ranker). This SVM ranker has been implemented in WEKA [4]. WEKA is an open source data mining and machine learning package implemented in JAVA at the University of Waikato. Many researchers and practitioners in the data mining and machine learning community commonly use WEKA, but past work has provided no empirically proven recommendation on the appropriate default values for the parameters $percentToEliminatePerIteration$ (percentage of attributes to be removed at each iteration) and $toleranceParameter$ (for checking the stopping criterion) for the SVM ranker. The parameter values are critical to the performance and running time of the ranker (see Section III). In fact, a bad selection of parameter values can entirely prevent experiment completion. We contend that default values built into WEKA for SVM ranker are not reasonable for experimentation. This work, as far as we know, is the first to conduct comprehensive experimentation with the SVM ranker in WEKA and recommend empirically proven default values for the $percentToEliminatePerIteration$ and $toleranceParameter$ parameters.

Since the introduction of SVM ranker [5], there has been little empirical work on the topic. As far as we know, there is no related work that empirically recommends default settings for the $percentToEliminatePerIteration$ and $toleranceParameter$ parameters for the SVM ranker in the WEKA tool. Therefore, we present, in this work, a comprehensive empirical evaluation of learning from imbalanced data by varying the $percentToEliminatePerIteration$ and $toleranceParameter$ parameters.

In this study, we built classification models using naïve Bayes (NB), multilayer perceptron (MLP), k-nearest neighbors (KNN), support vector machines (SVM), and logistic regression (LR) on the smaller subsets of selected attributes. Each classification model is assessed with the Area Under the ROC (Receiver Operating Characteristic) curve (AUC). The empirical validation of the different models was implemented through a case study of three consecutive releases of a very large telecommunications software system (denoted as LLTS), nine data sets from the Eclipse software project, and three data sets from NASA software project KC1. Our experimentation is done in WEKA using the 15 different data sets with varying degrees of class imbalance. In the experiments, ten runs of five-fold cross-validation were performed. Statistical analysis using Analysis of Variance (ANOVA) models is used to determine reasonable default values for the $percentToEliminatePerIteration$ and $toleranceParameter$ parameters. By varying the parameters $percentToEliminatePerIteration$ and $toleranceParameter$, this work is the first to compare the SVM ranker on imbalanced data and recommend a default setting for each of the parameters.

Experimental results demonstrate the default setting in WEKA for $toleranceParameter$ (1.0E-10) is not appropriate value, but 1.0E-03 is reasonable. This work further shows that SVM ranker with no backward elimination performs similar to or better than the ranker with full backward elimination. Thorough experimentation makes our work extremely comprehensive and dramatically augments the reliability of our conclusions.

The remainder of the paper is organized as follows. Section II presents the feature selection techniques. Section III describes the data sets, experimental design, and a discussion of the results. Section IV presents background on feature selection and the SVM ranker in different domains. Finally, we conclude the paper in Section V and provide suggestions for future work.

II. Feature Selection

Feature selection has been applied in many data mining and machine learning applications [6]. The main goal of feature selection is to select a subset of features that minimizes the prediction errors of classifiers. It is broadly classified as feature ranking and feature subset selection, where feature ranking sorts the attributes according to their individual predictive power, and feature subset selection finds subsets of attributes that collectively have good predictive power. In this study, we investigated a feature ranking method, called SVM ranker.

The Support Vector Machine (SVM) classifier is one of most commonly used classifiers. It builds a linear discrimination function using a small number of critical boundary instances (called support vectors) from each class while ensuring a maximum possible separation [7].

SVM has been extended to form an embedded method of feature selection. A linear classifier is trained and features are ranked based on the weight (calculated from the support vectors) of each feature. The larger the weight, the more important role the feature plays in the decision function. This ranking procedure can be applied recursively. Guyon et al. [5] introduced recursive feature elimination for support vector machines, SVM-RFE. SVM-RFE uses a backward elimination procedure recursively. At each iteration one or more features with the lowest score (weight) are eliminated. The process is repeated until a predefined number of features remains.

SVM ranker has been implemented in WEKA, called *SVMAttributeEval*. By default, WEKA uses $percentToEliminatePerIteration = 0$ and $attsToEliminatePerIteration = 1$ (one feature is removed at each iteration) and $toleranceParameter = 1.0E\text{-}10$. The tolerance parameter defines the stopping criterion for SVM model optimization. The lower the tolerance parameter, the higher the computational complexity. In our experimentation we first change the tolerance parameter to 1.0E-03 to reduce computational cost. Along with the default value of $percentToEliminatePerIteration$ and $attsToEliminatePerIteration$, this variation is called SVM ranker with full backward elimination. Our experiments also set $percentToEliminatePerIteration$ to 100 (regardless of value of $attsToEliminatePerIteration$), indicating no backward elimination, and consider different values for $toleranceParameter$, specifically 1.0E-03, 1.0E-05, 1.0E-07, and 1.0E-10. In total, five SVM rankers are evaluated in this study (tolerance of 1.0E-03 with full backward elimination, and tolerance of 1.0E-03, 1.0E-05, 1.0E-07, and 1.0E-10 with no backward elimination). Our experiments demonstrated that 1.0E-10 is not an appropriate value, and in particular 1.0E-03 is more reasonable. We also showed that SVM ranker with no backward elimination performed similar to or better than the ranker with full backward elimination, while the computational cost of no backward elimination is much lower than full backward elimination.

SVM ranker has been used as feature selection method in several domains. However, for imbalanced data in the software engineering domain, a comprehensive evaluation of the ranker, with varying values for $percentToEliminatePerIteration$ and $toleranceParameter$, has not been performed. Researchers and practitioners who use WEKA have no guidance on these settings and too often rely on the default values in the WEKA tool. Our work shows that the default value for $toleranceParameter$ is not appropriate for experimentation with imbalanced data. To our knowledge, there has been no previous study to empirically recommend $percentToEliminatePerIteration$ and $toleranceParameter$ values for SVM ranker in the WEKA tool.

III. Experiments

A. Experimental Data Sets

Experiments conducted in this study used software metrics and fault data collected from real-world software projects, including a very large telecommunications software system (denoted as LLTS) [1], the Eclipse project [8], and NASA software project KC1 [9].

The software measurement data set of LLTS contains data from four consecutive releases, which are labeled as SP1, SP2, SP3, and SP4. We only provide results for SP2, SP3, and SP4 since we couldn't get results for SP1 when tolerance parameter was set to 1.0E-10 for no backward elimination due to computational complexity. The software measurement data sets consist of 42 software metrics, including 24 product metrics, 14 process metrics, and four execution metrics [1]. The dependent variable is the class of the program module: fault-prone (*fp*) or not fault-prone (*nfp*). A program module with one or more faults is considered *fp*, and *nfp* otherwise.

TABLE I
Software Data Sets Characteristics

	Data	#Metrics	#Modules	%fp	%nfp
LLTS	SP2	42	3981	4.75%	95.25%
	SP3	42	3541	1.33%	98.67%
	SP4	42	3978	2.31%	97.69%
Eclipse	Eclipse2.0-10	208	377	6.1%	93.9%
	Eclipse2.0-5	208	377	13.79%	86.21%
	Eclipse2.0-3	208	377	26.79%	73.21%
	Eclipse2.1-5	208	434	7.83%	92.17%
	Eclipse2.1-4	208	434	11.52%	88.48%
	Eclipse2.1-2	208	434	28.8%	71.2%
	Eclipse3.0-10	208	661	6.2%	93.8%
	Eclipse3.0-5	208	661	14.83%	85.17%
	Eclipse3.0-3	208	661	23.75%	76.25%
NASA	KC1-20	62	145	6.90%	93.10%
	KC1-10	62	145	14.48%	85.52%
	KC1-5	62	145	24.83%	75.17%

From the PROMISE data repository [8], we also obtained the Eclipse defect counts and complexity metrics data set. In particular, we use the metrics and defects data at the software package level. The original data for the Eclipse packages consists of three releases denoted 2.0, 2.1, and 3.0 respectively. We transform the original data by: (1) removing all nonnumeric attributes, including the package names, and (2) converting the post-release defects attribute to a binary class attribute: fault-prone (*fp*) and not fault-prone (*nfp*). Membership in each class is determined by a post-release defects threshold *t*, which separates *fp* from *nfp* packages by classifying packages with *t* or more post-release defects as *fp* and the remaining as *nfp*. In our study, we use $t \in \{10, 5, 3\}$ for release 2.0 and 3.0, while we use $t \in \{5, 4, 2\}$ for release 2.1. These values are selected in order to have data sets with different levels of class imbalance. All nine derived data sets contain 208 independent attributes. Releases 2.0, 2.1, and 3.0 contain 377, 434, and 661 instances respectively.

The original NASA project, KC1 [9], includes 145 instances containing 94 independent attributes each. After removing 32 Halstead derived measures, we have 62 attributes left. We used three different thresholds to define defective instances, thereby obtaining three structures of the preprocessed KC1 data set. The thresholds are 20, 10, and 5, indicating that instances with numbers of defects greater than or equal to 20, 10, or 5 belong to the *fp* class. The three data sets are named KC1-20, KC1-10, and KC1-5.

The fifteen data sets used in the work reflect software projects of different sizes with different proportions of *fp* and *nfp* modules. Table I lists the characteristics of the 15 data sets utilized in this work.

B. Classification Algorithms

The software defect prediction models were built with five commonly used classification algorithms including naïve Bayes, multilayer perceptron, K-nearest neighbors, support vector machine, and logistic regression. The five learners were selected because of their common use in software engineering and other application domains. Unless stated otherwise, we use the default parameter settings for the different learners as specified in WEKA. Parameter settings are changed only when a significant improvement in performance is obtained.

1) Naïve Bayes (NB) [10] utilizes Bayes's rule of conditional probability and is termed 'naive' because it assumes conditional independence of the features.
2) Multilayer Perceptron (MLP) [11] is a neural network of simple neurons called perceptrons. Some related parameters of MLP were set as follows. The 'hiddenLayers' parameter was set to 3 to define a network with one hidden layer containing three nodes. The 'validationSetSize' parameter was set to 10 to cause the classifier to leave 10% of the training data aside to be used as a validation set to determine when to stop the iterative training process.
3) K-Nearest Neighbors (KNN) [12], also called instance-based learning, uses distance-based comparisons. The choice of distance metric is critical. KNN was built with changes to two parameters. The 'distanceWeighting' parameter was set to 'Weight by 1/distance' and the 'kNN' parameter was set to 5.
4) Support Vector Machine (SVM) [13], also called SMO in WEKA, had two changes to the default parameters: the complexity constant 'c' was set to 5.0 and 'build Logistic Models' was set to true. By default, a linear kernel was used.
5) Logistic Regression (LR) [14] is a statistical regression model for categorical prediction by fitting data to a logistic curve.

C. Performance Metric

The classification models are evaluated using the Area Under ROC (Receiver Operating Characteristic) Curve (AUC) performance metric. AUC has been widely used, providing a general idea of predictive potential of the classifier. The ROC curve is used to characterize the trade-off between true positive rate and false positive rate. An ROC curve illustrates the classifier's performance across all decision thresholds, i.e., a value between 0 and 1 that theoretically separates the *fp* and *nfp* modules. AUC is a single-value measurement that ranges from 0 to 1, where a perfect classifier provides an AUC value of 1. It has been shown that AUC has lower variance and is more reliable than other performance metrics (such as precision, recall, or F-measure) [15].

D. Results Analysis

Before using a feature ranking technique, the practitioner must choose how many features to select. These selected features will be used for modeling. In this study, we choose the top $\lceil \log_2 n \rceil$ features that have the highest scores, where n is the number of independent features for a given data set. The reasons why we select the top $\lceil \log_2 n \rceil$ features include (1) no general guidance has been found in related literature on the number of features that should be selected when using a feature ranking technique; (2) a software engineering expert with more than 20 years experience recommended selecting $\lceil \log_2 n \rceil$ number of metrics for software quality prediction; and (3) a recent study [16] showed that $\lceil \log_2 n \rceil$ is appropriate for various learners. Thus, for the three LLTS data sets, $\lceil \log_2 42 \rceil = 6$; for the nine Eclipse data sets, $\lceil \log_2 208 \rceil = 8$; and for the three NASA KC1 data sets $\lceil \log_2 62 \rceil = 6$.

Following the feature selection algorithm (SVM ranker in this study), the classification models are constructed with data sets containing only the selected features. The defect prediction models are evaluated with respect to the AUC performance metric. We used WEKA for the defect prediction model building and testing process. In the experiments, ten runs of five-fold cross-validation were performed. The ten results from the five folds were then combined to produce a single estimation. In total 18,750 models were evaluated during our experiments.

E. Experimental Results

The classification performance results are reported in Table II and III. Note that each value presented in the table is the average over the ten runs of five-fold cross-validation outcomes. Due to paper size limitations, we could not present each individual ranker's performance. We only present results of no backward elimination with tolerance parameters 1.0E-03 and 1.0E-10, and full backward elimination. All the results of three rankers over 15 different software data sets are reported. We also summarize the average performance (last row of table) for each feature selection technique across 15 data sets. The best model across all data sets is indicated in **boldfaced** print. The results demonstrate that, on average, SVM ranker with no backward elimination and tolerance parameter 1.0E-03 outperformed the other SVM rankers when four classifiers (NB, MLP, SVM, and LR) are applied to the selected subset of features. For the KNN learner, SVM ranker with full backward elimination performed best.

We also performed a two-way ANalysis Of VAriance (ANOVA) F test on the classification performance of learners and SVM rankers for the LLTS data sets and over all 15 data sets separately to statistically examine the various effects on performance of the classification models. Due to size limitation, we didn't present ANOVA results for the Eclipse and KC1 data sets. The underlying assumptions of ANOVA were tested and validated prior to statistical analysis. The two factors were designed as follows. Factor A represents five classifiers and Factor B represents five SVM rankers (no backward with four different tolerance parameters and full backward eliminations). The null hypothesis for the ANOVA test is that all the group population means are the same, while the alternate hypothesis is that at least one pair of means is different.

Table IV shows the ANOVA results. It includes two subtables, and each represents the result for each individual case (three data sets of LLTS and all 15 data sets). All the p-values for Factor A and the p-value for Factor B of the LLTS data sets are less than the typical cutoff value of 0.05, indicating that for the classification performance (in terms of AUC), the alternate hypothesis is accepted, namely, at least two group means are significantly different from each other for at least one pair of groups in the corresponding factors or terms. For Factor B of all 15 data sets, the p-value (0.9795) is much greater than the cutoff value of 0.05, which implies no significant difference exists between the five rankers across all 15 data sets. The p-values for the interaction terms A×B are greater than 0.05, indicating Factor A is the same at every level of B.

We further conducted multiple comparisons for the main factors and interaction term A×B with Tukeys Honestly Significant Difference (HSD) criterion. The multiple comparison results are shown in Figure 1 and 2, where each sub-figure displays graphs with each group mean represented by a symbol (○) and 95% confidence interval around the symbol. Two means are significantly different if their intervals are disjoint, and are not significantly different if their intervals overlap. Matlab was used to perform the ANOVA and multiple comparisons presented in this work. Based on the multiple comparison results, we can conclude the following points:

- For Factor A, LR performed best. This is true regardless of which data sets are used to build classification models. KNN, MLP, and NB produced moderate performance and SVM performed poorly.
- For Factor B, no backward elimination with tolerance parameter 1.0E-03 performed better than full backward elimination .

TABLE II
CLASSIFICATION RESULTS: NB, MLP, AND KNN LEARNERS

	NB			MLP			KNN		
	No Backward		Full Backward	No Backward		Full Backward	No Backward		Full Backward
Data Set	1.0E-03	1.0E-10	1.0E-03	1.0E-03	1.0E-10	1.0E-03	1.0E-03	1.0E-10	1.0E-03
SP2	0.7271	0.7412	0.7336	0.7660	0.7626	0.7645	0.7498	0.7361	0.7263
SP3	0.7227	0.7146	0.7255	0.7104	0.7035	0.7092	0.7445	0.7289	0.7273
SP4	0.6741	0.6612	0.6706	0.7520	0.7442	0.7406	0.7750	0.7497	0.7575
Eclipse2.0-10	0.8786	0.8787	0.8602	0.8646	0.8645	0.8504	0.8694	0.8696	0.8780
Eclipse2.0-5	0.8594	0.8587	0.8776	0.8713	0.8701	0.8872	0.8541	0.8538	0.8665
Eclipse2.0-3	0.8017	0.8017	0.8136	0.7970	0.7970	0.8020	0.7844	0.7844	0.8100
Eclipse2.1-5	0.8496	0.8496	0.8249	0.8105	0.8105	0.8129	0.8547	0.8547	0.8578
Eclipse2.1-4	0.8049	0.8055	0.8076	0.8252	0.8263	0.8409	0.8416	0.8418	0.8493
Eclipse2.1-2	0.8155	0.8156	0.8005	0.8542	0.8543	0.8434	0.8019	0.8010	0.8302
Eclipse3.0-10	0.8870	0.8868	0.8911	0.8535	0.8535	0.8673	0.8715	0.8710	0.8926
Eclipse3.0-5	0.8868	0.8868	0.8926	0.9094	0.9094	0.8978	0.8783	0.8783	0.9115
Eclipse3.0-3	0.8584	0.8593	0.8489	0.8913	0.8919	0.8851	0.8415	0.8412	0.8719
KC1-20	0.7285	0.7283	0.7105	0.7423	0.7415	0.7504	0.7411	0.7370	0.7734
KC1-10	0.7373	0.7363	0.7386	0.7161	0.7134	0.7186	0.7286	0.7273	0.7900
KC1-5	0.8647	0.8647	0.8711	0.7617	0.7630	0.7425	0.8136	0.8136	0.8384
Average	**0.8064**	0.8059	0.8045	**0.8084**	0.8070	0.8075	0.8100	0.8059	**0.8254**

TABLE III
CLASSIFICATION RESULTS: SVM AND LR LEARNERS

	SVM			LR		
	No Backward		Full Backward	No Backward		Full Backward
Data Set	1.0E-03	1.0E-10	1.0E-03	1.0E-03	1.0E-10	1.0E-03
SP2	0.6444	0.6052	0.5979	0.7803	0.7803	0.7800
SP3	0.6019	0.6128	0.5739	0.7401	0.7359	0.7139
SP4	0.6179	0.6356	0.6049	0.7599	0.7488	0.7503
Eclipse2.0-10	0.8737	0.8743	0.8582	0.8546	0.8557	0.8398
Eclipse2.0-5	0.8987	0.8982	0.9102	0.8886	0.8878	0.9017
Eclipse2.0-3	0.8363	0.8363	0.8533	0.8323	0.8323	0.8487
Eclipse2.1-5	0.8559	0.8560	0.8367	0.8504	0.8504	0.8277
Eclipse2.1-4	0.8490	0.8486	0.8499	0.8334	0.8325	0.8536
Eclipse2.1-2	0.8788	0.8787	0.8672	0.8742	0.8742	0.8595
Eclipse3.0-10	0.8903	0.8903	0.8897	0.8825	0.8825	0.8874
Eclipse3.0-5	0.9337	0.9337	0.9247	0.9292	0.9292	0.9216
Eclipse3.0-3	0.9098	0.9097	0.9002	0.9151	0.9151	0.9106
KC1-20	0.7632	0.7623	0.7673	0.7467	0.7465	0.7563
KC1-10	0.7393	0.7394	0.7417	0.7167	0.7188	0.7273
KC1-5	0.8086	0.8085	0.8109	0.7617	0.7617	0.7147
Average	**0.8068**	0.8060	0.7991	**0.8244**	0.8234	0.8195

TABLE IV
TWO-WAY ANALYSIS OF VARIANCE

(a) LLTS

Source	Sum Sq.	d.f.	Mean Sq.	F	p-value
A	2.16446	4	0.54111	499.23	0
B	0.01316	4	0.00329	3.03	0.017
A×B	0.01809	16	0.00113	1.04	0.4081
Error	0.78582	725	0.00108		
Total	2.98153	749			

(b) Fifteen Data Sets

Source	Sum Sq.	d.f.	Mean Sq.	F	p-value
A	0.1738	4	0.04346	6.79	0
B	0.0028	4	0.0007	0.11	0.9795
A×B	0.0486	16	0.00304	0.47	0.9598
Error	23.8446	3725	0.0064		
Total	24.0699	3749			

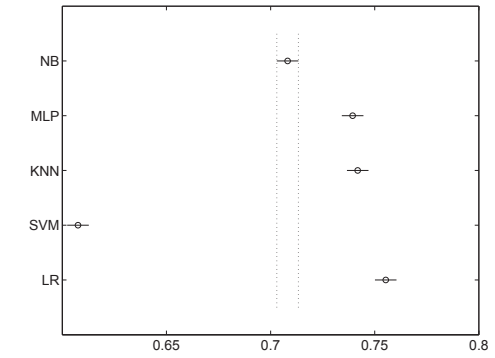
- For interaction A × B, 25 groups produced by five learners combined with 5 rankers are presented. It can be seen that Factor A was different at every level (groups) of Factor B. For example, LR performed better than other learners when no backward elimination with tolerance parameter 1.0E-03 ranker is used to select features.

F. Threats to Validity

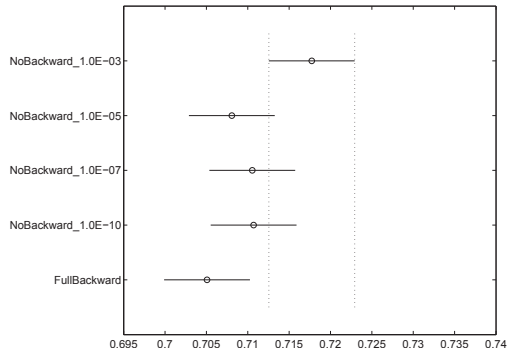
A typical software development project is very human intensive, which can affect many aspects of the development process including software quality and defect occurrence. Consequently, software engineering research that utilizes controlled experiments for evaluating the usefulness of empirical models is not practical. Experimental research commonly includes a discussion of two different types of threats to validity.

In an empirical software engineering effort, threats to external validity are conditions that limit generalization of case study results. The analysis and conclusion presented in this article are based upon the metrics and defect data obtained from 15 data sets of three software projects. The benchmark WEKA data mining tool was used for feature selection (SVM ranker) and all learners, and all of the parameter settings have been included in this work to allow the experiments to be repeated. The parameters for the SVM rankers were chosen to ensure good performance in many different circumstances and to be reasonable for the imbalanced data sets. Experimentation with different settings for the $tolerance Parameter$ provides guidance to the research community as to the recommended value for that parameter. Our comparative analysis can easily be applied to another software system. Moreover, since all our final conclusions are based on ten runs of five-fold cross-validation and statistical tests for significance, our findings are grounded in using sound methods.

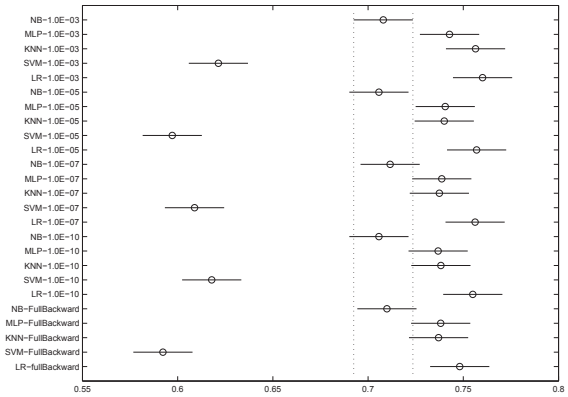
Threats to internal validity are unaccounted for influences on the experiments that may affect case study results. Poor fault proneness estimates can be caused by a wide variety of factors, including measurement errors while collecting and recording software metrics, modeling errors due to the unskilled use of software applications, errors in model-selection during the modeling process, and the presence of outliers and noise in the training data set. Measurement errors are inherent to the data collection effort. In our experiments we utilize 15 real-world imbalanced software data sets, which greatly enhances the reliability of our conclusions. Performing numerous repetitions of cross validation greatly reduces the likelihood of anomalous results due to selecting a lucky or unlucky partition of the data. Moreover, the experiments and statistical analysis were performed by only one skilled person in order to keep modeling errors to a minimum.



(a) Factor A

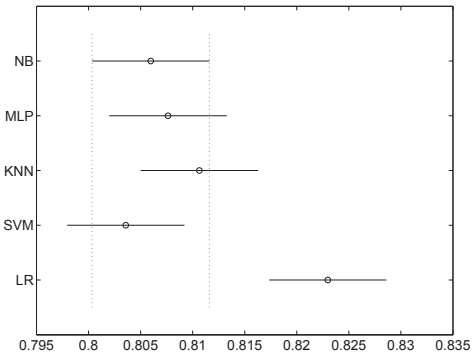


(b) Factor B

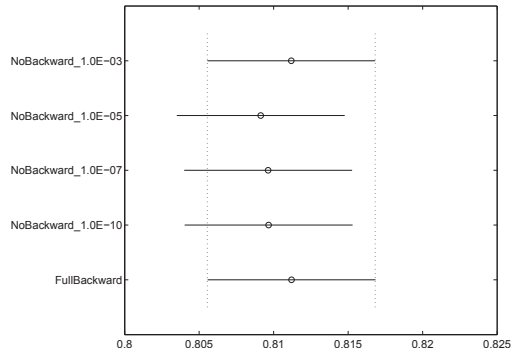


(c) Factor A×B

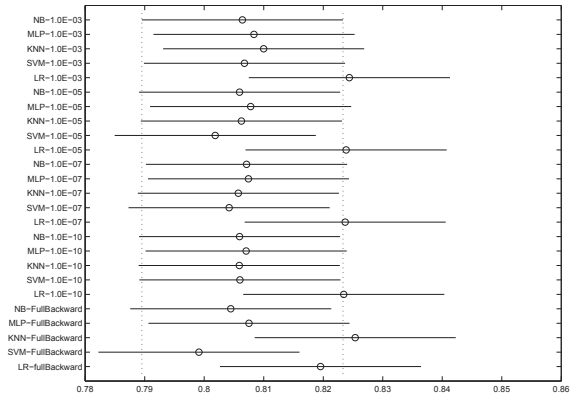
Fig. 1. LLTS: Multiple Comparison in Terms of AUC



(a) Factor A



(b) Factor B



(c) Factor A×B

Fig. 2. Fifteen Data Sets: Multiple Comparison in Terms of AUC

IV. Related Work

The main goal of feature selection is to select a subset of features that minimizes the prediction errors of classifiers. Feature selection has been applied in many data mining and machine learning applications. A good overview on feature selection was provided by Guyon and Elisseeff [17]. They outlined key approaches used for attribute selection, including feature construction, feature ranking, multivariate feature selection, efficient search methods, and feature validity assessment methods. Liu and Yu [18] provided a comprehensive survey of feature selection algorithms and presented an integrated approach to intelligent feature selection.

In this paper, we examine a feature ranking technique, called SVM ranker. Mladenic et al. [19] investigated three feature weighting methods and conclude that feature selection using weights from linear SVMs yields better classification performance than other feature weighting methods when combined with the three explored learning algorithms including NB, Perceptron, and SVM. Chang and Lin [20] showed that linear SVMs with simple feature rankings are effective on data sets in the Causality Challenge. The goal of Causality Challenge [21] is to investigate problems in which the training and testing sets might have different class distributions.

SVM ranker is used in the bioinformatics domain recently [22], [23]. Canual-Reich et al. [22] compared a feature perturbation method to SVM-RFE, 50% of features were removed at every iteration until 10% of total initial features remain. Results demonstrates that the perturbation method outperformed SVM-RFE for most sets of features. Abeel et al. [23] studied an ensemble feature selection method for biomarker identification. An ensemble of a single feature selection method (SVM-RFE, 20% features were removed at each iteration) was built. Experimental results show that the ensemble

feature selection method improves classification performances and biomarker stability.

We also noticed that although feature selection has been widely applied in many application domains for many years, its applications in the software quality and reliability engineering domain are limited. Chen et al. [24] have studied the applications of wrapper-based feature selection in the context of software cost/effort estimation. They conclude that the reduced data set improved the estimation. Rodriguez et al. [3] evaluated three filter- and three wrapper-based models for software metrics and defect data sets, with the conclusion that wrappers were better than filters but at a high computational cost.

V. Conclusion

This work presented a comprehensive experimental analysis of the performance of linear SVM for feature ranking (SVM ranker) with different $percentToEliminatePerIteration$ and $toleranceParameter$ parameters. Much of the related work on SVM ranker has not focused on selecting reasonable values for the two important parameters, $percentToEliminatePerIteration$ and $toleranceParameter$. In most previous studies, researchers and practitioners simply utilize a particular value for $percentToEliminatePerIteration$ and $toleranceParameter$ without much supporting evidence. Our analysis provides guidelines to researchers and practitioners in data mining and machine learning to select a value of 100 for $percentToEliminatePerIteration$ and 1.0E-03 for $toleranceParameter$ when selecting a subset of features.

In the experiments, we first set $percentToEliminatePerIteration$ to 100 (no backward elimination) and considered four different $toleranceParameter$ values (1.0E-03, 1.0E-05, 1.0E-07, and 1.0E-10). We then set $percentToEliminatePerIteration$ to 0 and using the default value of 1 for $attsToEliminatePerIteration$ (full backward elimination) and $toleranceParameter$ to 1.0E-03. In total, five SVM rankers are considered. Our experimentation on the SVM rankers is done in WEKA using 15 different data sets from three real-world software projects with varying degrees of class imbalance. Classification models were built using five commonly used learners: naïve Bayes (NB), multilayer perceptron (MLP), K-nearest neighbors (KNN), support vector machines (SVM), and logistic regression (LR) with the selected features. These models are used to identify faulty software modules. The conclusions of our experimentation show that the default settings in WEKA for $percentToEliminatePerIteration$ (0) and $toleranceParameter$ (1.0E-10) are not appropriate values, and in particular 100 for $percentToEliminatePerIteration$ and 1.0E-03 for $toleranceParameter$ are more reasonable. This work further shows the robustness of the LR learner when compared to four other learners. Thorough experimentation makes our work extremely comprehensive and dramatically augments the reliability of our conclusions.

Future work may include experiments using additional data sets from the software engineering domain as well as from other application domains. In addition, different data sampling techniques can be considered in the feature selection process of imbalanced data.

References

[1] K. Gao, T. M. Khoshgoftaar, and H. Wang, "An empirical investigation of filter attribute selection techniques for software quality classification," in *Proceedings of the 10th IEEE International Conference on Information Reuse and Integration*, Las Vegas, Nevada, August 10-12 2009, pp. 272–277.

[2] H. Wang, T. M. Khoshgoftaar, and A. Napolitano, "A comparative study of ensemble feature selection techniques for software defect prediction," in *Proceedings of the Ninth International Conference on Machine Learning and Applications*, Washington DC, USA, December 12-14 2010, pp. 135–140.

[3] D. Rodriguez, R. Ruiz, J. Cuadrado-Gallego, and J. Aguilar-Ruiz, "Detecting fault modules applying feature selection to classifiers," in *Proceedings of 8th IEEE International Conference on Information Reuse and Integration*, Las Vegas, Nevada, August 13-15 2007, pp. 667–672.

[4] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd ed. Morgan Kaufmann, 2005.

[5] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik, "Gene selection for cancer classification using support vector machines," *Mach. Learn.*, vol. 46, pp. 389–422, March 2002.

[6] H. Wang, T. M. Khoshgoftaar, K. Gao, and N. Seliya, "High-dimensional software engineering data and feature selection," in *Proceedings of 21st IEEE International Conference on Tools with Artificial Intelligence*, Newark, NJ, USA, Nov. 2-5 2009, pp. 83–90.

[7] H. Liu, J. Li, and L. Wong, "A comparative study on feature selection and classification methods using gene expression profiles and proteomic patterns," *Genome Informatics*, vol. 13, pp. 51–60, 2002.

[8] T. Zimmermann, R. Premraj, and A. Zeller, "Predicting defects for eclipse," in *ICSEW '07: Proceedings of the 29th International Conference on Software Engineering Workshops*. Washington, DC, USA: IEEE Computer Society, 2007, p. 76.

[9] A. G. Koru, D. Zhang, K. E. Emam, and H. Liu, "An investigation into the functional form of the size-defect relationship for software modules," *IEEE Trans. Software Eng.*, vol. 35, no. 2, pp. 293–304, 2009.

[10] G. H. John and P. Langley, "Estimating continuous distributions in bayesian classifiers," in *Proceedings of Eleventh Conference on Uncertainty in Artificial Intelligence*, vol. 2, San Mateo, 1995, pp. 338–345.

[11] S. Haykin, *Neural Networks: A Comprehensive Foundation*, 2nd ed. Prentice-Hall, 1998.

[12] D. W. Aha, D. Kibler, and M. K. Albert, "Instance-based learning algorithms," *Machine Learning*, vol. 6, no. 1, pp. 1573–0565, January 1991.

[13] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, 2000.

[14] S. Le Cessie and J. C. Van Houwelingen, "Ridge estimators in logistic regression," *Applied Statistics*, vol. 41, no. 1, pp. 191–201, 1992.

[15] Y. Jiang, J. Lin, B. Cukic, and T. Menzies, "Variance analysis in software fault prediction models," *the 20th IEEE international conference on software reliability engineering*, pp. 99–108, 2009.

[16] T. M. Khoshgoftaar, M. Golawala, and J. Van Hulse, "An empirical study of learning from imbalanced data using random forest," in *Proceedings of the 19th IEEE International Conference on Tools with Artificial Intelligence*, vol. 2, Washington, DC, USA, 2007, pp. 310–317.

[17] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *Journal of Machine Learning Research*, vol. 3, pp. 1157–1182, March 2003.

[18] H. Liu and L. Yu, "Toward integrating feature selection algorithms for classification and clustering," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 4, pp. 491–502, 2005.

[19] D. Mladenić, J. Brank, M. Grobelnik, and N. Milic-Frayling, "Feature selection using linear classifier weights: interaction with classification models," in *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*. New York, NY, USA: ACM, 2004, pp. 234–241.

[20] Y. wen Chang and C. jen Lin, "Feature ranking using linear svm," in *JMLR Workshop and Conference Proceedings*, vol. 3, June 1-6 2008, pp. 53–64.

[21] I. Guyon, C. Aliferis, G. Cooper, A. Elisseeff, J.-P. Pellet, P. Spirtes, and A. Statnikov, "Design and analysis of the causation and prediction challenge," in *JMLR Workshop and Conference Proceedings*, vol. 3, June 1-6 2008, pp. 1–33.

[22] J. Canul-Reich, L. Hall, D. Goldgof, and S. Eschrich, "Feature selection for microarray data by auc analysis," in *IEEE International Conference on Systems, Man and Cybernetics*, 2008, pp. 768 –773.

[23] T. Abeel, T. Helleputte, Y. Van de Peer, P. Dupont, and Y. Saeys, "Robust biomarker identification for cancer diagnosis with ensemble feature selection methods," *Bioinformatics*, vol. 26, pp. 392–398, February 2010.

[24] Z. Chen, T. Menzies, D. Port, and B. Boehm, "Finding the right data for software cost modeling," *IEEE Software*, no. 22, pp. 38–46, 2005.

Software Defect Prediction for High-Dimensional and Class-Imbalanced Data

Kehan Gao
Eastern Connecticut State University
Willimantic, Connecticut 06226
gaok@easternct.edu

Taghi M. Khoshgoftaar
Florida Atlantic University
Boca Raton, Florida 33431
khoshgof@fau.edu

***Abstract*—Software quality and reliability can be improved using various techniques during the software development process. One effective method is to utilize software metrics and defect data collected during the software development life cycle and build defect predictors using data mining techniques to estimate the quality of target program modules. Such a strategy allows practitioners to intelligently allocate project resources and focus more on the potentially problematic modules. Effectiveness of a defect predictor is influenced, among other factors, by the quality of input data. Two problems which often arise in the software measurement and defect data are high dimensionality and class imbalance. This paper presents an approach for using feature selection and data sampling together to deal with the problems. Three scenarios are considered: 1) feature selection based on sampled data, and modeling based on original data; 2) feature selection based on sampled data, and modeling based on sampled data; and 3) feature selection based on original data, and modeling based on sampled data. Several software measurement data sets, obtained from the PROMISE repository, are used in the case study. The empirical results demonstrate that classification models built in scenario 1) result in significantly better performance than the models built in the other two scenarios.**

***Index Terms*—software quality classification, high dimensionality, class imbalance, feature selection, data sampling**

I. Introduction

The success or failure of a software project depends on the product's quality and reliability. Software practitioners collect software metrics and fault data during the software development process, and then analyze the data for defect prediction modeling. Typically, a software quality estimation model is trained using software metrics and defect data collected from prior development experiences of the organization, and then the trained model is applied to the project under development. This helps practitioners strategically allocate project resources, for example by assigning more inspection and testing to the potentially problematic modules.

The effectiveness of software quality estimation models is influenced, among other reasons, by two key quality of data factors: (1) the set of software metrics (predictors or independent attributes) used to build the models, and (2) the proportion of minority (i.e., fault-prone) instances in the software measurement data set. Related literature has shown that an overabundance of features (i.e., attributes) exists in various software project data repositories [1], [2]. In addition, studies also show that not all features make equally important contributions to the dependent variable. Selecting a subset of features that are most relevant to the class attribute is necessary and may result in better predictions [3]. In the context of the software quality classification problem (e.g., classifying program modules into fault-prone (*fp*) and not-fault-prone (*nfp*) groups), class imbalance (or skewed data) occurs frequently [4]. The class imbalance problem occurs when, for a given data set, instances (program modules) of one class are clearly outnumbered by the instances of the other class. For software measurement and defect data, the *fp* modules are often much fewer than the *nfp* modules. A classic learner trained on such an imbalanced data set is likely to have a large number of misclassifications of the minority class (e.g., *fp* modules) instances. This is an extremely severe problem in the software quality assurance domain, as the model would lead to missed opportunities to re-inspect and correct a poor quality module prior to system deployment. Data sampling is a proven method for alleviating the problems associated with class imbalance [4].

This study investigates a process that combines feature selection and data sampling to deal with the problems of high dimensionality and class imbalance that exist during software defect prediction. We used nine filter-based feature selection techniques, which come from three different families. Among the nine techniques, three of them are standard feature ranking methods, five of them are threshold-based feature selection techniques we recently proposed [5], and the last one is signal-to-noise, a rarely-used feature ranking method. We used only one data sampling technique, that is random undersampling.

The process of using feature selection and data sampling may lead to several different scenarios depending on whether feature selection takes place before or after data sampling and which data set, sampled or unsampled data, is used to build a classifier. Three scenarios are examined in this study:

- Scenario 0: data sampling takes place **before** feature selection is performed, and then a classifier is built using the features selected and **unsampled** (original) data.
- Scenario 1: data sampling takes place **before** feature selection is performed, and then a classifier is built using the features selected and **sampled** data.
- Scenario 2: data sampling takes place **after** feature selection is performed, and then a classifier is built using the features selected and **sampled** data.

Two more scenarios are also produced, where one technique (feature selection or data sampling) is used alone, However, we ignore these two options in this study, because all software data sets investigated exhibit both the class imbalance and high-dimensionality problems. To our knowledge, this paper is one of the very few studies which have considered both feature selection and data sampling together.

In order to compare the effectiveness of the three approaches (scenarios), we conducted a case study based on nine software measurement and defect data sets obtained from the PROMISE software project repository [6]. The key contributions of this paper are summarized as follows: (1) Using filter-based feature selection techniques combined with sampling to deal with the common problems of software data sets, i.e., high dimensionality and class imbalance. (2) Exploring three approaches (scenarios) when using feature selection and data sampling simultaneously. (3) Employing nine different feature ranking techniques from three different families to make the results easier to generalize.

The remainder of the paper is organized as follows. Section II describes related work. The nine filter-based feature selection techniques and the random undersampling method, as well as the

classifier and the associated performance metric used in this study are presented in Section III. A case study is described in Section IV. Finally, conclusions and future work are summarized in Section V.

II. Related Work

Feature selection, as an effective method for handling high-dimensional data, has been extensively studied for a long time in the data mining and machine learning community. Generally, feature selection is divided into two categories, *wrapper*-based feature selection and *filter*-based feature selection. For wrapper-based techniques, the same classifier or inductive algorithm is used to both select the relevant features and execute the mining process [7]. Therefore, for a given data set, a wrapper-based technique may produce different feature subsets when using different learners. The potential problems of a wrapper-based technique lie in its high computational cost and a risk of overfitting to the model. In contrast, a filter-based technique is learner-independent. Once the data set is given, the filter-based technique will produce a feature subset (or feature subsets) that is (or are) correlated to the dependent attributes irrespective of which learner will be used after. In this study, the nine feature selection techniques used belong to the filter-based category.

Numerous variations of feature selection have been employed in a range of fields [1], [8], [9], [10], [11]. Jong et al. [8] introduced methods for feature selection based on support vector machines (SVM). Ilczuk et al. [9] investigated the importance of attribute selection in judging the qualification of patients for cardiac pacemaker implantation. In the context of text mining, Forman [10] investigated multiple filter-based feature ranking techniques. Rodríguez et al. [1] applied feature subset selection with three filter-based models and two wrapper-based models to five software engineering data sets. Chen et al. [11] have studied feature selection using wrappers in the context of software cost/effort estimation.

Class imbalance, which appears in various domains [12], [13], is another significant problem in data mining. One effective method for alleviating the adverse effect of skewed class distribution is sampling, which will add or remove instances from a data set until it becomes more balanced with respect to its class distribution [14], [15]. In this study, we used random undersampling due to its simplicity and effectiveness [15]. We will leave other sampling techniques as our future work.

While considerable work has been done for feature selection and data sampling separately, limited research can be found on investigating them both together, particularly in the software engineering field. Chen et al. [11] have studied data row pruning (data sampling) and data column pruning (feature selection) in the context of software cost/effort estimation. However, the data sampling in their study was not specific for the class imbalance problem, and unlike this study, the classification models are meant for non-binary problems. Moreover, they focused only on the case in which data sampling is used prior to feature selection.

A recent work of our research team investigated both feature selection and data sampling in the domain of software quality engineering [16]. The paper presented four approaches (scenarios) which include the three scenarios studied in this paper and the scenario where feature selection is used alone. In that paper, six standard filter-based feature ranking techniques were adopted. The conclusion was that data sampling performed prior to feature selection resulted in significantly better performance than data sampling performed after feature selection. The present study is an extension of the earlier work, exploring new ground in three ways: (1) We used nine different feature selection techniques from three different families instead of only six commonly used (standard) filter-based feature ranking techniques, which makes our conclusion more generalized; (2) In addition to considering the different scenarios, this study also looked at the different feature ranking techniques and examined their impact on the classification performance; and (3) The conclusion of this study is different than the previous one, possibly as a result of the aforementioned changes. We try to keep all the experimental settings the same as we did before to ensure effectiveness of the comparisons between the two studies.

III. Methodology

A. Feature Selection Techniques

In this paper, we investigate nine filter-based feature ranking techniques from three different families, including three standard methods, five threshold-based feature selection techniques, and the signal-to-noise approach. Following is an overview of each ranker family.

1) Standard Filter-Based Feature Rankers: Feature ranking assigns a score to each feature according to a particular method (metric), allowing the selection of the best set of features. Due to the space limitation, we only present three filter-based feature ranking techniques [17]: chi-square (CS), information gain (IG), and ReliefF (RF). More details are shown as follows.

i. The chi-square (CS) statistic, denoted as χ^2, is used to examine the distribution of the class as it relates to the values of the target feature. The null hypothesis is that there is no correlation. Given the null hypothesis, the χ^2 statistic measures how far away the actual value is from the expected value:

$$\chi^2 = \sum_{i=1}^{r} \sum_{j=1}^{n_c} \frac{(O_{i,j} - E_{i,j})^2}{E_{i,j}}$$

where r is the number of different values of the feature, n_c is the number of classes ($n_c = 2$ in this work), and $O_{i,j}$ and $E_{i,j}$ are the observed and expected number of instances with value i which are in class j, respectively. The larger the χ^2 statistic, the more likely the feature is relevant to the class.

ii. Information gain (IG) is a measure based on the concept of entropy from information theory [17]. IG is the information provided about the target class attribute Y, given the value of another attribute X. IG measures the decrease of the weighted average impurity of the partitions compared to the impurity of the complete set of data. IG tends to prefer attributes with a larger number of possible values.

iii. Relief is an instance-based feature ranking technique introduced by Kira and Rendell [18]. ReliefF (RF) is an extension of the Relief algorithm that can handle noise and multiclass data sets, and is implemented in the WEKA tool [17].

2) Threshold-Based Feature Rankers: The family of threshold-based feature selection (TBFS) techniques was developed and implemented by our research group within WEKA. A full discussion of this approach (although limited to cover only the AUC performance metric) can be found in [5]. In TBFS, each independent attribute is paired individually with the class attribute, and that two-attribute data set is evaluated using a number of different performance metrics. This feature ranking framework includes normalizing the attribute values (so that they fall between 0 and 1) and treating those values as the posterior probabilities from which to calculate performance metrics. In fact, this allows the use of performance metrics to describe how well the feature correlated with the class. Note that no classifiers were

built during the feature selection process. Five performance metrics used in TBFS are listed below.

i. Mutual Information (MI). Let $c(\mathbf{x})$ denote the actual class of instance $\mathbf{x}$, and let $\hat{c}^t(\mathbf{x})$ denote the predicted class based on the value of the attribute F^j and a given threshold t. Then MI is defined as:

$$\text{MI} = \max_{t\in[0,1]} \sum_{\hat{c}^t\in\{P,N\}} \sum_{c\in\{P,N\}} p(\hat{c}^t, c) \log \frac{p(\hat{c}^t, c)}{p(\hat{c}^t)p(c)}$$

where

$$p(\hat{c}^t = \alpha, c = \beta) = \frac{\mid \{\mathbf{x} \mid (\hat{c}^t(\mathbf{x}) = \alpha) \cap (c(\mathbf{x}) = \beta)\} \mid}{\mid P \mid + \mid N \mid},$$
$$p(\hat{c}^t = \alpha) = \frac{\mid \{\mathbf{x} \mid \hat{c}^t(\mathbf{x}) = \alpha\} \mid}{\mid P \mid + \mid N \mid},$$
$$p(c = \alpha) = \frac{\mid \{\mathbf{x} \mid c(\mathbf{x}) = \alpha\} \mid}{\mid P \mid + \mid N \mid},$$
$$\alpha, \beta \in \{P, N\}.$$

ii. Kolmogorov-Smirnov (KS) statistic measures the maximum difference between the cumulative distribution functions of examples in each class based on the normalized attribute $\widehat{F}^j$.

$$F_P(t) = \frac{\mid \{\mathbf{x} \in D \mid (\hat{F}^j(\mathbf{x}) \leq t) \cap (c(\mathbf{x}) = P)\} \mid}{\mid \{\mathbf{x} \in D \mid c(\mathbf{x}) = P\} \mid},$$
$$F_N(t) = \frac{\mid \{\mathbf{x} \in D \mid (\hat{F}^j(\mathbf{x}) \leq t) \cap (c(\mathbf{x}) = N)\} \mid}{\mid \{\mathbf{x} \in D \mid c(\mathbf{x}) = N\} \mid}.$$

KS is computed as

$$\text{KS} = \max_{t\in[0,1]} |F_P(t) - F_N(t)|.$$

With respect to KS, an attribute provides the best performance at a specific t value when the distance between the two distribution functions is maximized. The larger the KS value, the better the attribute is able to separate the two classes. The range of KS is between 0 and 1.

iii. Deviance (DV). For a given threshold t, define $v(\mathbf{x}) = 1$ if example $\mathbf{x}$ belongs to the positive class, otherwise, $v(\mathbf{x}) = 0$. Then DV is defined as:

$$\text{DV} = \min_{t\in[0,1]} \left[\sum_{\mathbf{x}\in S_t} \left(v(\mathbf{x}) - v(S^t)\right)^2 + \sum_{\mathbf{x}\in \bar{S}_t} \left(v(\mathbf{x}) - v(\bar{S}^t)\right)^2 \right]$$

where $v(S^t) = |S_t|^{-1} \sum_{\mathbf{x}\in S_t} v(\mathbf{x})$. In other words, the equation measures the sum of the squared errors from the mean class given a partitioning of the space based on each possible threshold t. The minimum value is preferred.

iv. Area Under the ROC Curve (AUC). The receiver operating characteristic [19], or ROC, curve graphs true positive rate on the y-axis versus the false positive rate on the x-axis. ROC curves are generated by varying the decision threshold t (between 0 and 1) used to transform the normalized attribute values into a predicted class. AUC is used to provide a single numerical metric for comparing the predictive power of each attribute. The range of AUC is between 0 and 1, and larger value is preferred.

v. Area Under the Precision-Recall Curve (PRC) is a single-value measure that originated from the area of information retrieval. A precision-recall curve is generated by varying the decision threshold t from 0 to 1 and plotting the recall (y-axis) and precision (x-axis) at each point in a similar manner to the ROC curve. The area under the PRC ranges from 0 to 1, and an attribute with more predictive power results in an area under the PRC closer to 1.

3) Signal-to-Noise: Signal to noise (S2N) [20], in the context of the feature ranking problem in data mining, defines how well a feature discriminates two classes in a two class problem. The equation to calculate S2N is

$$\text{S2N} = \frac{(\mu_P - \mu_N)}{\sigma_P + \sigma_N}$$

where μ_P and μ_N are mean values of a particular attribute for the samples from class P and class N, and σ_P and σ_N are standard deviations of this attribute from the sample set for class P and class N. Because of its discriminating power among the classes, S2N is highly efficient to properly order the features in terms of their relation to the output class. But this technique has not been used as often for feature selection.

B. Data Sampling Techniques

A number of data sampling techniques have been studied in the literature, including both majority undersampling and minority oversampling techniques [14], [15]. We consider random undersampling (RUS) as the data sampling technique in this study. Random undersampling is a simple, yet effective, data sampling technique that achieves more balance in a given data set by randomly removing instances from the majority (*nfp*) class. The post-sampling class distribution is a parameter for any data sampling technique. In our study, the post-sampling distribution of the *nfp* and *fp* instances is 65% and 35% respectively. Other settings such as 50:50 were also considered, but those results are not presented due to similarity of conclusions.

C. Classification Performance Metric

The effectiveness of each approach is assessed by evaluating the classification performance of the models subsequently trained and tested with that particular approach. In our experiments, we use AUC as the classification performance metric. Our selection of AUC is based on one of its characteristics, namely its invariance to a priori class probability distributions. AUC does not emphasize one class over the other as may be the case in some other performance metrics, thus it is not biased against the positive (*fp*) class. Given the imbalanced nature of our data sets (see in later section), AUC is an appropriate measure for comparing the classification performance of the learners. In fact, AUC serves as an aid to both feature ranking and final classification evaluation in this study.

D. Classifier

In this study, the software quality prediction models are built with support vector machine (SVM) [21]. This learner was selected because of its common use in the software engineering domain and data mining, and also because it does not have a built-in feature selection capability. SVM, also called SMO in WEKA, had two changes to the default parameters: the `complexity constant c` was set to '5.0' and `build Logistic Models` was set to 'true'. By default, a linear kernel was used.

IV. A Case Study

A. Data Sets

In our experiments, we use publicly available data, namely the Eclipse defect counts and complexity metrics data set obtained from the PROMISE data repository [6]. In particular, we use the metrics and defects data at the software packages level. The original data for the Eclipse packages consists of three releases denoted 2.0, 2.1,

TABLE I
DATA CHARACTERISTICS

Data#	Rel.	thd	#Attr.	#Inst.	#fp	%fp	#nfp	%nfp
1	2.0	10	209	377	23	6.1	354	93.9
2	2.0	5	209	377	52	13.8	325	86.2
3	2.0	3	209	377	101	26.8	276	73.2
4	2.1	5	209	434	34	7.8	400	92.2
5	2.1	4	209	434	50	11.5	384	88.5
6	2.1	2	209	434	125	28.8	309	71.2
7	3.0	10	209	661	41	6.2	620	93.8
8	3.0	5	209	661	98	14.8	563	85.2
9	3.0	3	209	661	157	23.8	504	76.2

and 3.0 respectively. Each release as reported by Zimmermann et al. [22] contains the following information: the name of the package for which the metrics are collected (name), the number of defects reported six months prior to release (pre-release defects), the number of defects reported six months after release (post-release defects), a set of complexity metrics computed for classes or methods and aggregated in terms of average, maximum, and total (complexity metrics), and the abstract syntax tree of the package consisting of the node size, type, and frequency (structure of abstract syntax tree(s)). For our study we transform the original data by: (1) removing all non-numeric attributes, including the package names, and (2) converting the post-release defects attribute to a binary class attribute with fault-prone (*fp*) being the minority class and not-fault-prone (*nfp*), the majority class. Membership in each class is determined by a post-release defects threshold thd, which separates *fp* from *nfp* packages by classifying packages with thd or more post-release defects as *fp* and the remaining as *nfp*. In our study, we use thd = {10, 5, 3} for releases 2.0 and 3.0 while we use thd = {5, 4, 2} for release 2.1. This results in three groups. Each group contains three data sets, one for each release. The reason why a different set of thresholds is chosen for release 2.1 is that we would like to keep similar class distributions for the data sets in the same group. All data sets contain 209 attributes (208 independent attributes and 1 dependent attribute). Table I shows the characteristics of the data sets after transformation for each group. These data sets exhibit different distribution of class skew (i.e., the percentage of *fp* examples).

B. Design

The primary objective of the experiments is to evaluate the effectiveness of feature selection techniques when combined with data sampling. Different scenarios may be produced depending on whether sampling is performed prior to or after feature selection and which data set, sampled or unsampled data, is used to build a classifier. The three different scenarios examined are described as follows:

- **Scenario 0**: Data sampling is performed **before** feature selection and the selected features are applied to the **unsampled** (original) data to form the training data set.
- **Scenario 1**: Data sampling is performed **before** feature selection and the selected features are applied to the **sampled** data to form the training data set.
- **Scenario 2**: Data sampling is performed **after** feature selection and the selected features are applied to the **sampled** data to form the training data set.

C. Results & Analysis

The experiments were performed on the three groups of Eclipse data sets. Nine feature ranking techniques from three different families were used to rank the attributes according to their respective scores. Then, we selected $\lceil \log_2 n \rceil$ attributes that had the highest scores, where n is the number of the independent attributes in the original data set. In this study, $n = 208$, so $\lceil \log_2 n \rceil = 8$. We choose $\lceil \log_2 n \rceil$ attributes because 1) related literature does not provide guidance on the appropriate number of features to select; and 2) one of our recent empirical studies [23] showed that it was appropriate to use $\lceil \log_2 n \rceil$ features when using WEKA to build random forests learners for binary classification in general and imbalanced data sets in particular. Although we used a different learner here, a preliminary study showed that $\lceil \log_2 n \rceil$ is still appropriate for various learners.

After feature selection and data sampling were implemented in the different scenarios, we built SVM models using the training data sets with the selected attributes, and we used AUC to evaluate the performance of the classifications. All results are reported in Table II. In the experiments, ten runs of five-fold cross-validation were performed. The values in the table represent the average AUC for the classification models constructed over the ten runs of five-fold cross-validation. For each data set, the average performance (Avg. column) for each feature selection method across three different scenarios, and the average performance (Avg. row) of each scenario over nine feature selection techniques are also presented. The best feature selection technique (based on their average performance) for each data set is highlighted with ***underline***, and the best scenario (based on their average performance) is highlighted with **bold**. The results demonstrate that the classification models have better performance when using AUC, PRC, IG and MI feature selection techniques and also that scenario 0 shows better performance than the other two scenarios.

We also conducted a two-way ANalysis Of VAriance (ANOVA) F test on the classification performance over the nine data sets to examine whether the performance difference (better/worse) is statistically significant or not. The two factors considered in the test are: Factor A, representing three scenarios, and Factor B, representing nine feature ranking techniques. The null hypothesis for the ANOVA test is that all the group population means are the same and the alternate hypothesis is that at least one pair of means is different. In addition, the interaction between Factor A and Factor B is also taken into account in the test. Table III shows the ANOVA results. The p-value is zero or close to zero for each main factor (Factor A and Factor B). This means that at least two scenarios performed significantly differently from each other and at least two feature selection techniques present significantly different performances. The p-value of the interaction term is greater than the typical cutoff 0.05, meaning that the interaction does not significantly affect classification performance. In other words, changing the value of Factor A will not significantly influence the value of Factor B, and vice versa.

We further carried out a multiple comparison test [24] on each main factor and their interaction with Tukey's honestly significant difference criterion. Figure 1, including three subfigures, shows the multiple comparisons for Factor A, Factor B, and interaction A×B, respectively. The figures display graphs with each group mean represented by a symbol (○) and 95% confidence interval as a line around the symbol. Two means are significantly different if their intervals are disjoint, and are not significantly different if their intervals overlap. Matlab was used to construct the ANOVA models and perform the multiple comparisons presented in this work, and the assumptions for constructing ANOVA models were validated. From these figures, we can see the following points:

- Of the three scenarios, Scenario 0 performs significantly better than the other two scenarios, followed by Scenario 2, then Scenario 1.
- Among the nine feature selection methods, AUC performs

TABLE II
CLASSIFICATION PERFORMANCE OF SVM IN TERMS OF AUC

Data#	Filter	S0	S1	S2	Avg.	Data#	Filter	S0	S1	S2	Avg.	Data#	Filter	S0	S1	S2	Avg.
	CS	0.8662	0.8567	0.8409	0.8546		CS	0.9174	0.9111	0.9034	0.9106		CS	0.9292	0.9248	0.9253	0.9264
	IG	0.8665	0.8555	0.8587	0.8602		IG	0.9199	0.9135	0.9137	0.9157		IG	0.9325	0.9290	0.9320	0.9312
	RF	0.8753	0.8602	0.8619	0.8658		RF	0.8484	0.8462	0.8463	0.8470		RF	0.8925	0.8914	0.8766	0.8868
	MI	0.8564	0.8436	0.8675	0.8558		MI	0.9160	0.9151	0.9228	0.9179		MI	0.9277	0.9218	0.9204	0.9233
1	KS	0.8471	0.8347	0.8268	0.8362	4	KS	0.9178	0.9128	0.9223	0.9176	7	KS	0.9249	0.9204	0.9186	0.9213
	DV	0.8656	0.8565	0.8613	0.8611		DV	0.9179	0.9106	0.9174	0.9153		DV	0.9232	0.9201	0.9307	0.9247
	AUC	0.8817	0.8641	0.8813	***0.8757***		AUC	0.9197	0.9144	0.9236	***0.9193***		AUC	0.9338	0.9286	0.9322	***0.9315***
	PRC	0.8712	0.8638	0.8624	0.8658		PRC	0.9132	0.9134	0.9134	0.9133		PRC	0.9300	0.9245	0.9315	0.9287
	S2N	0.8757	0.8652	0.8673	0.8694		S2N	0.9037	0.8962	0.8979	0.8993		S2N	0.9161	0.9179	0.9214	0.9185
	Avg	**0.8673**	0.8556	0.8587			Avg	**0.9082**	0.9037	0.9067			Avg	**0.9233**	0.9198	0.9209	
	CS	0.9092	0.9022	0.9080	0.9064		CS	0.9028	0.9054	0.9028	0.9037		CS	0.9409	0.9334	0.9352	0.9365
	IG	0.9132	0.9036	0.9144	0.9104		IG	0.9034	0.9066	0.9056	***0.9052***		IG	0.9408	0.9340	0.9338	0.9362
	RF	0.9078	0.8973	0.8862	0.8971		RF	0.8436	0.8518	0.8268	0.8407		RF	0.9156	0.9008	0.8893	0.9019
	MI	0.9124	0.9088	0.9128	***0.9113***		MI	0.9003	0.9027	0.9030	0.9020		MI	0.9405	0.9321	0.9338	0.9355
2	KS	0.9012	0.8964	0.9098	0.9025	5	KS	0.8998	0.9027	0.9001	0.9009	8	KS	0.9402	0.9312	0.9327	0.9347
	DV	0.9007	0.8926	0.9051	0.8995		DV	0.9015	0.9022	0.9012	0.9016		DV	0.9396	0.9304	0.9360	0.9353
	AUC	0.9134	0.9039	0.9137	0.9103		AUC	0.9007	0.9033	0.9048	0.9029		AUC	0.9402	0.9329	0.9335	0.9355
	PRC	0.9160	0.9048	0.9052	0.9087		PRC	0.9026	0.9042	0.9057	0.9042		PRC	0.9412	0.9341	0.9356	***0.9370***
	S2N	0.9088	0.8979	0.8949	0.9005		S2N	0.8908	0.8914	0.8906	0.8909		S2N	0.9379	0.9255	0.9272	0.9302
	Avg	**0.9092**	0.9008	0.9056			Avg	0.8939	**0.8967**	0.8934			Avg	**0.9374**	0.9283	0.9286	
	CS	0.8636	0.8611	0.8604	0.8617		CS	0.8885	0.8862	0.8862	0.8870		CS	0.9061	0.8999	0.9003	0.9021
	IG	0.8666	0.8645	0.8643	0.8651		IG	0.8881	0.8862	0.8867	0.8870		IG	0.9063	0.9004	0.9002	0.9023
	RF	0.8351	0.8293	0.8303	0.8315		RF	0.8728	0.8740	0.8608	0.8692		RF	0.8717	0.8625	0.8217	0.8519
	MI	0.8521	0.8499	0.8594	0.8538		MI	0.8894	0.8872	0.8862	***0.8876***		MI	0.9060	0.9014	0.9016	0.9030
3	KS	0.8550	0.8530	0.8593	0.8557	6	KS	0.8891	0.8866	0.8861	0.8873	9	KS	0.9058	0.9013	0.9013	0.9028
	DV	0.8513	0.8487	0.8568	0.8522		DV	0.8878	0.8850	0.8837	0.8855		DV	0.9066	0.9014	0.9016	0.9032
	AUC	0.8653	0.8647	0.8663	***0.8654***		AUC	0.8883	0.8865	0.8876	0.8875		AUC	0.9058	0.8999	0.8997	0.9018
	PRC	0.8603	0.8593	0.8594	0.8597		PRC	0.8887	0.8853	0.8864	0.8868		PRC	0.9064	0.9006	0.9001	0.9024
	S2N	0.8543	0.8531	0.8540	0.8538		S2N	0.8838	0.8813	0.8857	0.8836		S2N	0.9123	0.9054	0.9055	***0.9077***
	Avg	0.8559	0.8537	**0.8567**			Avg	**0.8863**	0.8843	0.8833			Avg	**0.9030**	0.8970	0.8924	

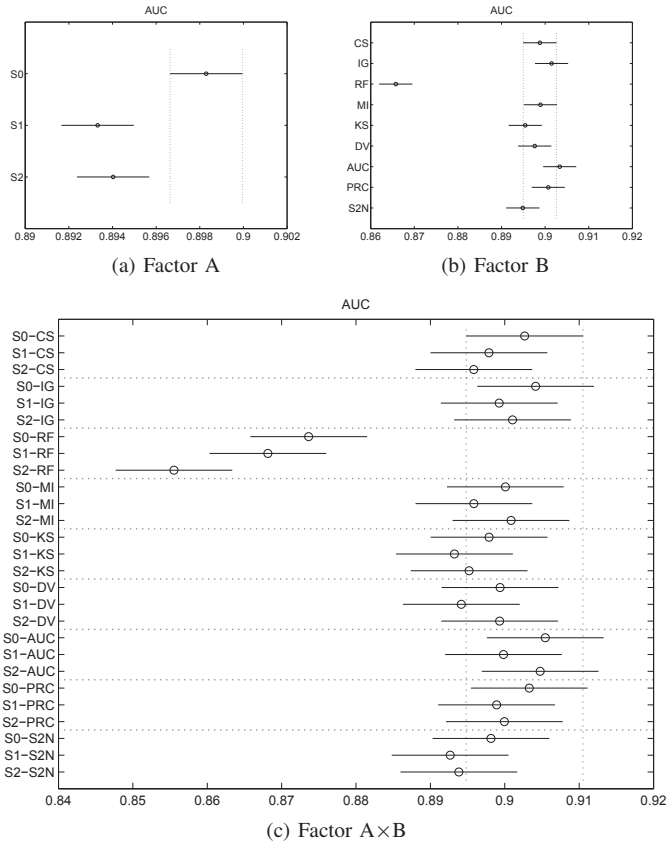
TABLE III
TWO-WAY ANOVA TABLE

Source	Sum Sq.	d.f.	Mean Sq.	F	p-value
A	0.0117	2	0.0059	7.23	0.0007
B	0.2794	8	0.0349	43.05	0.0000
A×B	0.0152	16	0.0009	1.17	0.2864
Error	1.9491	2403	0.0008		
Total	2.2553	2429			

best, followed by IG, PRC, MI, CS, and DV, then KS and S2N. RF performs significantly worse than others. The nine techniques ordered in terms of their performance from best to worst are: AUC^A, IG^{AB}, PRC^{AB}, MI^{AB}, CS^{AB}, DV^{AB}, KS^B, S2N^B, RF^C. The methods labeled with the same superscript are from the same performance group, in which no statistically significant difference is found between the methods, while the methods labeled with different superscripts come from different performance groups, in which statistically significant differences are found between the methods.

- There are 27 groups for interaction A×B; these are formed by each of nine feature selection techniques being combined with three scenarios. The group means demonstrate that the classification performance is not heavily influenced by these interactions. Among 27 groups, S0-AUC (classification models built in Scenario 0 and using the AUC feature selection technique) demonstrates better average performance than the other groups over the nine data sets. RF-based groups perform worst.

In addition to the primary experiment discussed above, we also compared defect prediction models built on smaller subsets of attributes to those built with a complete set of attributes (no feature selection or sampling used at all). Table IV shows the classification



(a) Factor A (b) Factor B

(c) Factor A×B

Fig. 1. Multiple Comparisons

TABLE IV
CLASSIFICATION PERFORMANCE OF SVM ON FULL DATA SETS

Data#	AUC	Data#	AUC	Data#	AUC
1	0.8064	4	0.7988	7	0.8030
2	0.8904	5	0.8039	8	0.8897
3	0.8235	6	0.8154	9	0.8673

performances of the SVM learner using the original data with 208 software metrics. Each value presented in the table is the average AUC over the ten runs of five-fold cross-validation outcomes. The results demonstrate that all three strategies (scenarios) perform better or significantly better than the case where no feature selection or sampling is used at all, except for the 9th data set when RF is used.

V. CONCLUSION

In the context of software defect prediction, two key problems often faced by software practitioners are the presence of excessive metrics in training data sets and a relatively small proportion of fault-prone modules to learn from.

In order to overcome the problems, we studied a process using feature selection and data sampling together to modify the training data to improve software defect prediction models. Three scenarios are investigated: (1) feature selection based on sampled data, and training data based on original data; (2) feature selection based on sampled data, and training data based on sampled data; and (3) feature selection based on original data, and training data based on sampled data. The main objective of this paper is to examine and compare all three scenarios and assess their effectiveness in the context of software quality prediction. We used nine different filter-based feature ranking techniques to select the software metrics and random undersampling to deal with class imbalance. The SVM classifier was employed to build classification models. The experiments were performed on nine software measurement data sets obtained from the PROMISE repository. The results show that sampling performed prior to feature selection and training data based on original data resulted in significantly better performance than the other two approaches (scenarios). In addition, among the nine filter-based feature ranking techniques, AUC performed better than other methods. Moreover, all three strategies performed significantly better than the case where no feature selection or sampling is used. The above result is of particular importance to a software quality analyst, since a useful model can be built using only selected software metrics. This provides a less cumbersome and more insightful model for analyzing the software quality trends of the target project, as compared to analyzing the model with respect to a large set of metrics.

Future work will include experiments using data sets from different software projects. Various sampling techniques, learners, and performance metrics will also be examined in future work.

REFERENCES

[1] D. Rodriguez, R. Ruiz, J. Cuadrado-Gallego, and J. Aguilar-Ruiz, "Detecting fault modules applying feature selection to classifiers," in *Proceedings of 8th IEEE International Conference on Information Reuse and Integration*, Las Vegas, Nevada, August 13-15 2007, pp. 667–672.

[2] H. Wang, T. M. Khoshgoftaar, K. Gao, and N. Seliya, "Mining data from multiple software development projects," in *Proceedings of the 3rd IEEE International Workshop Mining Multiple Information Sources*, Miami, FL, Dec. 6 2009, pp. 551–557.

[3] K. Gao, T. M. Khoshgoftaar, H. Wang, and N. Seliya, "Choosing software metrics for defect prediction: an investigation on feature selection techniques," *Software: Practice and Experience. Special Issue: Practical Aspects of Search-Based Software Engineering*, vol. 41, no. 5, pp. 579–606, April 2011.

[4] J. Van Hulse, T. M. Khoshgoftaar, and A. Napolitano, "Experimental perspectives on learning from imbalanced data," in *Proceedings of the 24th International Conference on Machine Learning*, Corvallis, OR, USA, June 2007, pp. 935–942.

[5] T. M. Khoshgoftaar and K. Gao, "A novel software metric selection technique using the area under roc curves," in *Proceedings of the 22nd International Conference on Software Engineering and Knowledge Engineering*, San Francisco, CA, July 1-3 2010, pp. 203–208.

[6] G. Boetticher, T. Menzies, and T. Ostrand. (2007) Promise repository of empirical software engineering data. [Online]. Available: http://promisedata.org/

[7] G. H. John, R. Kohavi, and K. Pfleger, "Irrelevant features and the subset selection problem," in *Proceedings of the 11th International Conference on Machine Learning*, 1994, pp. 121–129.

[8] K. Jong, E. Marchiori, M. Sebag, and A. van der Vaart, "Feature selection in proteomic pattern data with support vector machines," in *Proceedings of the 2004 IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology*, Oct 7-8 2004.

[9] G. Ilczuk, R. Mlynarski, W. Kargul, and A. Wakulicz-Deja, "New feature selection methods for qualification of the patients for cardiac pacemaker implantation," *Computers in Cardiology*, vol. 34, no. 2-3, pp. 423–426, 2007.

[10] G. Forman, "An extensive empirical study of feature selection metrics for text classification," *Journal of Machine Learning Research*, vol. 3, pp. 1289–1305, March 2003.

[11] Z. Chen, T. Menzies, D. Port, and B. Boehm, "Finding the right data for software cost modeling," *IEEE Software*, vol. 22, no. 6, pp. 38–46, November 2005.

[12] V. Engen, J. Vincent, and K. Phalp, "Enhancing network based intrusion detection for imbalanced data," *International Journal of Knowledge-Based and Intelligent Engineering Systems*, vol. 12, no. 5-6, pp. 357–367, 2008.

[13] A. H. Kamal, X. Zhu, A. S. Pandya, S. Hsu, and M. Shoaib, "The impact of gene selection on imbalanced microarray expression data," in *Proceedings of the 1st International Conference on Bioinformatics and Computational Biology; Lecture Notes in Bioinformatics; Vol. 5462*, New Orleans, LA, 2009, pp. 259–269.

[14] N. V. Chawla, K. W. Bowyer, L. O. Hall, and P. W. Kegelmeyer, "SMOTE: Synthetic minority over-sampling technique," *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, 2002.

[15] C. Seiffert, T. Khoshgoftaar, and J. Van Hulse, "Improving software-quality predictions with data sampling and boosting," *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, vol. 39, no. 6, pp. 1283–1294, Nov. 2009.

[16] T. M. Khoshgoftaar, K. Gao, and N. Seliya, "Attribute selection and imbalanced data: Problems in software defect prediction," in *Proceedings of 22nd IEEE International Conference on Tools with Artificial Intelligence*, Arras, France, October 27-29 2010, pp. 137–144.

[17] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd ed. Morgan Kaufmann, 2005.

[18] K. Kira and L. A. Rendell, "A practical approach to feature selection," in *Proceedings of 9th International Workshop on Machine Learning*, 1992, pp. 249–256.

[19] F. Provost and T. Fawcett, "Robust classification for imprecise environments," *Machine Learning*, vol. 42, pp. 203–231, 2001.

[20] L. Goh, Q. Song, and N. Kasabov, "A novel feature selection method to improve classification of gene expression data," in *Proceedings of the Second Conference on Asia-Pacific Bioinformatics*, Dunedin, New Zealand, 2004, pp. 161–166.

[21] J. Shawe-Taylor and N. Cristianini, *Support Vector Machines*, 2nd ed. Cambridge University Press, 2000.

[22] T. Zimmermann, R. Premraj, and A. Zeller, "Predicting defects for eclipse," in *Proceedings of the 29th International Conference on Software Engineering Workshops*. Washington, DC, USA: IEEE Computer Society, 2007, p. 76.

[23] T. M. Khoshgoftaar, M. Golawala, and J. Van Hulse, "An empirical study of learning from imbalanced data using random forest," in *Proceedings of the 19th IEEE International Conference on Tools with Artificial Intelligence*, vol. 2, Washington, DC, USA, 2007, pp. 310–317.

[24] M. L. Berenson, M. Goldstein, and D. Levine, *Intermediate Statistical Methods and Applications: A Computer Package Approach*, 2nd ed. Englewood Cliffs, NJ: Prentice-Hall, 1983.

BUGMINER: Software Reliability Analysis Via Data Mining of Bug Reports

Leon Wu Boyi Xie Gail Kaiser Rebecca Passonneau
Department of Computer Science
Columbia University
New York, NY 10027 USA
{leon,xie,kaiser,becky}@cs.columbia.edu

Abstract

Software bugs reported by human users and automatic error reporting software are often stored in some bug tracking tools (e.g., Bugzilla and Debbugs). These accumulated bug reports may contain valuable information that could be used to improve the quality of the bug reporting, reduce the quality assurance effort and cost, analyze software reliability, and predict future bug report trend. In this paper, we present BUGMINER*, a tool that is able to derive useful information from historic bug report database using data mining, use these information to do completion check and redundancy check on a new or given bug report, and to estimate the bug report trend using statistical analysis. Our empirical studies of the tool using several real-world bug report repositories show that it is effective, easy to implement, and has relatively high accuracy despite low quality data.*

1 Introduction

Finding and fixing the faults in a software is an indispensable while time-consuming quality assurance task in software development. Our definition of *fault* is a programming error that leads to an erroneous result in some programs during execution. A *software bug* is the common term used to describe a fault, error, flaw, mistake, or failure in a program that produces an incorrect or unexpected result, or causes it to behave in unintended ways. When a software bug is identified, it is often reported and recorded into a bug report database using some bug tracking tools so that further analysis or fix can be performed, possibly by a developer or tester. For some real-world software, their bug report databases have accumulated a large amount of historic bug reports. For example, as of February 2011, Debbugs, *i.e.*, Debian bug tracking system, has accumulated 615,000 bug reports [4, 3].

These accumulated bug reports may contain valuable information that could be used to improve the quality of the bug reporting, reduce the cost of quality assurance, analyze software reliability, and predict future bug report trend. One of the challenges in bug reporting is that the bug reports are often incomplete (*e.g.*, missing data fields such as product version or operating system details). Another challenge is that there are often many duplicate bug reports for the same bug. Software developers or testers normally have to review these redundant bug reports manually, which is time-consuming and cost inefficient.

We developed a tool named BUGMINER that is able to derive useful information from historic bug reports using data mining techniques, including machine learning (*e.g.*, SVM [15, 5]) and natural language processing, and use these information to do completion check through classification and redundancy check through similarity ranking on a new or given bug report. BUGMINER can also perform bug report trend analysis using Weibull distribution [13]. We implemented the tool and experimented it using three real-world bug report repositories including Apache Tomcat [1], Eclipse [7], and Linux Kernel [11]. Our experiments demonstrate that it is effective, easy to implement, and has relatively high accuracy despite low quality data.

The rest of the paper is organized as follows. In the following section, we give background information on bug reporting. In Section 3, we present the details of our approach, followed by our empirical studies in Section 4. Lastly, we compare related work in Section 5, before we conclude in Section 6.

2 Background on Bug Reporting

Bug tracking tools are often developed as a database-driven web application. The web interface allows multiple geographically distributed users to enter the bug reports simultaneously. The backend database stores the records for the reported bugs. Table 1 lists some main attributes (*i.e.*, data fields or columns) of a typical bug report for Apache Tomcat using Bugzilla [1, 2]. These attributes are meta in-

formation of the bug report. The field bug_id is an unique identifier for a distinct bug instance. A bug report is often modified by subsequent reviewers or developers who are trying to verify or fix the bug. Table 2 lists the additional commentary entries related to the same bug listed in Table 1. Each new entry (*i.e.*, new long_desc record) records the author name, entry date and time, and the free text description. The entry date and time for the first and last long_desc record, along with the first author's name, are also stored in the main attributes list of the same bug (*i.e.*, creation_ts, delta_ts, and reporter). There is no predefined limit on how many additional commentary entries a bug report can hold. Bug report datasets will be further explained in Section 4.2.

Table 1. Main attributes of a bug report

Attribute Name	Sample Value	Attribute Name	Sample Value
bug_id	48892	component	Connectors
creation_ts	2010-03-11 12:10:09 -0500	delta_ts	2010-12-14 14:30:22 -0500
short_desc	Use URIEncoding...	rep_platform	All
cclist_accessible	1	op_sys	All
classification_id	1	bug_status	NEW
classification	Unclassified	bug_severity	enhancement
product	Tomcat 7	priority	P2
reporter	reporter 1	assigned_to	dev

Table 2. Additional attributes

Attribute Name	long_desc 1	long_desc 2	long_desc 3
isprivate	0	0	0
who	reporter 1	reporter 2	reporter 3
bug_when	2010-03-11 12:10:09 -0500	2010-04-04 10:18:48 -0400	2010-12-14 14:30:22 -0500
thetext	Here is a ...	There are ...	For encoding ...

3 Approach

3.1 Architecture

Figure 1 illustrates the architecture of BUGMINER. There are two types of bug reporters: human users such as software developers, testers, and end users; automatic error reporting processes that run as a service on users' computers. The bug reporters generate new bug reports and enter the related information via the bug tracking tool's interface. The bug tracking tool then store the new bug report into the bug report database.

BUGMINER consists of three data mining and statistical processing engines: automatic completion check engine; automatic redundancy check engine; and bug report trend analysis engine. These three engines process the historic data stored in the bug report database and the new bug report coming in. The results from these engines are then directed to the bug tracking tool so that these results can be reviewed and stored. In the following subsections, we will describe each engine in detail.

3.2 Attributes and Feature Selection

BUGMINER analyzes bug report data based on two sets of attributes: 1) static meta information, and 2) bag-of-words (*i.e.*, a collection of distinct free text words) attributes. For each bug report in Bugzilla, users need to fill in a predefined set of bug information, as shown in Table 1. This set of attributes has two characteristics: 1) static: the list of fields is fixed for all types of software products, and those fields are available for all bug reports; 2) meta information: they describe the general information about the bug report but doesn't go to the details of the problem. Bug report analysis based solely on the static and meta information is very limited. In BUGMINER, we further include the free text data of a bug report in our analysis.

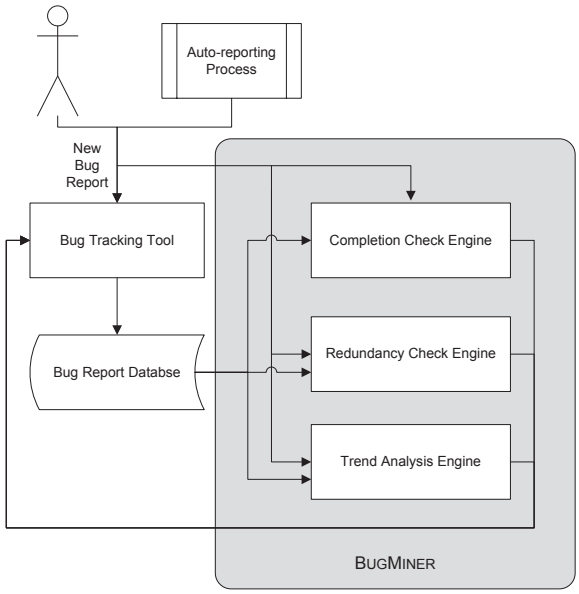


Figure 1. BUGMINER **architecture**

The free text data usually describes a bug scenario in natural language followed by some sample code. We represent the textual data as a bag-of-words. Each data instance is a high dimensional vector with words being attributes. The values of the attributes are Term Frequency-Inverse Document Frequency (TF-IDF) weight [12], which gives a higher weight on words that are frequent in certain data records but not too common across all records. Stemming, a process of reducing inflected (or sometimes derived) words to their stem or root form, is not necessary because the vocabulary usually doesn't have a variety of morphed forms, and imperfect stemming may bring in additional noisy content unnecessarily. Our feature selection also bases on inverse document frequency (IDF) and global term frequency. Words with a low IDF (*e.g.*, stopwords such as 'the' and 'a') are removed because they are too common and lack discriminative power. Words with a very low global term frequency are also removed because they are rare and their inclusion leads to a high dimensionality, which may cause "curse of dimensionality" problem in machine learning.

3.3 Automatic Completion Check Engine

When a bug report is filed, the bug information submitted are sometimes incomplete (*e.g.*, missing data fields). BUGMINER's automatic completion check engine derives these missing data fields through mining historic data and classification using the partially filled information. It reduces manual effort, keeps the bug report complete, and helps developers and testers to analyze the bug report more precisely.

3.3.1 Classification Using Support Vector Machine

Missing field autocompletion can be solved as a supervised learning problem. By training a classification model on existing data, we can predict the missing values. In BUGMINER, we use Support Vector Machines (SVM) as the classifier. SVM is a popular machine learning method because of its high performance. It formulates the classification modeling process as a quadratic minimization problem, and finds hyperplanes in a high dimensional space that separate data instances of different categories, while maximizing the margins between categories.

We first use a set of historic bug reports (*e.g.*, each one with n attributes) as training data to build a linear SVM model. For a new or given bug report with one missing data field a (*i.e.*, $n-1$ attributes filled and 1 attribute missing), we use the trained SVM model as a classifier and the filled $n-1$ attributes to predict the value of the missing a field for this bug report. In the case of multiple data fields are missing for a report (*e.g.*, $n-m$ attributes filled and m attributes missing), we use the SVM model and the $n-m$ filled attributes to predict the missing fields one by one.

3.4 Automatic Redundancy Check Engine

A common way of searching a bug report database to find out whether a new or given bug report already exists or not is to use keyword search, which normally uses keyword in combination with some wildcat characters such as '%' and '?' to construct database query string that can be executed on the database table. This kind of search based on keyword matching is often imprecise and may generate a large amount of useless or irrelevant results. The similarity ranking used by BUGMINER's automatic redundancy check engine is able to tell whether the new bug report is a duplicate or not more precisely. Furthermore, the similarity ranking can find out the most similar prior bug reports and sort them for the user.

3.4.1 Similarity Ranking Using Cosine Similarity

We represent bug report dataset in a vector space model (*i.e.*, term vector model), an algebraic model for representing text documents as vectors of identifiers, such as index terms [14]. Each bug report is a vector that consists of a list of feature values. As described in Section 3.2, BUGMINER uses two sets of features: 1) static meta information; 2) bag-of-words attributes with TF-IDF values.

We measure the similarity between two bug reports based on Cosine similarity, *i.e.*, the Cosine of the angle between the two vectors that represent these two bug reports, as shown in the following formula:

$$Distance_{COS}(a,b) = \frac{\sum_i a_i \times b_i}{\sqrt{\sum_i a_i^2} \times \sqrt{\sum_i b_i^2}},$$

where a and b represent two vectors. Its result equals 1 when the angle between two vectors is 0 (*i.e.*, two vectors are pointing in the same direction), and its result is less than 1 otherwise.

For a new or given bug report, we compute the Cosine similarity value (*i.e.*, csv) between this new bug report's vector and all the prior bug reports' vectors, and then rank the csv values in an descending order. The historic bug report with the highest csv value (*i.e.*, the closest one to 1) is the most similar prior record.

3.4.2 Similarity Ranking Using KL Divergence

In addition to Cosine similarity, we rank all prior bug reports based on their relevance to the new bug report using probability distribution. Kullback-Leibler (*i.e.*, KL) divergence [6, 12] is an effective relevance metric that assumes each data instance in a high dimensional feature space is characterized by a probability distribution. KL divergence measures the dissimilarity between two probability distributions, as shown in the following formular:

$$D_{KL}(a||b) = \sum_{t \in V} P(t|M_a) log \frac{P(t|M_a)}{P(t|M_b)},$$

where M_a and M_b represent the probability distributions for vector a and b respectively. V is the vocabulary of all terms and t is a term in V. KL divergence measures how bad the probability distribution M_a is at modeling M_b. Previous work [19] presents results suggesting that model comparison approach outperforms both query-likelihood and document-likelihood approaches. However, this metric is asymmetric, *i.e.*, $D_{KL}(a||b) \neq D_{KL}(b||a)$. In order to use it as a distance metric, we adopt a symmetrized KL divergence method for similarity ranking, which is defined as:

$$Distance_{KL}(a,b) = \frac{1}{2} D_{KL}(a||b) + \frac{1}{2} D_{KL}(b||a).$$

The result is symmetric and nonnegative. It equals 0 when two distributions are identical. It is bigger than 0 otherwise, and the larger the value the greater their dissimilarity.

For a new or given bug report, we compute the symmetrized KL divergence value (*i.e.*, kld) between this new bug report's vector and all the prior bug reports' vectors,

and then rank the kld values in an ascending order. The historic bug report with the lowest kld value (*i.e.*, the closest one to 0) is the most similar prior record.

3.4.3 Is the New Bug Report a Duplicate? What are the Similar Bugs Reported before?

We categorize a new or given bug report into one of the three categories according to the ranked csv and kdl values, along with the value ranges they fall into:

- If a prior report exists with $csv \geq c\_r_2$ and $kld \leq k\_r_1$, it is highly likely to be a duplicate (or repeat) of a prior report.
- If a prior report exists with $c\_r_1 < csv < c\_r_2$ or $k\_r_1 < kld < k\_r_2$, it has similar prior report.
- If all prior reports have $csv \leq c\_r_1$ and $kld \geq k\_r_2$, it does not have any similar prior report.

The value range parameters (*i.e.*, c_r_1, c_r_2, k_r_1, and k_r_2) can be determined based on heuristics obtained from experiments.

3.5 Bug Report Trend Analysis Engine

After major software releases, the number of software bugs tend to increase initially. As these bugs are fixed, the number of bugs gradually decreases, which resembles the "bathtub curve" in reliability engineering. The increase and descrease of the number of bugs normally lead to the similar trend of the number of bug reports. Weibull distribution can be used to model this kind of pattern and provide the basis for trend analysis.

3.5.1 Report Incidence Distribution

For the Weibull distribution, the incidence (*e.g.*, failure or bug report) density function $f(t)$ and cumulative incidence distribution function $F(t)$ are

$$f(t;\lambda,k) = \frac{k}{\lambda}(\frac{t}{\lambda})^{k-1}e^{-(t/\lambda)^k}, \quad t \geq 0,$$

$$F(t;\lambda,k) = 1 - e^{-(t/\lambda)^k}, \quad t \geq 0,$$

where $k > 0$ is the shape parameter and $\lambda > 0$ is the scale parameter of the distribution. The instantaneous incidence rate (or hazard function) when $t \geq 0$ can be derived as

$$h(t;\lambda,k) = \frac{f(t;\lambda,k)}{1 - F(t;\lambda,k)} = \frac{k}{\lambda}\left(\frac{t}{\lambda}\right)^{k-1}.$$

A value of $k < 1$ indicates that the incidence rate decreases over time. A value of $k = 1$ indicates that the incidence rate is constant (*i.e.*, k/λ) over time. In this case, the Weibull distribution becomes an exponential distribution. A value of $k > 1$ indicates that the incidence rate increases with time.

3.5.2 Estimation of Coming Bug Report

We first use historic data to fit the Weibull function and derive the λ and k parameters. Then for any given time t, which is the number of weeks (or other chosen time units such as days or hours) after the starting date, the number of bug reports that may happen during that week can be estimated using the Weibull's density function $f(t)$. The result is an estimate of how many bug reports may happen during the t-th week after the starting event, *e.g.*, a new software release. Similarily, the instantaneous incidence rate can be estimated using the hazard function $h(t)$. These estimates give software developers or testers a baseline for designing the software testing and maintenance plan.

4 Empirical Studies

4.1 Implementation

We implemented BUGMINER in Java using some existing machine learning and statistical analysis tools, including Weka [17] and MATLAB [9].

4.2 Bug Report Datasets and Data Processing

We experiment BUGMINER on the bug report repositories of three real-world software (Apache Tomcat [1], Eclipse [7], and Linux Kernel [11]). Table 3 lists some statistics of these bug report repositories. For example, the Apache Tomcat dataset contains two product versions—Tomcat 3 and Tomcat 7. The OS is the operating system the software runs on. The components are the functional components of the software.

Table 3. Software and bug report datasets

Software Name	# bug reports	# product	# OS	# components
Apache Tomcat	1525	2	16	16
Eclipse	1674	2	17	13
Linux Kernel	1692	16	1	106

We first apply pattern matching to extract static meta information, as listed in Table 1. Then we process free text descriptions using tokenization and bag-of-words feature selection as described in section 3.2. The dimensionalities of the term feature space range from 4000 to 13,000 depending on the dataset. After the attribute data sources are combined, the final vector space to represent bug report instances includes static meta information and bag-of-words features.

4.3 Results and Analysis

Our experimental results show that BUGMINER is effective in automatic completion check, automatic redundancy check, and bug report trend analysis. The following subsections present the detailed results and analysis.

4.3.1 Classification for Missing Field Autocompletion

For missing field autocompletion, we train classification model on 80% of the data and do blind-test on the remaining 20% of the data. For example, for Apache Tomcat, we use 1220 (or 80%) bug reports as training data and use 305 (or 20%) bug reports as the testing data. Table 4 lists the classification results for the Tomcat version. The accuracy of the classfication on testing instances is 99.02%. This means the product version in this case can be determined by the automatic completion check engine highly accurately.

Table 4. Classification results of products

TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
0.991	0.014	0.996	0.991	0.993	0.989	tomcat 3
0.986	0.009	0.973	0.986	0.98	0.989	tomcat 7
0.99	0.012	0.99	0.99	0.99	0.989	Weighted Avg.

Table 5 lists the classification results for the operating system version for Tomcat. The accuracy of the classification on testing instances is 68.52%.

Table 5. Classification results of OS versions

TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
0.888	0.449	0.758	0.888	0.818	0.719	all
0.356	0.081	0.432	0.356	0.39	0.637	linux
0.087	0.018	0.286	0.087	0.133	0.535	other
0.176	0.014	0.429	0.176	0.25	0.581	solaris
0.786	0.047	0.629	0.786	0.698	0.869	windows xp
0.685	0.294	0.632	0.685	0.647	0.696	Weighted Avg.

Table 6 lists the classification results for the software component related to the bug report. The accuracy of the classfication on testing instances is 53.11%. The results show that it is relatively difficult to accurately determine the problematic component based on the bug reports in this case.

Table 6. Classification results of components

TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
0.5	0.007	0.714	0.5	0.588	0.747	auth
0.868	0.067	0.73	0.868	0.793	0.9	catalina
0.2	0.039	0.313	0.2	0.244	0.58	config
0.368	0.037	0.583	0.368	0.452	0.665	connectors
0.5	0.003	0.5	0.5	0.5	0.748	encoding
0.622	0.041	0.676	0.622	0.648	0.79	jasper
0.667	0.003	0.667	0.667	0.667	0.832	manager
0.6	0.154	0.513	0.6	0.553	0.723	servlet
0.1	0.007	0.333	0.1	0.154	0.547	webapps
0.531	0.091	0.535	0.531	0.518	0.72	Weighted Avg.

We also did some experiments on the bug report datasets of Eclipse and Linux Kernel. Table 7 shows the summary of classification accuracy rates for the datasets tested. As the number of classes increases, the accuracy rate tends to decrease; nevertheless, the accuracy rates (*e.g.*, 53.11% for Tomcat's components) are relatively high if they are compared to the chance baseline (*i.e.*, probability is $1/n$ if there are n possible components).

4.3.2 Similarity Ranking

We first transform the historic training bug reports and the testing bug report to vectors using the vector space model.

Table 7. Summary of classification accuracy

Software Name	product	OS	components
Apache Tomcat	99.02%	68.52%	53.11%
Eclipse	97.90%	66.47%	67.37%
Linux Kernel	76.33%	N/A	58.88%

After the csv and kld value for each training bug report are calculated, all the training bug reports are then sorted in an descending order based on the csv value and in an ascending order based on the kld value. The bug reports at the top of the ranked lists are the most similar ones to the testing bug report.

Based on the heuristics from the experiments, we determine the value range parameters as $c\_r_1 = 0.2, c\_r_2 = 0.9, k\_r_1 = 2.0$, and $k\_r_2 = 10.0$ for Tomcat. Table 8 lists some sample results for a given bug report #393. From the results, the bug report #393 is highly likely to be a duplicate of some prior reports because there exists historic bug reports with $csv \geq 0.9$ and $kld \leq 2.0$ (*i.e.*, bug report #330 and #296). Furthermore, bug report #228 is likely to be a similar bug report of #393 because it has $0.7 < csv < 0.9$ or $2.0 < kld < 10.0$. To determine whether a new or given bug report is in fact a duplicate usually requires human judgement. Our manual verification shows that the similarity ranking results produced by BUGMINER are highly accurate despite the low quality data.

Table 8. Similarity ranking results

bug_id	csv	kld
330	0.928	1.940
296	0.917	0.816
228	0.717	9.868

4.3.3 Trend Analysis

We implement the bug report trend analysis based on the Weibull distribution. We first aggregate the historic data to compute a vector of the time (i-th week) and the number of bug reports whose first reporting date falls in the i-th week. Then a result vector returns the 95% confidence intervals for the estimates of the parameters of the Weibull distribution given the historic vector data. The two-element row vector estimates the Weibull parameter λ and k. The first row of the 2-by-2 matrix contains the lower bounds of the confidence intervals for the parameters, and the second row contains the upper bounds of the confidence intervals.

Table 9 shows the estimates of the Weibull parameters for Apache Tomcat 3. The value of k is less than 1, which indicates that the incidence rate decreases over time. The related curve fit is illustrated in Figure 2. The starting time, (*i.e.*, the 0 on the x-axis) is the week of August 25, 2000. The curve fit shows that the Weibull distribution closely resembles the actual bug report incidence distribution.

Table 9. Weibull parameter estimates

Software	λ	λ_{low}	λ_{high}	k	k_{low}	k_{high}
Tomcat 3	0.3885	0.2280	0.6621	0.2241	0.2041	0.2461

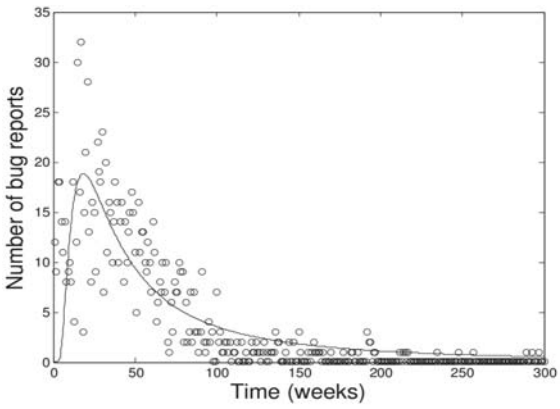


Figure 2. Weibull fit for Tomcat 3

5 Related Work

Some prior studies have been done on applying data mining on software engineering. [10] described the concept of software intelligence and the future of mining software engineering data. [18] presented a general overview of data mining for software engineering and described an example of duplicate bug detection using vector space-based similarity. [16] also described an approach to detect duplicate bug reports using both natural language and execution information. Our redundancy check engine uses both probability distribution-based KL divergence and vector space-based Cosine similarity ranking, instead of only vector space-based similarity. Furthermore, our approach provides a similarity ranking list that can be used for search, instead of only Yes and No on duplication check. [8] presented text mining of bug reports to identify security issues. Their work aims to identify security problems such as buffer overlow through mining the bug reports. Their purpose and techniques are different from our approach.

6 Conclusion

In this paper, we presented BUGMINER, a tool that is able to derive useful information from historic bug report database via data mining, use these information to do completion check and redundancy check on a new or given bug report, and to estimate the bug report trend using statistical analysis. We did empirical studies of the tool using several real-world bug report repositories. The experimental results show that BUGMINER is effective, easy to implement, and has relatively high accuracy despite low quality data. BUGMINER can be integrated into some existing bug tracking tools or software testing suites for more intelligent and cost-efficient software reliability analysis.

7 Acknowledgments

Wu and Kaiser are members of the Programming Systems Laboratory, funded in part by NSF CNS-0717544, CNS-0627473 and CNS-0426623, and NIH 2 U54 CA121852-06.

References

[1] Apache Project. http://issues.apache.org/bugzilla/, 2011.
[2] Bugzilla. http://www.bugzilla.org, 2011.
[3] Debian Bug Tracking System. http://www.debian.org/Bugs, 2011.
[4] Debian Project. Debian Bug report logs – #615000. http://bugs.debian.org/cgi-bin/bugreport.cgi?bug=615000, retrieved 2011-03-03.
[5] C. Cortes and V. Vapnik. Support-Vector Networks. *Machine Learning*, 20, Springer, 1995.
[6] T. M. Cover and J. A. Thomas. *Elements of Information Theory*, Wiley, 1991.
[7] Eclipse. http://bugs.eclipse.org/bugs/, 2011.
[8] M. Gegick, P. Rotella, and T. Xie. Identifying security bug reports via text mining: An industrial case study. In *Proc. of the 7th IEEE Working Conference on Mining Software Repositories (MSR)*, Cape Town, pp. 11–20, May 2010.
[9] A. Gilat. *MATLAB: An Introduction with Applications 2nd Edition*, John Wiley & Sons., July 2004.
[10] A. E. Hassan and T. Xie. Software Intelligence: Future of Mining Software Engineering Data. In *Proc. of the FSE/SDP Workshop on the Future of Software Engineering Research (FoSER 2010)*, Santa Fe, NM, pp. 161–166, November 2010.
[11] Linux Kernel. http://bugzilla.kernel.org/, 2011.
[12] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*, Cambridge University Press, 2008.
[13] S. E. Rigdon and A. P. Basu. Estimating the intensity function of a Weibull process at the current time: Failure truncated case. *Journal of Statistical Computation and Simulation (JSCS)*, vol. 30, pp. 17–38, 1988.
[14] G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Communications of the ACM*, v. 18 n. 11, pp. 613–620, November 1975.
[15] V. N. Vapnik. *The nature of statistical learning theory*, Springer-Verlag, New York, 1995.
[16] X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun. An Approach to Detecting Duplicate Bug Reports Using Natural Language and Execution Information. In *Proc. of the 30th International Conference on Software Engineering (ICSE)*, pp. 461–470, ACM Press, 2008.
[17] I. H. Witten, E. Frank, L. Trigg, M. Hall, G. Holmes, and S. J. Cunningham. Weka: Practical Machine Learning Tools and Techniques with Java Implementations. In *Proc. of the ICONIP/ANZIIS/ANNES'99 Workshop on Emerging Knowledge Engineering and Connectionist-Based Information Systems*, pp. 192–196, 1999.
[18] T. Xie, S. Thummalapenta, D. Lo, and C. Liu. Data Mining for Software Engineering. *Computer*, vol. 42, no. 8, pp. 55–62, IEEE Computer Society, Los Alamitos, CA, USA, 2009.
[19] J. Lafferty and C. Zhai. Document language models, query models, and risk minimization for information retrieval. In *Proc. of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 111–119, 2001.

Specification and Runtime Verification of API Constraints on Interacting Objects

Fang Deng, Haiwen Liu, Jin Shao, Qianxiang Wang
School of Electronics Engineering and Computer Science, Peking University
Beijing, China
{dengfang07, liuhw08, shaojin07, wqx}@sei.pku.edu.cn

***Abstract*—Most applications need to invoke some Application Programming Interfaces (APIs), e.g. JDK (Java Development Kit) API. When invoking those APIs, applications must follow some API constraints. Violation of these constraints will lead to some severe program defects. To detect this kind of defects, lots of static and dynamic approaches are explored, using formally described API constraints. While most existing approaches explore API constraints on a single object, this paper focuses on API constraints on interacting objects (COIOs). We proposed a novel specification language LACOIO (Language for API Constraint on Interacting Objects) which can cover all state-of-the-art reported API COIOs. We implemented a LACOIO compiler which can automatically generate monitoring code for runtime verification from API constraints that are specified using LACOIO and implemented the runtime verification framework. We evaluated the proposed approach by comparing it with Tracematch – a popular constraint specification language.**

Keywords- Runtime verification; runtime monitoring; specification; program analysis.

I. INTRODUCTION

Most applications need to invoke some Application Programming Interfaces (APIs), e.g. JDK (Java Development Kit) API. When invoking APIs, applications must follow some API constraints. For example, one constraint related to JDK is: "*instance of java.io.InputStream must be closed explicitly*". Violation of these constraints will lead to some severe program defects. To detect this kind of defects, lots of static and dynamic approaches are explored [9, 12]. For both kinds of approaches, formally described constraints are widely used in the detecting process. Static analysis tools can read these formally described constraints and check the code automatically, while dynamic verification frameworks can use these constraints to generate monitors automatically to verify applications at runtime. Finite state automaton (FSA), temporal logic, and regular expression are typical formal methods used to describe API constraints [3, 5, 8].

While most existing approaches explore API constraints on a single object [7, 8, 11], some studies explore API constraints on multiple interacting objects (COIOs) [2, 10]. For example, one constraint related to JDK (we call it SafeIterator Constraint) is: "*an array list of java.util.ArrayList cannot be updated when it is being traversed by an iterator of java.util.Iterator"*. Fig. 1 shows the FSA-based expression for this constraint [3].

However, the event symbols in this FSA reveal only the event names without information about which objects these events belong to. Actually, this FSA makes some implicit assumptions: event CREATE_LIST occurs on a java.util.ArrayList object *al*; event CREATE_ITERATOR occurs on the java.util.Iterator object *i* which is created by *al*; all UPDATE events belong to the same object *al*; and all NEXT events belong to the same object *i*.

When an API constraint involves a group of objects like this, it is very difficult to distinguish which object an event is associated with. Some approaches [1, 5] add free variables to events (symbols in the FSA) to address this problem. These approaches use a variable binding strategy to associate objects with events. In Fig. 2, two free variables *al* and *i* are introduced as well as two operators "=" and ".". Edge "*al*= CREATE_LIST" indicates that the returned object of the operation which is matched to the event CREATE_LIST is bound to object *al*. Edge "*al*.UPDATE" matches the updating operations which happen on object *al*. Other edges can be explained similarly. Specification in Fig. 2 has two benefits. First, it points out which object an event is associated with; secondly, it points out how objects *al* and *i* are bound together, i.e., *i* is the Iterator object that is created by *al*.

However, for constraints on interacting objects whose number is not stable, it is still very difficult to describe them formally, even with existing free-variable-based approaches (See section II for details). Moreover, the approach of formal specification has great impact on the implementation of corresponding verification framework. In this paper, we proposed a novel specification language LACOIO (Language for API Constraint on Interacting Objects) to describe API constraints. The proposed language is powerful enough to describe state-of-the-art reported API COIOs easily and leads to efficient runtime verification.

The rest of the paper is organized as follow: In Section II we present a COIO example to introduce our motivation and give a brief introduction to COIO specification problem. In Section III we present the proposed specification language LACOIO. In Section IV we introduce the implementation of LACOIO compiler and the runtime verification framework. Section V is the evaluation of our work. Section VI is the introduction of related work. We conclude our paper in Section VII.

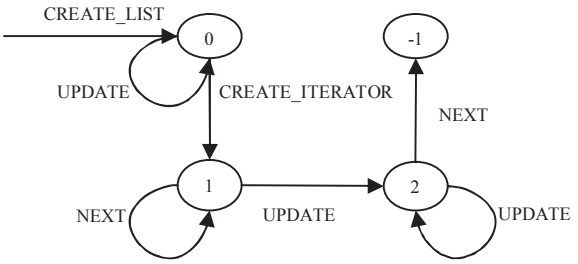


Figure 1. FSA for the constraint on ArrayList and Iterator.

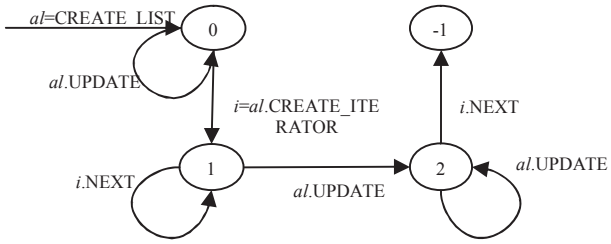


Figure 2. Extended FSA for constraint on ArrayList and Iterator

```
20 DB db = new DB();
21 Connection conn=db.dbConnect(
22              "jdbc:mysql://…", "root", "");
23 Statement stmt = conn.createStatement();
24 ResultSet rs1 = stmt.executeQuery(…);
25 ResultSet rs2 = stmt.executeQuery(…);
26 // rs1 is unavailable after rs2 is created.
27 while (rs1.next())// Constraint is violated!
28 {…}
```
Figure 3. Sample codes that violate SafeJDBC constraint.

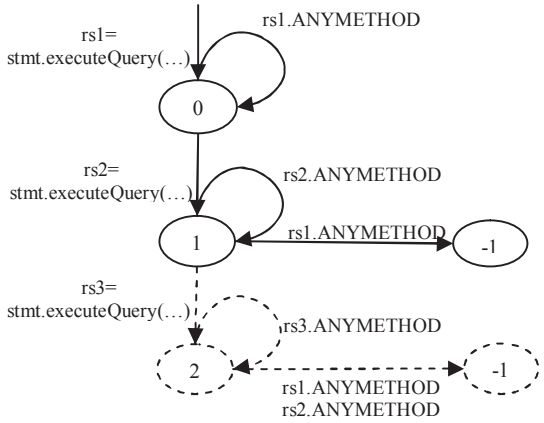


Figure 4. An example which tries to use FSA with free variables to specify SafeJDBC.

II. PROBLEM STATEMENT

To clarify the complexity of API COIO verification, we present another motivating example. The constraint is about JDBC (we call it SafeJDBC Constraint): "*if two java.sql.ResultSet objects (e.g. rs1 and rs2) are created by the same java.sql.Statement object (e.g. stmt) through Statement.executeQuery(...) method, the first ResultSet rs1 becomes unavailable after rs2 is created*" [4]. In other words, we cannot access *rs1* through any of its methods after *rs2* is created. Fig. 3 is a piece of sample code that violates SafeJDBC. SafeJDBC involves at least 3 objects, *stmt* of class Statement, the *rs1* and *rs2* of class ResultSet. Now we try to describe it by FSA which introduces free variables similarly as Fig. 2. In Fig. 4, the real line elements describe the violation pattern involving three objects: *stmt*, *rs1*, *rs2*. "-1" in the FSA stands for the state when a violation is detected. Usually, at runtime, such an FSA instance will be created to verify behavior of the three objects.

However, the problem is: what if a third instance of ResultSet *rs3* is created? Even if we can extend the FSA to describe the violation pattern when *rs3* is involved (see the dotted line elements in Fig. 4), what if more instances of ResultSet are created?

There are two facts that make it problematic to describe the constraint by FSA, even with introduction of free variables. (1) The verification must be carried out on instance-level (not the class-level). Although this type of constraint involves finite classes, the number of involved objects is undetermined. It is impossible to enumerate all runtime instances in an FSA. (2)The other characteristic of such kind of constraints is that, whether behavior of one object will lead to violation of the constraint depends on behavior of another object of the same class. This situation prevents us from describing the behavior of such two objects in two separate FSAs.

A. Problem-related Terminologies

Before introducing the details of COIO specification problem, we first explain some relevant terminologies.

Monitor. A monitor is an instance for a constraint. It inspects and verifies the behavior of a set of interacting objects. In Fig. 4 the monitor is an FSA instance which is created to verify the behavior of *stmt, rs1, rs2, rs3*.

Monitee. At runtime if an object is involved in the verification and will be monitored, we call it a *monitee*. In Figure 4, object *stmt, rs1, rs2, rs3* are monitees.

Monitee set. All monitees that are verified in one monitor constitute the monitee set of the monitor. In Fig. 4 the monitee set is {*stmt, rs1, rs2, rs3*}. The monitee set of a monitor can be divided into several subsets according to the monitees' types. The monitees of a certain class C constitute the monitee set of C. For example, there is a monitee set {*rs1, rs2, rs3*} of ResultSet.

Binding point. Assume C1, C2 are two classes, C1 has a method *m* whose signature involves C2, we call *m* a *binding point* for C1 and C2, e.g., method `ResultSet Statement.executeQuery(String str)` is a binding point for Statement and ResultSet.

Binding relationship. Assume *m* is a binding point for C1 and C2. At runtime when *o1* and *o2* are objects of C1 and C2 respectively and *m* is executed on *o1,* we call *o1* and *o2* are bound (by *m*) or *o2* is bound to *o1* (by *m*), written *o1 BP o2*. For example, when "`rs2 = stmt.executeQuery(…)`" is executed, *stmt* and *rs2* are bound by method "`ResultSet Statement.executeQuery(String str)`", written *stmt BP rs2*.

B. Problem of Specification

In this paper we restrict our focus on COIO whose monitee set is defined as $\{k\} \cup \{ i \mid k\ BP\ i \}$, where *k* is an instance of a

class C. *k* is called the **core object** in the monitor while C is called the **core class** of the COIO. We reviewed state-of-the-art literatures and found even with this restriction the proposed approach can cover all state-of-the-art API constraints.

From the definitions of monitee set of COIO monitor, we know that there is always one core object in the monitee set of each COIO monitor and all the other objects are involved by binding relationship. This indicates a one-to-many mapping between the object of core class and objects of other involved classes in COIO. Constraint SafeJDBC is such a COIO: *stmt* is the core object in its monitee set and all the ResultSet typed objects that are created by *stmt* are involved in the same monitor by binding point "`ResultSet Statement.executeQuery (String str)`". Fig. 5 illustrates the monitee set, its monitees, and related objects.

The key challenge for traditional FSA monitor is that its monitee set must be a fixed set. That means, for FSA, we must present each possible object of the monitee set in FSA description, which is impossible in the aforementioned case before runtime because the number of objects in the monitee set is undetermined. For example, in Fig. 4, we must include all the objects *stmt, rs1, rs2, rs3* to describe the constraint. It is likely that more ResultSet objects are created in the near future. So the FSA-based approach can't describe such undetermined cases.

III. Language for API Constraint on Interacting Objects

LACOIO can describe complex COIOs in a succinct way and make specifications easy to understand. Complete grammar and more examples can be found in [13]. In this section, we explain the LACOIO specification with the example of SafeJDBC constraint (see Fig. 6).

A. Monitor:

A monitor is defined with the key word **monitor**. At runtime a monitor instance represents an instance of COIO verification case.

B. Monitee set management and binding:

The monitee set is defined with key word **monitee**, contains the objects that are involved in a monitor instance. The monitor usually contains two kinds of monitee sets: monitee set for core class which contains only one core object and monitee sets for other classes. In our specification, we do not distinguish these two kinds of monitee sets. Monitee sets are maintained dynamically at runtime and do not need to be determined before runtime. In the example, two monitee sets @stmt and @rs are specified respectively for class Statement and ResultSet. At runtime each SafeJDBC monitor maintains such two sets. Since Statement is the core class for SafeJDBC, there is only one object of Statement in @stmt which is the core object. Whenever a ResultSet object is obtained by the binding point "`ResultSet Statement. executeQuery(…)`", the returned ResultSet object is bound to the core object and thus put into @rs.

C. User-defined variable :

We allow users to define their own variables with keyword **var** to store historical states that can be acquired in the future, e.g. a reference to certain monitee in the monitee set, a status tag, etc. In Fig. 6, "**`var`** `ResultSet $last`" defines a variable called $last of type ResultSet. It is used to store the reference to the most recently created ResultSet object.

D. Event

When a method is invoked on a monitee, it will be mapped to the corresponding event that is specified in the constraint according to particular mapping rules. It is similar with the process of mapping method calls to edges in an FSA. An event declaration consists of event type, event name, contract part, pre/post-condition, revise part, and action.

The contract part. The contract part specifies the rules that how method invocations should be mapped to events. For example, in Fig. 6, the contract part "`@stmt.executeQuery (String)`" of event `STQUERY` states that invocation to "`executeQuery(…)`" methods on the object which is in the *@stmt* set should be mapped to the `STQUERY` event.

Pre/Post-condition. Pre/Post-conditions are defined for every event in the constraint, which are identified by "`[ ]`" before/after a contract part. They are part of the mapping rules. Only when pre/post-conditions are both *true* can a method invocation be matched to an event successfully. They are boolean expressions on *$target*, *$args[n]*, *$return* and the user-defined variables. *$target, $args[n],* and *$return* are reserved keyword that are used to refer to the target object, the *n*th argument, and the return value of a method call, respectively. The pre/post-conditions are checked before/after the invocation of the method which is specified in an event declaration. When

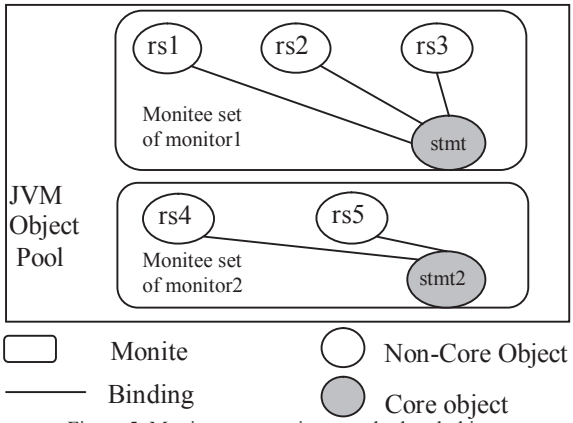


Figure 5. Monitee set, monitees, and related objects.

```
1 monitor SafeJDBC {
2 monitee @stmt< Statement>, @rs<ResultSet>;
3       var ResultSet $last = null;
4       init Connection.createStatement()
5               (@stmt.add($return))
6               { /* empty action */}
7       event STQUERY
8               @stmt.executeQuery(String)
9               (@rs.add($return))
10              { $last = $return;  }
11      validate RSNEXT
12              [$last!=$target]
13              @rs.next()
14              { System.out.println("Error");  }
15 }
```

Figure 6. Specification of constraint SafeJDBC

pre/post-conditions are both *true*, we say that an event match happens. For example, when "`next()`" is invocated on a ResultSet object *rs1*, the monitor will try to map the invocation to *RSNEXT* according to the contract part of event *RSNEXT*. The pre-condition "`$last!=$target`" of *RSNEXT* will be checked to determine whether the accessed ResultRet *rs1* is not the most recently created one. If the condition is satisfied, event `RSNEXT` will be matched.

Revise part. Revise part states how the monitee sets should be maintained when an event match happens. There are four types of operations for the monitee set: add, remove, assign and clear. For example, when `STQUERY` event match happens, the revise part "`@rs.add($return)`" states that the returned ResultSet object should be added into monitee set *@rs*.

Action. Action includes operations to the monitees, user-defined variables, *$target/$return/$args[n]*, and also some simple java statements. The action part is executed only when the event match happens. For example, when event `STQUERY` is matched, operation "`$last = @return`" will be executed.

There are two special types of events: `init` and `validate` event. An `init` event (denoted by keyword **init**) represents the start of the verification, and it is also the point of construction of the core object. When an `init` event match happens, a monitor is instantiated, thus there must be only one `init` event. In SafeJDBC specification, the contract part "`Connection.createStatement()`" of the `init` event states that invocations of "`createStatement()`" method on all Connection objects should be mapped to the `init` event. When a `validate` event (denoted by keyword **validate**) match happens, a violation is detected. In the action part of `validate` event, users can perform some actions to handle constraint violations. For example, in the SafeJDBC specification example, when "`next(…)`" is invoked on one of the ResultSet monitees and ResultSet object referred by *$target* does not equal *$last*, `validate` event `RSNEXT` is matched and constraint violation is reported.

IV. THE IMPLEMENTATION

The specification file will be compiled to monitoring codes. The compiler takes LACOIO specification as input and compiles it into two parts. (1) One is an AspectJ [6] file which acts as a middle representation file. In general, the responsibility of the AspectJ file is to declare probes that map method invocations to events and transfer messages at runtime. Pointcuts will be generated according to event contract declarations. The codes in an advice will be generated to transfer the method invocation messages to corresponding envoy. (2) The other part is classes that contain the actual verification logic codes which include classes of envoys and monitors. AspectJ file is subsequently weaved to the original byte code file of the target system and an instrumented class file is produced. The instrumented class and the verification logic classes will be deployed to JVM.

The structure of the runtime verification framework is shown in Fig. 7. JVM objects pool is presented in the left side in which there are objects of verification concern. The squares

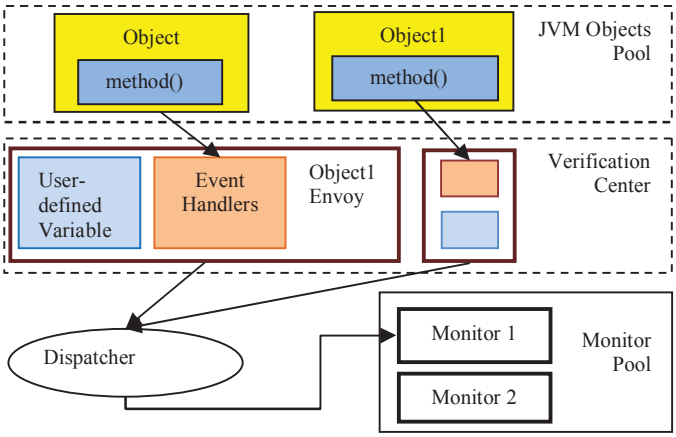


Figure 7. Runtime verification framework for COIO

in the objects represent methods that have been instrumented with probes. The probes send messages to the corresponding objects. The handler of the envoy will process the messages sent from the probes. And if necessary, the envoy will pass the message to the monitor for further processing.

V. EVALUATION

A. Expression Ability

We compare the ability of expression of LACOIO with that of Tracematch[1]. Tracematch is a prevalent language that can also describe some constraints on multiple interacting objects by introducing free variables. The basic logic that LACOIO and Ttracematch use is equivalent to FSA. They introduce different extended features which make them more powerful than FSA. Moreover, LACOIO makes the specification of constraints which involve multiple objects more succinct and understandable by allowing explicit monitee set declaration, flexible user-variable definition and pre/post-conditions.

We present another example SafeDropDownList to illustrate the advantages that LACOIO brings. SafeDropDownList is originally a constraint on the usage of DropDownList and ListItem in ASP.NET framework, which was first proposed in [2]. We simulate two similar interfaces of DropDownList and ListItem in Java (see Fig. 8). A DropDownList *list* may contain several ListItem *i1, i2*... The ListItem *i1, i2*… can be obtained by calling method "`findItemByName(String)`" on *list*. A ListItem is set to "selected" or "deselected" by calling the mehod "`setSelected(boolean isSlected)`" on it.

```
public interface DropDownList
{ ListItem findItemByName(String item_name);… }
public interface ListItem
{ void setSelected(boolean isSlected);… }
```

Figure 8. Stimulated DropDownList and ListItem in Java

```
5  DropDownList list = new DropDownList();
       ……
10 ListItem i1=list.findItemByName(ITEM_NAME_1);
11 ListItem i2=list.findItemByName(ITEM_NAME_2);
     ……
21 i1.setSelected(true);
22 //Forget to call: i1.setSelected(false);
23 i2.setSelected(true); //Constraint Violated!
```

Figure 9. Sample codes that violate SafeDropDownList.

```
1 monitor SafeDropDownList{
2 monitee @list<DropDownList>,@items<ListItem>;
3 var ListItem $selected = null;
4
5   init DropDownList.new()
6        (@list.add($return))
7        { /* empty action */}
8   event FIND_ITEM
9        @list.findItemByName(String)
10       (@items.add($return))
11       { /* empty action */}
12  event SELECT
13       [args[0]==true && $selected== null]
14       @items.setSelected(boolean)
15       {   $selected= @target;      }
16  validate SELECT_FALSE
17   [args[0]==true && $selected!= null]
18   @items.setSelected(boolean)
19   {System.out.println("Error Reported"); }
20  event DESELECT
21   [args[0]==false && $selected==$target]
22   @items.setSelected(boolean)
23   {  $selected = null;      }
24}
```

Figure 10. SafeDropDownList Specification in LACOIO

```
tracematch(DropDownList l, ListItem i1, ListItem i2) {
sym get1 after returning(i1) :
      call(* DropDownList.findItemByName(..)) && target(l);
sym get2 after returning(i2) :
      call(* DropDownList.findItemByName(..)) && target(l);
sym select after : target(i1) && private(boolean b) (
  call(void ListItem.setSelected(boolean) && args(b) && if(b)));
sym deselect after : target(i1) && private(boolean b) (
  call(void ListItem.setSelected(boolean) && args(b) && if(!b)));
sym select2 after : target(i1) && private(boolean b) (
  call(void ListItem.setSelected(boolean) && args(b) && if(b)));
distinct i1, i2;
  (get1 get2 (get1|get2)* select (get1|get2|select)* select2) |
  (get1 select (get1|select)* get2 (get1|get2|select)* select2)
  { error("Selecting " + i2 + " without deselecting " + i1);}
}
```

Figure 11. SafeDropDownList Specification in Tracematch

SafeDropDownList states that: "multiple ListItems in a DropDownList cannot be set to 'selected' at the same time". In other words, before trying to select a ListItem by calling `setSelected(true)` on it, one must first call `setSelected(false)` on the previously selected ListItem. Fig. 9 is a piece of sample code that violates this constraint.

Fig. 10 and 11 are constraint specifications written in LACOIO and Tracematch respectively. Both specifications define the same number of events (or symbols in Tracematch). But in Tracematch users are required to write a quite verbose regular expression. In LACOIO, we allow user to define a simple variable "*$selected*" to store the current selected ListItem. So with our approach we only need to check whether *$selected* is *null* when a second ListItem is about to be selected. However, with Tracematch, a pattern of events must be expressed in a temporal manner as regular expression, which is actually not necessary for verification. For example, the information of the temporal order of "select", "get1" and "get2" is not necessary for identifying a violation. All that is necessary to know is that the three events did happen before "selected2" happens.

Although LACOIO does not express the logic formula explicitly, by introducing user-defined variable properly it can express all the constraints that can be expressed by FSM and regular expression. Our motivation to hide the logic formula is that we find many state-of-the-art constraints are not pure temporal problem and can be expressed by our language more straightforwardly. (See [13] for more specification examples).

B. *Performance*

In this sub-section, we illustrate the performance improvement that our language design brings to the verification of some typical type of complex COIOs like SafeDropDownList (mentioned in section V.A) which involve interacting objects of the same class. To evidence this point, we verify a Java application which manipulates one DropDownList and multiple ListItems with respect to the SafeDropDownList constraint. The experiments are conducted on a PC equipped with Intel Pentium IV 3.0G Hz processor, a 2GB RAM, and a single 300GB 7200 RPM SATA disk. The operation system is Ubuntu 9.10. JDK version is 6.0.

Fig. 12 shows the execution time when Tracematch and our approach are applied. We run the application multiple times. X-axis shows the total number of ListItems that are created in a particular round of execution. Y-axis shows the time cost for the execution. As we expect, as the number of ListItems increases, the execution time increases for both Tracematch and our approach. But the increasing rate for Tracematch is approximately 7.4 times as that for our approach.

The main reason for this result is that, in LACOIO, monitee sets are declared explicitly and maintained at runtime. Since the DropDownList object is the core object of the monitor, the behavior of all ListItems that belong to the same DropDownList will be verified in one monitor instance. Compared to LACOIO monitor, the Tracematch monitor specified in Fig. 11 states the behavior pattern of three objects: DropDownList l, ListItem i1 and i2. In fact, the specification implies that information of each pair of ListItems which belongs to the same DropDownList are tracked and matched in one pattern. If there are three ListItem i1, i2, i3, the behavior of object pairs <i1,i2>,<i1,i3> and <i2,i3> will be verified by

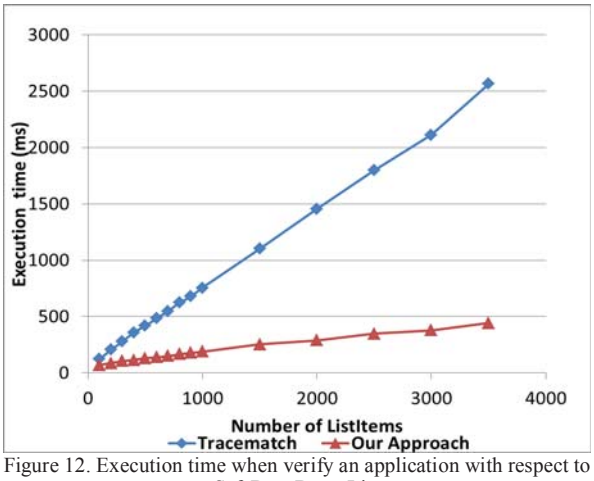


Figure 12. Execution time when verify an application with respect to SafeDropDownList

three automaton instances that are derived from the specified regular expression. But in LACOIO monitor, <i1, i2, i3> are verified in one monitor instance, which avoids unnecessary verification and decreases the complexity.

VI. RELATED WORK

Tracematch [1] is a seamless extension of AspectJ and is a new history-based language. The logic that it relies is regular expression. It introduces free variables in the matching patterns. At runtime, the monitored objects and other values can be bound to the declarative variables so that the event can be matched not only by event kind but also by the value that associated with the free variables. MOP [5] is a formal framework for software development and analysis, in which the developer specifies desired properties using definable specification formalisms, along with code to execute when properties are violated or validated. It supports several logic formalisms, including CFG, FSM, regular expression, etc. MOP allows specifications with parameters. The free variables in Tracematch and parameters in MOP allow user to specify some COIOs. However, since the free variables or parameters are limited to a fixed number, when they are applied to verify some complicated COIOs that involve multiple objects of the same class like SafeDropDownList, the specification will became verbose and hard to understand. LACOIO can lower the complexity of such specification and its monitor leads to a better performance for this type of COIOs.

Jaspan [2] proposed a lightweight specification system which can describe framework constraints that are temporal and involves multiple objects. Main differences between work in [2] and ours are: (1) the former is designed for static analysis. Some relations among objects are predicated so that they are not precise. The predicted relations are used for further analysis which makes the result unsound. LACOIO in this paper does not make any prediction. The verification is based on real runtime information of objects' behavior and its result is sound (but not complete). (2) LACOIO is more flexible as it introduces user-defined variable, actions, etc, while the language in [2] only allows very simple pre/post-conditions, which makes it not applicable to specify constraints like SafeIterator and SafeHashMap[13].

Yahav [4] proposed an approach to verify safety properties using separation and heterogeneous abstractions. They presented a language for specifying separation strategies for decomposing a single verification problem into a set of sub-problems and then utilized heterogeneous abstraction during the verification of the transformed verification problem. However, this is not a general approach to describe many state-of-the-art API constraints. Moreover, this approach is designed for static program analysis, which is not appropriate for runtime verification of COIO.

VII. CONCLUSION

This paper proposed a novel specification language LACOIO to describe complex constraints on multiple interacting objects. We presented the language design and the implementation of the language compiler and the runtime verification framework. We have demonstrated that with LACOIO, we can describe complex COIOs in an easy and succinct manner and our language design has led to improvement of performance.

ACKNOWLEDGMENT

This work is supported by the National Basic Research Program of China (973) under Grant No. 2009CB320703 the Science Fund for Creative Research Groups of China under Grant No. 60821003.

REFERENCES

[1] C. Allan, et al. "Adding trace matching with free variables to AspectJ," in *OOPSLA '05: Proceedings of the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*. New York, NY, USA: ACM, 2005, pp. 345–364.

[2] C. Jaspan and J. Aldrich, "Checking temporal relations between multiple objects," Institute for Software Research, Carnegie Mellon University, Pittsburgh, Pennsylvania, Tech. Rep., 2008.

[3] E. Bodden, P. Lam, and L. Hendren, "Finding programming errors earlier by evaluating runtime monitors ahead-of-time," in *SIGSOFT '08/FSE-16: Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*. New York, NY, USA: ACM, 2008, pp. 36–47.

[4] E. Yahav and G. Ramalingam, "Verifying safety properties using separation and heterogeneous abstractions," in *PLDI '04: Proceedings of the ACM SIGPLAN 2004 conference on Programming language design and implementation*. New York, NY, USA: ACM, 2004, pp. 25–34.

[5] F. Chen and G. Rosu, "MOP: an efficient and generic runtime verification framework," in *OOPSLA '07: Proceedings of the 22nd annual ACM SIGPLAN conference on Object-oriented programming systems and applications*. New York, NY, USA: ACM, 2007, pp. 569–588.

[6] Gregor Kiczales, et al. "An Overview of AspectJ". in *ECOOP '01: Proceedings of the 15th European Conference on Object-Oriented Programming*. London, UK: Springer, 2001, pp. 327–353.

[7] K. Bierhoff and J. Aldrich, "Modular typestate checking of aliased objects," in *OOPSLA '07: Proceedings of the 22nd annual ACM SIGPLAN conference on Object-oriented programming systems and applications*. New York, NY, USA: ACM, 2007, pp. 301–320.

[8] M. B. Dwyer and R. Purandare, "Residual dynamic typestate analysis exploiting static analysis: results to reformulate and reduce the cost of dynamic analysis," in *ASE '07: Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*. New York, NY, USA: ACM, 2007, pp. 124–133.

[9] M. Gopinathan and S. K. Rajamani, "Enforcing object protocols by combining static and runtime analysis," in *OOPSLA '08: Proceedings of the 23rd ACM SIGPLAN conference on Object-oriented programming systems languages and applications*. New York, NY, USA: ACM, 2008, pp. 245–260.

[10] N. A. Naeem and O. Lhotak, "Typestate-like analysis of multiple interacting objects," in *OOPSLA '08: Proceedings of the 23rd ACM SIGPLAN conference on Object-oriented programming systems languages and applications*. New York, NY, USA: ACM, 2008, pp. 347–366.

[11] R. E. Strom and S. Yemini, "Typestate: A programming language concept for enhancing software reliability," *IEEE Trans. Softw. Eng.*, vol. 12, no. 1, pp. 157–171, 1986.

[12] S. Fink, et al. "Effective typestate verification in the presence of aliasing," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 17, no. 2, pp. 1–34, 2008.

[13] www.sei.pku.edu.cn/~dengfang07/LACOIO.html

Applying Lightweight Formal Approach to Automatic Configuration Inspection

Sachoun Park[1], and Gihwon Kwon[2]
[1]Electronic and Telecommunications Research Institute,
138, Gajeongno, Yuseong-Gu, Daejeon, Korea
sachem@etri.re.kr
[2]Department of Computer Science, Kyonggi University,
San 94-6, Yiui-Dong, Youngtong-Gu, Suwon-Si, Kyonggi-Do, Korea
khkwon@kgu.ac.kr

Abstract

To alleviate the complexity and changeability of development of large scale embedded system, the Model Driven Development is accepted by various fields of industries and their standards. In this approach, design models with vast configuration data are translated into executives automatically. However it's difficult to detect incorrect values from the configuration information. In this paper, we introduce lightweight formal approach using SMT solver for the automation of configuration inspection activity, which is rather to be suited to industrial purpose.

Keywords : Embedded System, Automatic Configuration Inspection, Formal method, SMT

1 Introduction

Software developers in the project of large scale embedded system such as automotive, avionics, and mobile system are challenged to meet delivery dates in the face of complex system architectures and ever-evolving technological platforms. To overcome these challenges, MDD(Model Driven Development) is adopted by the automotive industrial standards, AUTOSAR[1]. For the aeronautic domain, the European project DIANA investigated the use of MDD in the context of Integrated Modular Avionics and the ARINC 653[2] standard. MDD facilitates on systematic use of models from a very early phase of the design process and through various model transformation steps automatically generates source code[3]. However, in those standards, the heavy use of configuration data expose developers to error-prone configuration file editing, so the validation for early error detection of configuration is needed.

System configuration is one of activities performed during the development of a system, which is to provide configuration information for all the software within the system. It's an important activity of the development of embedded software, because it gives the flexibility to the system development. The output of the activity is the system configuration description which is used to actually generate and build the executable image. Mainly configuration data are used to set to particular values for operating system, system timers, and so on.

Source code inspection is a commonly used technique for verification of software in industry. Recently from international standard like ISO 26262[4] and DO178-B[5] rigorous applying of verification activities is emphasized. However, the inspection is performed manually, which is time consumed and error-prone. Thus more rigorous and automatic inspection is required and formal methods are popular techniques in this sense. Although formal methods are becoming more widely used, formal proof has achieved less industrial acceptance, due to cost and limitations of tools and scalability problem. However, configuration inspection with lightweight formal method can be a practical approach for automation of verification and validation[6].

In this paper, we describe an automated approach to configuration inspection that utilizes lightweight formal method using SMT(Satisfiability Modulo Theory)[7] solver. In our approach configuration files are represented by configuration trees and these trees are translated into logical expressions in the form of SMT input language.

This work was supported by the GRRC(Gyeonggi Regional Research Center) program of Gyeonggi province and the WBS(10038486) program of the Ministry of Knowledge Economy, Korea.

Also the set of constraints of configuration data is translated into logical expression. The logical expressions are solved by SMT solver for detecting violation of constraints of the configuration.

Our major contributions in this paper include (1) the establishment of the approach to apply formal method to automation of configuration inspection, (2) the design of the translation algorithms from the configuration code to the logical expressions, (3) the implementation of a prototype software tool to support the activities, and (4) the presentation of a case study of inspecting DPS(Driver Positioning System) to evaluate the correctness of the approach and the tool.

The remainder of this paper is organized as follows. Section 2 presents the logical foundation as background. Section 3 describes the automation of configuration inspection. Section 4 reports applying the method to DPS system as a case study with prototype tool. Finally, we conclude the paper and point out future research in section 5.

2 Background

In the last two decades, the great progress has been made in the field of determining the satisfiability of first order logic formulas, because it focuses not general first order satisfiability, but rather satisfiability with respect to some background theory, which fixes the interpretations of certain predicate and function symbols. The research field of satisfiability of formulas based on those background theory is SMT[9].

In SMT formulas usually involve several theories at once, which include not only theories of real and integer arithmetic but also theories of array, list, tree, record and so on. This is particular useful to verify software system. Since recent SMT solvers such as Yices[10] are good at solving linear arithmetic problem, we translate the inspection problem into Yices input language and solve the linear arithmetic formulae to fine out any violation of the system configuration. In this paper, we use logical expressions constructed by the following abstract syntax:

$\phi ::= \neg\phi \mid \phi \wedge \phi \mid (\phi) \mid term = term \mid term > term$

$term ::= variable \mid constant \mid term + term$

This Arithmetic involves the addition and multiplication of the real numbers, so the domain of *variable* and *constant* is real number. If $P = \{p_1, \ldots, p_n\}$ denotes the set of logical expressions of the configuration data and each element in P forms (*variable* = *constant*), the logical formula of the whole configuration data is $\phi_P \equiv p_1 \wedge \ldots \wedge p_n$. If $C = \{c_1, \ldots, c_m\}$ denotes the set of logical expressions of configuration constraints, the logical formula representing the configuration constraints is $\phi_C \equiv c_1 \wedge \ldots \wedge c_m$. In order to check the correctness of configuration data, we can check $\models \phi_P \wedge \neg\phi_C$. If the result of the satisfiability check is unsatisfiable, then the configuration is correct, otherwise incorrect data exit in the configuration. And from the model of the formula $\phi_P \wedge \neg\phi_C$, we can obtain the values of violated variables in the configuration.

3 Automation of Configuration Inspection

In this section, we describe the automatic method of configuration inspection. Configuration files consist of the definition of data structure and the set of assignment of variables called configuration parameters. In order to be analyzed, the configuration files are represented by configuration tree which looks like parse tree and then the logical expressions is constructed from the configuration tree. With constraints between configuration parameters, the set of logical expressions is solved by SMT solver.

3.1 Construction of configuration tree

Configuration data consists of the part of definition of structure and the part of declaration of variables and their value assignments. For example, a structure type A and B is defined and its variables are declared with their values in the left hand side of figure 1. The right hand side of below figure shows the configuration tree corresponding to the configuration code.

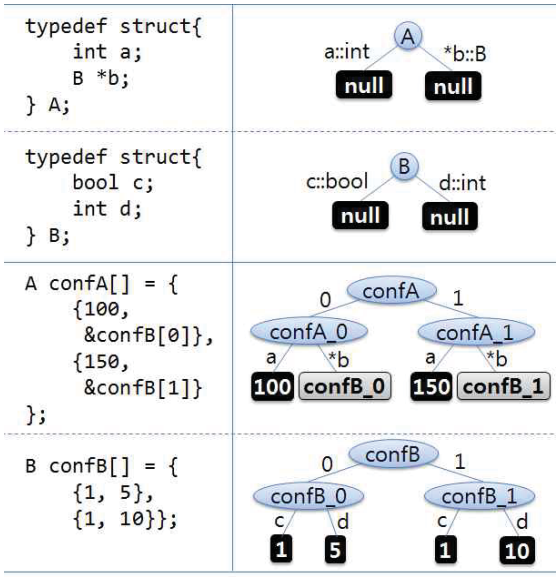


Figure 1. Configuration data and tree representation

In the figure 1, roots of trees are labeled by the name of defined structure type or the name of declared variable.

Each arc in the trees is labeled by the name of elements corresponding to a structure or an index number for array type. Non-terminal nodes are labeled by the composition of names of their parent and arc. Terminal nodes are labeled with values and the name of reference if the value of given variable is reference. For all variables declared in the given configuration files, if configuration trees are constructed completely, we can make a configuration tree as each arc forward a reference node links to the non-terminal node in other tree which has the same name with the reference node like figure 2.

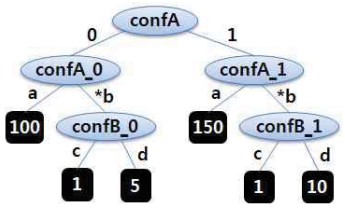


Figure 2. Example of configuration tree

After making the merged tree, logical expressions can be constructed. For example, the logical expression of configuration tree in the figure 2 is as follow:

$confA\_0\_a = 100 \wedge confA\_1\_a = 150 \wedge$
$confB\_0\_c = 1 \wedge confB\_0\_d = 5 \wedge$
$confB\_1\_c = 1 \wedge confB\_1\_d = 10$

In the above expression, a name of variables consists of the name of the parent node and the label of arc corresponding to each terminal node and equalities stand for the assignments of configuration code.

3.2 Formalization of configuration constraints

Configuration constraints are generally specified in standard documents as requirements and each constraint is the particular relation among deferent configuration parameters. It is presented by natural language such as "if $a > 100$ then c is true" and "If c is true then $d \times 20 > a$". This kind of statement can be naturally translated into the logical expression. However, the remarkable assumption is that there exists the mapping point as the same name between two variables in the expression of constraint and of configuration files. In order to translate the constraints into logical expressions, the names of variable within the logical expression of the constraint should be refined with respect to the configuration tree. To do this, we can assume the sub-tree whose root node has the highest arc among the set of arc labeled by names of the variable used in the expression of the constraint. For example, the logical expression $(a > 100) \Rightarrow c$, representing the constraint "if $a > 100$ then c is true", is refined to $(confA\_0\_a > 100) \Rightarrow (confB\_0\_c = 1)$ and $(confA\_1\_a > 100) \Rightarrow (confB\_1\_c = 1)$, which are caused by the array **confA** in the figure 2. If all refinements of expressions of constraint are completed, we can translate those expressions into Yices input language. From the second constraint "If c is true then $d \times 20 > a$", we can make below two logical expressions:

$(confB\_0\_c = 1) \Rightarrow (confB\_0\_d \times 20 > confA\_0\_a)$
$(confB\_1\_c = 1) \Rightarrow (confB\_1\_d \times 20 > confA\_1\_a)$

Notice that the above first formula is violated. So we can think that the configuration data is incorrect. However, with examining given constraints carefully, two example constraints are in the transitive relation. Through the connection of these two constraints, we can guess the fact "if $a > 100$ then $d \times 20 > a$". If it is right, the violation is meaningless. Because the value of a is not larger than 100, the relation between $d \times 20$ and a needs not be considered. That problem occurs frequently in the development of embedded system, because developers apt to declare some dummy variables in the configuration code carelessly. To mitigate safety critical problems, these constraints should be modified.

4 Applying Automatic Inspection to DPS

DPS(Driver Positioning System) is an embedded software for controlling the seat adjustment functionality of a car, which is developed by ETRI and consists of two ECU(Electronic Control Unit) for driver seat module and steering column module, respectively. This system is based on AUTOSAR (AUTomotive Open System ARchitecture) that is an open and standardized automotive software architecture, jointly developed by automobile manufacturers, suppliers and tool developers[11].

AUTOSAR CT(conformance test)s check whether a variant of a BSW module or cluster (the test object) implemented by one or another product supplier, complies with relevant AUTOSAR specifications. AUTOSAR CTs verify whether the object under test satisfies selected conditions imposed in the relevant specification documents[12]. AUTOSAR CT Specifications are a part of the formally released AUTOSAR document set, and have a strict relationship to the relevant AUTOSAR specification documents. AUTOSAR CT can be applied to test objects on several levels of integration of BSW modules. As the size of the test objects increases from modules to subsystems to complete system including the RTE. Currently, AUTOSAR CT focus on individual modules[13].

Each CTSpec of BSW modules includes testcases not only for the correct functionality of test objects in terms of

their public input/output behavior in typical, but also for the correct configuration. For the behavioral testing, concrete testcases and methods are proposed with TTCN-3 language. But for static testing, it gives natural-language testcases and just recommends inspection of source code without any methods. In this point of view, our method is one of practical methods to inspect the violation of configuration. The automatic configuration inspection to DPS was performed on the development machine. AUTOSAR BSW modules used in DPS are 29 modules, in which 19 modules are examined with 113 testcases for configuration inspection. Figure 3 shows the snapshot of inspection tool and presents the violation out of configuration within CanNM modules[14]. Fortunately detected configuration error is not real error to lead the system to bad situation, but caused by dummy data in the configuration. One of violated constraints is as follow:

If a Network Management PDU is received the CanNm Message Cycle Timer is reloaded with the node specific timing parameter CANNM_MSG_REDUCED_TIME. The node specific time CANNM_MSG_REDUCED_TIME should be greater than ½ CANNM_MSG_CYCLE_TIME and less than CANNM_MSG_CYCLE_TIME.

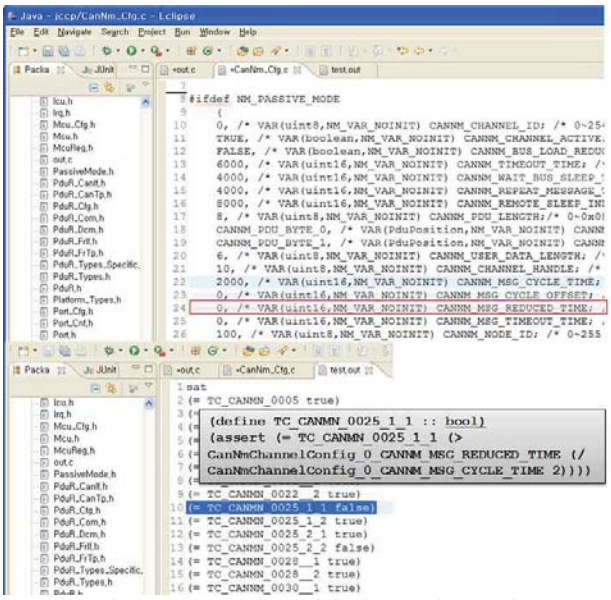


Figure 3. Snapshot of the inspection tool

5 Conclusions and Future researches

To overcome the problem of development of large scale embedded system, the Model Driven Development is accepted and vast configuration data are used, which gives flexibility to the development. To do automatic configuration inspection, we introduce lightweight formal approach using SMT solver. We proposed the translation algorithm and developed prototype tool as eclipse plug-in.

In the case AUTOSAR CT, among 6000, 4500 testcases are included in static testing, some part of them are about configuration inspection[15]. It is very hard to inspect configuration files by testers who are not involved in development process, because the files are generated by IDE tool and written with various macros. Therefore this automation of inspection is very useful for developers and testers. Formalization of requirements is helpful to establish correctness of requirements and check the inconsistence among requirements. In this paper, although our approach is applied to automotive system, we have plans to apply this method to check configuration constraints in ARINC 653 standard in the near future. Another future research is how to systematically translate requirements in a specification document to logical expressions. We expect that the requirement patterns can be applied.

References

[1] AUTOSAR, http://www.autosar.org

[2] Airlines Electronic Engineering Committee, Avionics Application Software Standard Interface – ARINC Specification 653 – Part 1, Aeronautical Radio Inc., 2006.

[3] A. Hrvath and D. Varro, “Model-Driven Development of ARINC 653 Configuration Tables,” In the Proceedings of the 29th Digital Avionics Systems Conference (DASC), 2010.

[4] ISO/DIS 26262, Road vehicles – Functional Safety – Part 1 to Part 8, 2009.

[5] DO-178B, Software Considerations in Airborne Systems and Equipment Certification, RTCA, Incorporated, 1992.

[6] S. Liu, F. Nagoya, Y. Chen, M. Goya, and J.A. McDermid, "An Automated Approach to Specification-Based Program Inspection," In the Proceedings of ICFEM 2005, LNCS 3785, pp. 421-434, 2005.

[7] L. Moura and N. Bjørner, "Satisfiability Modulo Theories: An Appetizer," invited paper to SBMF 2009.

[8] B. Dutertre and L. Moura, "A Fast Linear-Arithmetic Solver for DPLL(T)," In Proceedings of the 18th Computer-Aided Verification conference, pp. 81-94, 2006.

[9] A. R. Bradley and Z. Manna, The Calculus of Computation: Decision Procedures with Applications to Verification, Springer, 2007.

[10] Yices solver, http://yices.csl.sri.com

[11] AUTOSAR, AUTOSAR technical overview, 13.05.2010.

[12] AUTOSAR, AUTOSR BSW & RTE Conformance Test Specification Part 2: Process Overview, 30.11.2009.

[13] AUTOSAR, AUTOSR BSW & RTE Conformance Test Specification Part 3: Creation & Validation, 30.11.2009.

[14] AUTOSR, AUTOSAR, Conformance Test Specification of CAN Network Management.

[15] A.Alain Gilberg et all, “Conformance Testing for the AUTOSAR Standard,” ERTS 2010.

Formalizing Reusable Aspect-Oriented Concurrency Control

Neelam Soundarajan, Derek Bronish, Raffi Khatchadourian
Computer Science and Engineering
Ohio State University
{neelam,bronish,khatchad}@cse.ohio-state.edu

Abstract

Java and its library provide powerful concurrency control mechanisms. However, their use can lead to similar synchronization code being scattered *across multiple classes, synchronization and functional code being* tangled *together, and similar code being duplicated in many applications. Aspect-oriented (AO) programming makes it possible to address these problems. The precise behavior of systems built using AO techniques can, however, be difficult to understand. We propose a specification approach to precisely express key concurrency and synchronization properties of such systems. We illustrate the approach with a simple example.*

1 Introduction

Java provides a number of concurrency control mechanisms that allow a designer to specify that particular methods of a class are *synchronized*. When a *thread* invokes such a method on an object, it is suspended until it acquires the *lock* associated with the object, then proceeds to execute the method, releasing the lock when the method finishes. A finer-grained version allows an individual statement (or block) of a method to synchronize on the *this* object or, possibly, a different object. However, use of these mechanisms can reduce the degree of concurrency considerably. To alleviate this, the Java library provides the `Lock` interface implemented by classes such as `ReentrantLock`. By using *lock* objects appropriately, one can regain concurrency. But, at the same time, their use introduces some problems. First, if concurrency was not foreseen or its degree not anticipated when a class was designed, the approach requires invasive changes to the class. Second, synchronization and functional code of a class are often *tangled* together. Third, similar synchronization code may be *scattered* across multiple classes since similar synchronization concerns may arise in each one.

Aspect-oriented (AO) techniques help address such problems. The synchronization code can be contained in an *aspect*, separating it from the functional code. Moreover, by defining the *pointcuts* in the aspect suitably, the aspect's *advice* can apply to methods of multiple classes, thereby meeting the synchronization needs of each class. Synchronization code that is common to different applications can be defined in *abstract* aspects with *subaspects* for each application specializing it for that application, thereby eliminating code duplication.

The precise behavior of systems built in this manner can, however, be difficult to understand and formal specifications can help. We develop an approach that allows us specify, in the form of *contracts*, important behaviors, especially concurrency behaviors, of abstract aspects; and, in the form of *subcontracts*, specify how subaspects specialize the contracts to achieve behavior specific to the given system. We illustrate the approach by applying it to a simple example based on one in [7].

In Section 2, we consider related work. In Section 3, we sketch the model underlying our specifications. In Section 4, we present our specification approach via a simple case-study. Section 5 concludes the paper.

2 Background

Several approaches have been developed for reasoning about sequential AO systems. Dantas and Walker [3] consider *harmless* advice that does not modify the behavior of *base-code*, i.e., the underlying program. In [6], we consider how to provide information to arrive at the "richer" behavior caused by the advice without reanalyzing base-code. Katz and Katz [5] consider similar *rely-guarantee* specs. Aldrich [1], and Griswold *et al.* [4] propose ways to minimize the effects of aspects on base-code. Zhao and Rinard [12] consider abstract aspects but the the types of abstractness they allow are limited. Xu *et al.* [10] review approaches to reasoning about concurrency. Long & Long [8] present a formal specification of Java concurrency. Yang & Poppleton [11] present a model checker for concurrent Java.

3 Model Of Multi-threaded Computation

Fig. 1 outlines, in RESOLVE [9] style, the main components of a model of multi-threaded computation in a language with reference semantics as in Java. A *shared heap state*, lines 1–3, represents the *objects* in the sys-

tem. A *thread*, lines 7–8, consists of its id, a boolean indicating if it is active, and its control-flow-state which, lines 4–6, represents the sequence of method calls made in the thread that have not yet completed. A *synchronized lock*, lines 9–11, consists of the id of the associated object, whether it is currenly locked and, if so, the id of the holding object, and two sets of threads waiting to execute, respectively, synchronized and unsynchronized methods on the object. The other type of lock (not for a specific object), lines 12–13, consists of a boolean indicating whether it is locked, if so the id of the object holding it, and the set of ids of the objects waiting for it. A *lock*, lines 14–15, is one of these two types; note that this model is inadequate for dealing with *read-write* locks which we do not consider. A *system*, lines 16–17, consists of a heap, its current threads, and its locks.

```
1 math type Heap_State is
2  partial function from ObjectId
3         to ObjectValue
4 math type Control_Flow_State is
5  sequence of tuple of
6   (m: method sig, obj: ObjectId)
7 math type Thread is tuple of
8   (id, active, cfs)
9 math type Synch_Lock is tuple of
10   (object, locked, holder,
11    waitersForSynch, waitersForUnsynch)
12 math type Other_Lock is tuple of
13   (locked, holder, waiters)
14 math type Lock is
15   Synch_Lock or Other_Lock
16 math type Threaded_System is tuple of
17   (heap, threads, locks)
18 constraint: l: Synch_Lock
19  l.locked ⇒ [l.holder.active
20  ∧ head(l.holder.cfs).obj=lId
21  ∧ head(l.holder.cfs).m is synchronized
22  ∧ ∀ t ∈ waitersForSynch:
23    not t.active ∧ head(t.cfs).obj=lId
24    ∧ head(t.cfs).m is synchronized]
```

Figure 1. Model of threaded system

The model must satisfy certain constraints, one of which, lines 18–27, is that if a `synch_Lock` is locked, the thread holding it must be `active`, must, as indicated by its `cfs`, be currently executing a synchronized method on the corresponding object, and threads whose id's are in `waitersForSynch` must be inactive and waiting to execute a method on the object.

4 Case Study

Due to lack of space, we present our specification approach via a simple case-study, in Fig. 2, based on the `Shape` example of [7]. Instances of the class represent shapes with `x`–, `y`–coordinates, and `height` and `width` which may be accessed using `get` methods; `move` methods allow us to move the shape; `magnify()` and `shrink()` to expand and contract it.

```
1 class Shape
2 {protected int x = 0, y = 0,
3   height = 5, width = 10;
4  public int getX() { return x; }
5  //getY(), getHeight() etc. similar
6  public void moveNorth() { y++; }
7  //moveSouth(), moveEast(), etc. similar
8  public void magnify() {
9   height=height+5; width=width+10;}
10 public void shrink() {
11  height=height-5; if(height<=0)height=5;
12  width=width-10;if(width<=0)width=10;}}
13
14 protected final Lock locLock =
15  new ReentrantLock();// lock for location
16 protected final Lock dimLock =
17  new ReentrantLock();// and dimension
18 public int getX() { int tx;
19  locLock.lock(); tx = x;
20  locLock.unlock(); return tx; }
21 //getY() similar; getHeight(),getWidth()
22 // similar but using dimLock
23 public void magnify() {
24  dimLock.lock();...;dimLock.unlock();}
25 //move methods similar but using locLock
```

Figure 2. Shape class and Split locks

Suppose the application has many threads accessing a `shape`. Making each method *synchronized* would prevent interference between the threads but minimize concurrency. Instead, we use *two locks*, lines 14–17, one for location, the other for dimension. The methods, lines 18–25, which access/modify the location, respectively dimension, acquire the former, respectively the latter, perform their work, and release the lock.

4.1 Split Lock Aspect

In the AO approach in Fig. 3, `_lock1`, `_lock2` serve the same purpose as `locLock`, `dimLock`; `fstSetOps()`/`secondSetOps()` pointcuts correspond to calls that should use `_lock1`/`_lock2`. The advice, lines 7-10, is the synchronization code. Fig. 4 defines the subaspect for *Shape*, achieving the same level of concurrency as before but without scattering/tangling.

Suppose now `shrink()` was changed as follows: if the object was shrunk below a certain size, move it to a more visible place by changing `x` and `y`; and suppose we left the subaspect as in Fig. 4. This system may behave acceptably in all test cases. Indeed, the only time it

```
1 public abstract aspect SplitLockAspect
2  perthis(fstSetOps() || scndSetOps()){
3  protected abstract pointcut fstSetOps();
4  protected abstract pointcut scndSetOps();
5  private Lock _lock1 = new ReentractLock();
6  private Lock _lock2 = new ReentractLock();
7  before():fstSetOps(){ _lock1.lock();}
8  after() :fstSetOps(){ _lock1.unlock();}
9  before():scndSetOps(){ _lock2.lock();}
10 after() :scndSetOps(){ _lock2.unlock();}
```

Figure 3. SplitLock Aspect

would misbehave is if one thread invoked, say, `getX()` on a `shape` at the same time another was executing `shrink()` on it, *and* the size of the `shape` was below our minimum, *and* the timing of the two executions resulted in a strange value being returned for `x`.

```
1 aspect ShapeSplitLockAspect
2      extends SplitLockAspect {
3  pointcut fstSetOps() :
4     execution(Shape.getX())
5   ||execution(Shape.getY())
6   ||execution(Shape.moveNorth())||...
7  protected pointcut scndSetOps() :
8   execution(Shape.getHeight())||...
9   ||execution(Shape.shrink())||...
```

Figure 4. Subaspect for Shape System

4.2 Contracts and Subcontracts

Consider an abstract aspect *AB*. We specify *AB* in an *aspect contract*. For *abstract* portions of *AB*, we will use *abstraction concepts* in the contract. These will not be *defined* in the contract but will be *used* in it. The contract will also impose certain constraints that definitions, provided in subaspects, for the concepts must satisfy.

Part of the `SplitLockAspect` contract appears in Fig. 5. Line 5 requires that the definition, in a subaspect, of `firstSetOps()` must be such that all methods whose execution join points match it must be from a *single* class. This represents that the lock is not intended to prevent simultaneous execution of methods from different classes. The next clause, for `scndSetOps()`, is similar. The next clause, line 6, requires that these two classes be the same. The next clause requires a given method to be mapped to (at most) one of these pointcuts. The subaspect in Fig. 4 can be easily seen to meet these requirements.

The constraint in lines 10–11 requires that the (union of the) set of objects accessed by the methods mapped to `firstSetOps` be disjoint from the set accessed by methods mapped to `secondSetOps`. But the information needed to check this constraint is *not* part of the subaspect in Fig. 4. The *abstraction concept*,

```
1 abstraction concepts:
2 set AccessedObjects(set mthds)
3  //objects accessed by methods in mthds
4 constraints:
5 [(|fstSetOps.Class|=|scndSetOps.Class|=1)
6 ∧(fstSetOps.Class=scndSetOps.Class)
7 ∧(firstSetOps.Mthds∩scndSetOps.Mthds=∅)]
8 AccessedObjects(S1 ∪ S2) =
9  AccessedObjects(S1)∪AccessedObjects(S2)
10 [(AccessedObjects(fstSetOps.Mthds)∩
11   AccessedObjects(scndSetOps.Mthds) = ∅)]
12 results:
13 [(m1∈fstSetOps.Mthds)∧(m2∈scndSetOps.Mthds)]
14       ⇒ [ nosuspension(m1,m2)
15           ∧ nosuspension(m2,m1)]
16 [(m1∈fstSetOps.Mthds)∧(m4∈ fstSetOps.Mthds)]
17       ⇒ noconcurrency(m1,m4)
18 [(m3∈scndSetOps.Mthds)∧(m2∈scndSetOps.Mthds)]
19       ⇒ noconcurrency(m3,m2)
```

Figure 5. Contract for SplitLock Aspect

`AccessedObjects` (line 2) (and its definition in the subcontract), help address this. The contract does not define it but the name (and associated comment) is suggestive: the set of objects accessed by the methods in `mthds`. The constraint in lines 10–11 uses it to capture the key requirement that the intersection between the sets of objects accessed respectively by the methods mapped to the two pointcuts be empty. Lines 8–9 impose a simple constraint on the definition, in the subcontract, of `AccessedObjects`: that it be distributive.

Lines (11-16) specify the results of using the aspect: if two methods are mapped to the two pointcuts, their respective executions will not suspend each other; but if they are mapped to the same pointcut, there will be no concurrency between their executions.

The subcontract `ShapeSplitLockAspect` needs to only define `AccessedObjects`. Since it is required, by the aspect contract, to be distributive, we can define it by specifying its value for each method mapped to the two pointcuts. This is easily done for our example; thus, for `getX()`, the value will be the set {`x`}; for `magnify()`, it will be {`width, height`}; etc.

Next we have to check that the contract's constraints are satisfied. It is at *this* point that we can locate the bug considered earlier where `shrink()` may modify `x` and `y`. Given this code, `AccessedObjects(shrink)` should be {`x, y, height, width`}; hence, the constraint in lines (8-9) is violated; thus, for example, `x` is in both `AccessedObjects(shrink)` and `AccessedObjects(getX)` and these two methods are mapped to the two pointcuts and the contract is violated.

Next, the definitions in the subaspect and subcon-

tract can be plugged into the aspect contract, in particular, into the *results* clauses of the contract to arrive at the specialized versions of these clauses that apply to this particular system. In the current case this will allow us to conclude, for example, from lines (15–16) of the contract, that if a thread was currently executing `getHeight()` on a shape object and another invoked `magnify()` on the same object, the latter would be suspended until the former finishes. To build another application in which `SplitLockAspect` could be used, we would only have to define the corresponding subaspect and subcontract, defining respectively the pointcuts and `AccessedObjects`; next, verify that, with these definitions, the constraints in Fig. 5 are satisfied; and, finally, arrive at the behavior of the application by plugging in the definitions into the results in Fig. 5.

How do we define **nosuspension**() and **noconcurrency**()? These are primitives provided by the formalism and are defined in terms of the model of Section 3. Thus **noconcurrency**(M), where M is a set of methods, means if `t1` and `t2` are two threads in the `threads` component of a system and `m1, m2` are elements of M, `m1` is in `cfs`, the control-flow-state of `t1`, `m2` is in `cfs` of `t2`, and the corresponding `obj` components are equal to each other, then the `active` component of `t2` must be *false* if that of `t1` is *true*. The precise set of primitives and their definitions is part of our current work. One other point is worth noting. Standard approaches to formalizing aspects typically make use of *pre-* and *post*-conditions. Why doesn't our contract for `SplitLockAspect` involve these? The answer is that our focus is on *concurrency* issues. Thus, for example, the constraint expressed in lines (8–9) of Fig. 5 can be compared with the notion of *interference freedom* [10]. Our clause, by requiring that the intersection of the sets of objects accessed by the two groups of methods to be empty, ensures that no method in the first group will interfere with any method in the second group.

5 Discussion

Our goal was to show how common synchronization patterns may be implemented using abstract aspects that are then specialized, using subaspects, for individual systems; and develop an approach to specify essential properties of the aspects and subaspects. Although the question of and reasoning about AO programs has received much attention, the problem of abstract aspects, especially for concurrency behaviors, and their specializations has not, to our knowledge, been addressed.

Our contracts and subcontracts had to refer to more than just the class variables. Hence we defined a model that provided a view of the threads that exist at runtime. And in our specs, we referred to components of this model, allowing us to specify the concurrency behaviors of interest. Importantly, the use of *abstraction concepts*, in conjunction with the constraints imposed on them, allowed us to represent the *intent* of the abstract aspect.

In Section 4, we relied on intuitive reasoning to show that the subaspect and its subcontract satisfy the aspect contract's requirements. For more complex systems, tool support would be needed. Bodden and Havelund [2] describe an AO algorithm, *Racer*, that uses new *concurrency-related* pointcuts that may be used to identify concurrency bugs. It should be possible to use a *Racer*-like approach to build a tool that will spot violations of our contracts/subcontracts and we plan to pursue this in future work. Along a different line, our model may help identify additional AO primitives that will be of use in writing concurrent programs; for example, primitives that allow the programmer access to information about the threads in the system may be useful. These would be analogous to such primitives as *cflow* but tuned to the needs of concurrency behaviors.

References

[1] J. Aldrich. Open modules: Modular reasoning about advice. In *Proc. of ECOOP*, pages 144–168. Springer, 2005.

[2] E. Bodden and K. Havelund. Racer: Race detection using "AspectJ". In *ISSTA*, pages 155–166. ACM, 2008.

[3] D. Dantas and D. Walker. Harmless advice. In *POPL '06*, pages 383–396. ACM, 2006.

[4] W. Griswold, K. Sullivan, Y. Song, M. Shonle, N. Tewari, Y. Cai, and H. Rajan. Mod. softw. des. with crosscutting interfaces. *IEEE Softw.*, 23(1):51–60, 2006.

[5] E. Katz and S. Katz. Incremental analysis of interference among aspects. In *Wkshp. on fnds. of AO langs.*, 2008.

[6] R. Khatchadourian, J. Dovland, and N. Soundarajan. Enforcing behavioral constraints in evolving AO programs. In *Wkshp. on fnds. of AO langs.*, 2008.

[7] D. Lea. *Concurrent Programming in Java, Second Edition*. Addison-Wesley, 2000.

[8] B. Long and B. Long. Formal specification of java concurrency to assist software verification. In *Int. Symp. on Par. and Dist. Processing*. IEEE-CS, 2003.

[9] M. Sitaraman and B. Weide. Component-based software using "RESOLVE". *Software Eng. Notes*, 19(4):21–63, 1994.

[10] Q. Xu, W. de Roever, and J. He. Rely-guarantee method for verifying shared variable concurrent programs. *Formal Aspects of Computing*, 9(2):149–174, 1997.

[11] L. Yang and M. Poppleton. Jcsprob: Implementing integrated formal specifications in concurrent java. In *CPA*, pages 67–88, 2007.

[12] J. Zhao and M. Rinard. Pipa: Behavioral interface for "AspectJ". In *FASE*, pages 150–165. Springer, 2003.

PIPE+ - A Modeling Tool for High Level Petri Nets

Su Liu, Reng Zeng, Xudong He
School of Computing and Information Sciences
Florida International University
Miami, Florida 33199, USA
{sliu002, rzeng001, hex}@cs.fiu.edu

Abstract— Petri nets are a formal, graphical and executable modeling technique for the specification and analysis of concurrent systems and have been widely applied in computer science and many other engineering disciplines. Low level Petri nets are simple and useful for modeling control flows; however, they are not powerful to define data and system functionality. High level Petri nets were developed to support data and functionality definitions [1]. To support the practical applications of Petri nets formalism, tools for designing and executing Petri nets are necessary. Although there are many existing tools for supporting low level Petri nets [5], few tools are available for high level Petri nets. There is especially a lack of tools to support high level Petri net notation proposed in the international standard [1]. In this paper, we present a tool, called PIPE+*, to support a subset of high level Petri nets proposed in [1]. PIPE+ is built upon an existing low level Petri net tool PIPE (Platform Independent Petri Net Editor) [2]. This paper describes the functionality of PIPE+ as well as illustrates the process of extending PIPE, which provides helpful insights for others to create Petri net tools suit their own needs. Furthermore, PIPE+ is an open source tool and thus is available for various enhancements from worldwide research community.

Keywords: Petri Net; Modeling Tools

I. INTRODUCTION

Petri nets have been used to describe a wide range of systems since their invention in 1962 [3]. They are a graphical and formal method for describing and studying concurrent and distributed systems [4]. Low level Petri nets are suitable to model control flows and are applicable to work flow systems; however are not adequate to describe complex systems. High level Petri nets (HLPNs) were developed to support the definition of data and functional processing [1]. To support the practical applications of Petri net formalism, tools for creating and executing Petri nets are needed. In [5], a Petri net tool database listed the tools developed in the past several decades. Unfortunately, many of the tools described in the database as well as in literature are no longer maintained or available and few of them support HLPNs, especially the HLPN definitions and notations proposed in the 2001 international standard [1]. Table 1 lists representative tools supporting some forms of HLPNs.

The HLPNs proposed in the international standards draws concepts from predicate transition nets, colored Petri nets, and algebraic Petri nets; and provides abstract and general definitions and notions. It is desirable to create a tool for editing and simulating the HLPNs; unfortunately no such tool exists yet. A realistic attempt is to develop a tool to support a restricted and concrete realization of HLPNs. The most critical components of the HLPN definitions are the net annotations with regard to transitions, which are algebraic terms of Boolean type. In [15], we viewed first order logic formulas as the algebraic terms associated with transitions and thus adopted the predicate transition net view as a concrete realization of HLPNs.

Table I A List of High Level Petri net Tools

Name	High level Net Type	Graphical Editor	Simulator
AlPiNA	Algebraic Petri Nets	Yes	No
CoopnBuilder	CO-OPN language	Yes	No
CPN Tools	Colored Petri Nets	Yes	Yes
HISIm	Hybrid Petri Nets	Yes	Yes
Renew	Object Oriented Petri nets	Yes	Yes
PIPE+	High level Petri nets	Yes	Yes

It requires tremendous effort to build a HLPN tool from scratch and is especially difficult to assure the quality of the initial design. It is desirable to leverage successful results and extend mature and existing tools. It is of tremendous value to build upon open source tools so that the resulting new tool can be shared and improved by the worldwide research community. PIPE+* is a tool to support HLPN where transition conditions are defined in terms of first order logic formulas [14]. PIPE+ extends a well developed open source low level Petri net tool called PIPE (Platform Independent Petri net Editor) [6]. Furthermore PIPE+ retains the original low level Petri net editing and executing features, which allows user to choose appropriate net levels to model target systems.

In this paper, we first briefly review the background of basic and HLPN concepts. We introduce the chosen low level Petri net tool PIPE that PIPE+ built on, and then present implementation details of the extension according to

* PIPE+ can be downloaded at http://www.cis.fiu.edu/pub/PIPEplus/

the HLPN concepts. We discuss limitations and applications of the tool, and our contributions and perspectives.

II. PETRI NETS AND HIGH LEVEL PETRI NETS

The Petri net structure consists of a finite set of places (drawn as circles), a finite set of transitions (drawn as bars), a finite set of directed arcs (drawn as arrows), and a set of tokens (drawn as dots) to define an initial marking. The arcs connect from a place to a transition or vice versa, never between places or between transitions. The places from which an arc runs to a transition are called the input places of the transition; the places to which arcs run from a transition are called the output places of the transition. The places can contain multiple tokens and thus are of multi set type (or bag). A distribution of tokens over the places of a net is called a marking. A transition may fire whenever there are enough tokens in all input places.

According to the international standard [1], a HLPN graph comprises: a net graph, place types, place marking, arc annotations, transition condition and declarations. The net graph is the net structure; place types are non-empty sets, restrict the data structure of tokens in the place; place markings are collection of elements (data items) associated with places, called tokens; arc annotations are inscribed with expressions which may comprise constants variables (e.g., x, y) and function images (e.g., f(x)); transition conditions are Boolean expressions inscribed in; declarations comprising definitions of place types, typing of variables and function definitions. For net execution, the most important is transition enabling. Enabling a transition involves the marking of its input places. When an enabled transition occurs, the enabling tokens from input place's are subtracted and the resulting tokens of the transition Boolean expression are added to the output places.

III. AN OVERVIEW OF PIPE

PIPE [2] is a Platform Independent Petri net Editor to edit, animate and analyze low level Petri nets, which has clear design and incorporates the latest XML Petri net standards of storing format, the Petri Net Markup Language (PNML). It is implemented in Java and can be logically divided into three major components [6], shown in Figure 1: the graphical user interface (GUI), a layer managing the interactions between the GUI and the modules (DataLayer), and analysis modules.

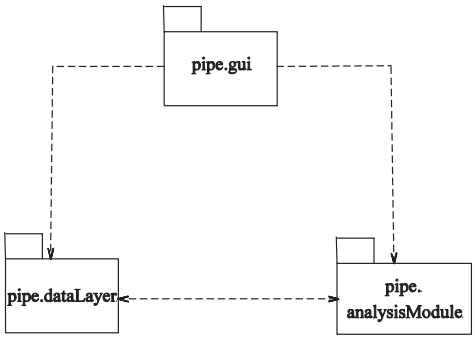


Figure 1 Package Diagram for PIPE

A. Graphical User Interface

PIPE's graphical user interface is developed using Java Swing API as it provides full GUI functionalities and mimics the platform it runs on. Besides, as PIPE is a cross platform application this was deemed useful for providing a native look and feel. The GUI component includes GUIFrame, GUIView and classes such as action, handler and widgets supporting Swing APIs. From a user perspective, there are two major parts: Editor and Simulator.

- ***Editor***: Users are able to edit a low level Petri net by clicking and drawing Petri net graphical elements through the menu bar, toolbar. On the toolbar, it lists all the Petri net element thumbnails, such as place, transition and arc, which can be selected and added to the white canvas (tabbed pane) of the editor. Besides, these added elements' annotations and attributes can be defined by selecting one of the elements and pop up an editing dialog box.
- ***Simulator***: There is a switcher button between editor mode and simulation mode. Using the simulator, a user is able to fire a random transition or fire a number of transitions randomly selected among enabled ones. The simulation process includes subtracting tokens from input places and adding them to output places while firing a transition. Besides, the animation history is displayed on the left bottom of the interface frame by listing transition's label orderly.

B. Internal Architecture of PIPE—The DataLayer

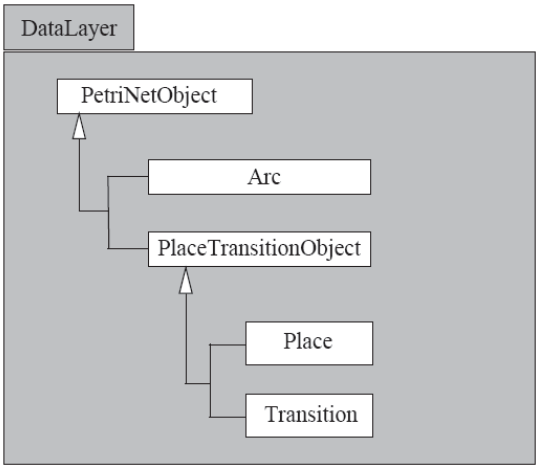


Figure 2 The Hierarchy of PetriNetOjbect Classes

The core component of PIPE is the data layer, which maintains states and contains all the classes used to represent a Petri net. Figure 2 shows the hierarchy of important Petri net object classes [6], including Arc, Place and Transition classes inherited from PetriNetObject because they have common variables and methods, such as id, name, location, etc.

In the data layer component, each Petri net is encapsulated by an instance of the DataLayer class, which contains all the Petri net objects stored in a list enabling the

easy addition of new objects. It contains not only methods to access all its internal objects and to return its internal lists, but also methods to calculate the current markup, initial markup, forwards incidence matrix, backwards incidence matrix, combined incidence matrix and enabled transitions.

Besides data layer, PIPE has analysis module to do analysis and conclusions on the properties of Petri net model, such as boundedness, liveness, reachable markings and so on.

C. *Saving and Loading*

PIPE is capable of saving and loading nets and writing the Petri net data layer into a Petri net Markup Language (PNML). An Extensible Stylesheet Language Transformation (XSLT) is used to transform it between PNML and XML files.

IV. PIPE+

A. *Overview of the Extension*

Similar to PIPE, PIPE+ is also an editor and a simulator. The editor is to model a system visually through a graphical interface. The goal is to utilize all the benefits that a HLPN provided with convenience. The details are presented below according to the HLPN concept's six elements in reference [1]. The simulator is no longer a simple black dot token animation game but to manage the movement of meaningful data. We developed a mandatory compiler with an interpreter to process token data inside transition conditions, which are defined using restricted first-order logic. Besides, a simulation algorithm is applied to ensure its fairness and improve its performance.

B. *A Net Graph*

Since the graphical elements of a HLPN are the same as low level ones, PIPE's graphical editor is retained.

C. *Place Type and Place Marking*

The main difference between high level and low level Petri nets is that tokens are no longer black dots, but complex structured data. Place types are non-empty sets that restrict the data structure of tokens in the places. The data structure is an array of basic types, such as integer and string, and defined by user. For example, assuming a log in user account as a token has two elements, username and password, which are represented by two basic data types, string and integer. In a HLPN's place, a place data type is inscribed to restrict the data structure of tokens. In another way, the data type of tokens can be added into the place has been already defined beforehand.

To implement the concept that tokens with data structure, a data storage system is needed. Based on PIPE, the data layer package is modified by adding three classes: DataType, Token, abToken (Figure 3).

- DataType: The main data structure in class DataType is a list storing basic types' name, which is used to show what data structure the token or place holds. The data structure consists of an array of basic types, such as string, integer, etc. For our tool, basic types are limited to strings and integers for the simplicity but are adequate for most of applications. For the convenience of extension on basic types, we introduce a new structure BasicType to data layer. The structure BasicType (see Figure 4) includes a flag data field "Kind" to indicate which type it is (in PIPE+, 0 represents integer, 1 represents string). Space is allocated to both integer and string since it is undecided before the "Kind" is defined. Further extension on basic types needs to enhance the class BasicType by allocating extra space and redefine "Kind".

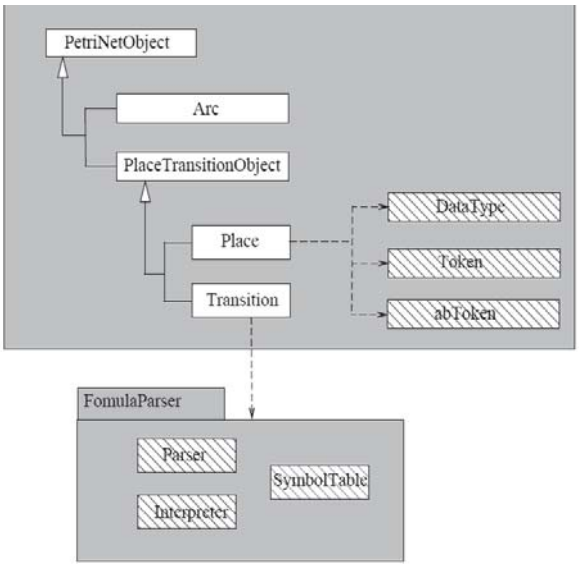


Figure 3 Extensions on DataLayer for PIPE+

- Token: Class Token is added to the data layer to maintain data value. The important field is a list storing instances of value with type of the BasicType, see Figure 4. Token is a basic data storage element in the places and its value is calculated by the transitions. The simulation process is fetching data value from the token's BasicType and fill the calculated result value to another token's BasicType.
- Abstract Token: Since first-order logic covers quantification, the whole collection of tokens in a place need to be checked by transition condition expressions. For example, if an expression includes "$\exists x \in X$", all the tokens in "X" needs to be checked to see whether a "x" exists, so the whole collection of tokens is fetched while checking enabledness of a transition. The tokens in this type of place are defined as a power set. A new class abToken (abstract token) is added into the data layer to store the power set. It has a field storing a list of regular tokens with the same data type, so it also has a data type to restrict the tokens data structure. We flatten the nested power sets by duplicating some fields.

For example, in a library system, one user may borrow a list of books, so that the database (power set) in library system is {username, password, books_borrowed{book1, book2,...} } is converted into {username, password, book1}, {username, password, book2}. This design sacrifices the space for the convenience of implementation, which can be further improved.

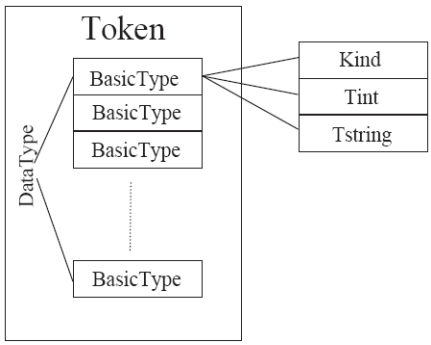


Figure 4 Structure of Class Token

As a result, the places in PIPE+ stores a list of regular tokens or an abstract token that contains a collection of regular tokens. Whether the connected transition can fetch a regular token or an abstract token depends on the place is a power set or not. The user can add, edit and delete tokens from places to create a net marking.

In PIPE+, a place stores tokens by List container, the place's capacity is built as unbounded (remember it has nothing to do with the number of different tokens that may appear in a particular place). However, in the discussion of [7], bounded and unbounded places have the same expressive power. A bounded place is preferable for the reason of visualization and redundancy.

In PIPE+, copies of token are allowed to store in the same place. Since whether the place needs to remove its copies of token depends on what the model it is, this can be further improved by supporting an option of copy remove.

D. Transition Conditions and Arc Annotations

Transition conditions are guards controlling the flowing of the tokens. PIPE+ use first-order logic to define transition condition formulas, which, syntactically, consists of variables and logic operators. Variables in the formula are predicates that can be instantiated by value from input tokens. Combined with logic operators the formula can be calculated. Semantically, as transition is a guard to control token flows, it has to check the value of tokens from input places and formulate new tokens conform to the output place type, the formula consists of two parts: pre-condition and post-condition. However, in PIPE+, the user is not supposed to separate the two conditions explicitly, because the interpreter can differentiate them by the type of variables.

In PIPE+, arc annotations are variables to assist transition expression calculation by mapping token values to expression's predicate variables. Arc variables are restricted to be appeared in the connected transition expression's variables for the mapping. Since a transition is connected by input and output arcs and arcs are connected to places, the predicate variables in the transition expressions are classed into input variables and output variables. For example, in Figure 5, a and b are input variables while c is output variable.

In a transition calculation process shown in Figure 5: In step (1), each token in the connected place is firstly bounded to the connected arc variable; as a pair, {variable, token}, they are fetched into a symbol table of the transition (note the pair with output variable's token value is temporarily empty and to be filled by the result of the expression calculation). In step (2), the input variables in the transition expression can locate token value through the pair's arc variables by looking up symbol table. In step (3), after transition expression calculation, the output variables are assigned with result value and the symbol table's output variable pairs are filled by the value. In step (4), the output pairs' token are added to the connected output places according to the arc's variables. For example, as c is on the output arc, c's token in symbol table [bob] is added to the output place.

Unless the transition formula cannot be satisfied by the value of the tokens fetched into symbol table, the tokens from input places (both regular token and abstract token) are consumed. However, a power set place with an abstract token usually has a backward arrow that makes it an output place as well, so the abstract token is returned to the place according to the post condition of the transition formula. In the case when the formula is unsatisfied, the currently fetched tokens in the symbol table are not consumed, so they are returned to the input places.

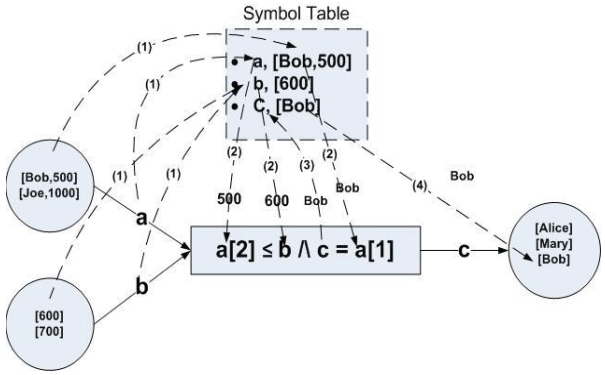


Figure 5 An Enabled Transition Formula Calculation Process

- *Restricted First Order Logic Transition Formula Expression*: In PIPE+, it is called restricted because the grammar we built for the tool has limitations. Since each predicate variable has to be instantiated, the user cannot use free variable that does not appear in the arc annotation, otherwise the calculation result is undetermined. Also, it does not support predefined function, like f(n), since the meaning of the function has to be declared beforehand, which is equivalent to define its

operations in a single logical sentence by using the connecting operator " ∧ ", which simplifies the implementation of expression interpreter. However, the restricted version of first order logic is still very powerful, because it does support complex expressions, such as:

$$(a = b) \wedge \exists c \in C((c[1] > a[2]) \wedge (C' = C - \{b\} \cup \{[a[1], c[2]]\})). \quad (1)$$

In (1), lower case letters represent regular tokens, upper case represent power set; *C*' by convention represents output variables and also is a power set (upper letters); it further indicates the clause is a post-condition because output variables at the left side of the equation means assignment; $c[n]$ means the n^{th} element value in *c*'s data structure.

- *Parser and Interpreter*: Because logical formulas need to be parsed and interpreted, we build a compiler with a parser and an interpreter for the restricted first order logic formula. The parser includes a scanner, which is built by a lex file and generated by jflex 1.4.3 [16]. A BNF grammar is built in cup file and generated by leveraging the tool jcup v11 [17]. Since the transition formula does not explicitly separate pre and post conditions, but only pre-conditions need to be calculated when checking whether the transition is enabled to fire or not, the interpreter has to differentiate pre and post conditions. A trick is found that in the post-condition, it usually starts with an output variable equals a subformula, for example, in (1),$(C' = C - \{b\} \cup \{[a[1], c[2]\})$ is a post-condition because $C'$is output variable. Therefore in the interpreter, when checking a clause with an "=" operator, the left hand side of the "=" variable is checked. If it is input variable, this clause is a pre-condition and the "=" is interpreted as a logic operator equal, which results in a Boolean true or false; on the other hand, if it is an output variable, the clause becomes a post-condition and the operator now is an assignment clause that assign the result value of right hand side formula to the left hand side output variable.
- Symbol Table: In PIPE+ each transition maintains a temporary symbol table to facilitate the interpreter. It does not use one big table for all the transitions, because it may cause name conflict and is hard to manage. The symbol table contains a list of elements that are structured by a pair of key and object. The pair of key and object is obtained from the transition's connected arc annotation and place. The reason we maintain the pair of key and object instead of key and token is because besides regular token type, the key may pair with a power set (abstract token type). Moreover, the symbol table is initiated each time before a logical formula is checked and cleared after the firing process.
- Declarations: In the standard[1], it comprising definitions of place types, typing of variables and function definitions. In PIPE+, the declarations are already in the modeling process by defining place data types, transition condition formulas and arc annotations.

E. Extensions On GUI

The GUI package in PIPE mainly consists of a GUIFrame, a GUIView, and some supporting classes. The GUIFrame is PIPE's graphical frame includes a menu, a toolbar and a statusbar. The GUIView is the panel to draw Petri net graphical elements. Since requirements and concepts for HLPNs are token storage and flow, our modification to PIPE's GUI is focused on Petri net elements places, transitions and arcs. The common procedure to extend PIPE's GUI is adding new selections on graphical elements' property setting menu for new features. In PIPE+, after modifying the gui.handler package for each Petri net element class, the new selections are shown in a popup menu by right clicking a Petri net element. The places now have the choices of defining data type and editing tokens; the transitions can contain logical formulas; the arcs can be labeled by variable key. These new features are triggered by additional selections on GUI and used through customized panels or dialogs.

F. Simulator

The simulator not only needs to execute the net model visually, but also has to ensure correctness, fairness and good performance. In PIPE+, the HLPN simulator designs as follows:

1) Graphical Simulation: Since in a low level Petri net, tokens are just black dots flowing from one place to another and the animation is visible to the user. In contrast, tokens in HLPNs are complex structured data, and especially when the number of tokens is large, which are inappropriate to be displayed upon graphical net; otherwise the graphical annotations are unreadable. Since the execution procedure is invisible to a user, the result can only be checked by looking into the contents of Places. In PIPE+, to view the tokens in the Places, user can open the Place edit panel and the value of tokens are displayed under the text area of Token List. Besides, the firing history is retained from PIPE by listing the fired transition name orderly and updated instantly after a transition fires, thus the user clearly knows a transition is fired.

2) Transition Occurrence Scheduling Algorithm: A scheduler is needed to coordinate the simulator's token flow strategy efficiently. Since the performance of the simulator mostly affected by the times of transition condition calculation, PIPE+ chooses the scheduling algorithm from [9] to minimize the recalculation of transition condition checking. The idea is to keep track of disabled transitions discovered during the search of enabled transitions, and use the locality principle, that is an occurring transition only affects the marking on immediate neighboring places, and hence the enabling of a limited set of neighbor transitions. For the implementation, we

maintain an unknown list and a disabled list. All transitions initialized as unknowns will be randomly picked and checked for enabling status. If the status is disabled, the transition will be moved to disabled list. Upon occurrence of a transition, we update the status of neighboring transitions to the unknown list if they are in the disabled list. The neighboring transitions can be found through occurred transitions' output places. Therefore, the disabled transition avoid recalculation if the tokens of its input places are not changed.

3) Enabling a Single Transition: In the HLPN concpets, tokens are meaningful data, when a selected transition start to check its expression, the expression's variables are to be instantiated. Since a transition may connect to a number of input places, where each place contains a list of tokens, to see whether the transition is enabled or disabled, it has to check all the possible combinations of instantiation tokens from its input places. For example, if there are three input places and each place has 3 tokens, the number of their combinations is $3 \times 3 \times 3 = 27$. If one of the three input places is a power set, no matter how many regular tokens inside the abstract token, it only counts as one abstract token. So the combinations reduces to $3 \times 3 \times 1 = 9$ combinations.

4) An Summarization of the Complete Internal Simulation Process:

a) All transitions in the net graph are initially stored in an unknown list; a disabled list is initialized to be empty;

b) A transition is randomly selected from the unknown list, and is checked for enabledness;

c) During the checking process of the selected transition, all the connected arcs and places of the transition are found;

d) Combinations of tokens from the transition's input places are orderly choosen to fill in its symbol table. Since symbols in symbol table are pairs of [key, object]. The keys are from arcs label; the objects are regular tokens. If the input place is a power set, the whole abstract token is sent as an object, otherwise only its first token is sent and the remaining tokens are still in place. For the symbol's key from output arcs, the object is empty because it is to be filled during transition firing action (after interpreting the post condition of transition formula);

e) The formula expression in the transition is checked utilizing a parser. A Boolean value is returned: if it is true, the transition is enabled and is fired immediately; if it is false, the transition is not enabled with the current input tokens, the tokens in symbol table will go back to the input places; if all the combination of input tokens cannot enable the transition, the transition is moved into a disabled list Both checking and firing a transition formula needs to parse and interpret the formula; however, the checking process only affects a formula's pre-condition while the firing process only affects a formula's post-condition.

f) After firing the transition, the tokens in the symbol table are sent to the output places according to the variables of arcs annotation and added to the tail of output places' token list. Since it changes the place marking of the output places, according to the scheduler algorithm's locality principle, if the dependent transitions are in the disabled list, it can now be moved back to the unknown list. Then go back to step b).

g) In step b, when unknown list is empty, the simulation process ended.

V. SOME ISSUES OF PIPE+

A. Limitations of PIPE+

1) Limited Basic Types: As we mentioned above, currently, the place data type of PIPE+ only supports two basic types, string and integer. Since PIPE+ using a structure to define basic types, the structure can be extended to accommodate more types.

2) Flat Tokens: For the convinience of implementation, the place data type of PIPE+ does not support nested powerset, such as {Bob, {book1, book2}}, but instead, it stores two flat tokens {Bob, book1}, {Bob, book2}.

3) Restricted First-order Logic for Transition Formula: A new grammar is built for the convenience of interpretation and to avoid ambiguity.

4) No True Concurrency: PIPE+ only supports interleaving semantics. Besides, it does not support timed Petri nets.

5) Randomly Fire Transition: The selection of the next transition to fire is performed randomly rather than choosen by the user.

6) Analysis Module: Lack of an integrated tool to analyze the properties of a net model;

7) Bugs and Errors: Since this is the first version of PIPE+ and our main purpose is to introduce the new tool, bugs are unavoidable.

B. Testing PIPE+

The most important part of testing is the transition condition formula. As the new parser and the interpreter were built for the restricted first-order logic formula, its correctness has to be assured. Our test cases are designed mainly on complex formulas including quantifier, relation expressions, arithmetic expressions and set expressions.

C. Compared to CPN Tools

CPN Tools [18] is a widely used industrial strength tool for constructing and analyzing colored Petri nets [19]. The main difference between CPN Tools and PIPE+ is the underlying specifying languages. Colored Petri nets utilize a functional programming language standard ML [12] while PIPE+ utilizes first-order logic formula. Since first order logic is well known, it is easier to use PIPE+. However CPN Tools provide more functionality, especially with regard to analysis.

D. Using PIPE+

PIPE+ has been applied to a Mondex[8] smart card system, which is an electronic purse payment system based on smart card technology. The model for a concrete transaction between two purses has eight operations (including abort) and four statuses, and we translated into PIPE+ model with ten transitions and four places. Figure 6 is a screenshot of Mondex in PIPE+, in which the simulation of a transition firing sequence is shown at the left bottom of the interface's frame. After no more transition is available to fire, the result of the simulation is a final marking that can be read by opening the places, msg_out and ConPurse, to view contents which are tokens' data.

VI. CONCLUSIONS

In this paper, we present a tool PIPE+ supporting high level Petri nets editing and simulation. We believe PIPE+ can be a valuable tool for concurrent and distributed system modeling and simulation. PIPE+ is built upon an open source tool PIPE for low level Petri nets. We illustrated the process of extending PIPE, and discussed our design strategies, which provide helpful insights for others to create Petri net tools suit their own needs. Furthermore, PIPE+ is an open source tool and thus is available for sharing and continuous enhancements from worldwide research community.

Acknowledgements This work was partially supported by NSF grants HRD-0833093.

REFERENCES

[1] High-level Petri Nets-Concepts, Definitions and Graphical Notation, Version 4.7.1, 2000

[2] Pere Bonet, Catalina M. Llado, Ramon Puigjaner, "PIPE v2.5: a Petri Net tool for performance modeling," Proc. 23rd Latin American Conference on Informatics (CLEI 2007), San Jose, Costa Rica, October 2007

[3] Reisig, Wolfgang, "Petri nets: an introduction," Springer-Verlag New York, Inc.NY, 1985

[4] Tadao Murata, "Petri Nets: Properties, Analysis and Applications," Proceedings of IEEE, vol. 77 No.4, Chicago, IL, April 1988

[5] Petri Net Tool Database. http://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/db.html

[6] James Bloom, Clare Clark, Camilla Clifford, Alex Duncan, Haroun Khan, Manos Papantoniou, "Platform Independent Petri-net Editor: Final Report," London, March 2003

[7] Carlos A. Heuser, Gernot Richter, "Constructs for Modeling Information Systems with Petri Nets," 13th International Conference on Application and Theory of Petri Nets, 1992, Sheffield, UK

[8] Reng Zeng, Xudong He, "A Formal Specification of Mondex Using SAM," The Fourth IEEE International Symposium on Service-Oriented System Engineering, 2008

[9] Kjeld H. Mortensen, "Efficient Data-Structures and Algorithms for a Coloured Petri Nets Simulator," 3rd Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools, Aarhus University, August 2001

[10] Didier Buchs, Steve Hostettler, Alexis Marechal, and Matteo Risoldi, "AlPiNA: An Algebraic Petri Net Analyzer," J. Esparza and R. Majumdar (Eds.): TACAS 2010, LNCS 6015, pp. 349–352, 2010

[11] A.V. Ratzer, L. Wells, H.M. Lassen, M. Laursen, J.F. Qvortrup, M.S. Stissing, M. Westergaard, S. Christensen, and K. Jensen, "CPN Tools for Editing, Simulating, and Analysing Coloured Petri Nets," Proc. of 24th International Conference on Applications and Theory of Petri Nets , 2003

[12] R. Milner, M. Tofte, R. Harper, and D. MacQueen, "The definition of Standard ML," MIT Press, Cambridge, MA, 1997

[13] CPN ML Reference, http://www.daimi.au.dk/designCPN/man/Reference/Reference.Main3.CPN.ML.pdf

[14] Andrews, Peter, "An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof," 2nd ed. Kluwer Academic Publishers, 2002.

[15] X. He and T. Murata: "High-Level Petri Nets – Extensions, Analysis, and Applications", Electrical Engineering Handbook (ed. Wai-Kai Chen), Elsevier Academic Press, 2005, 459-476.

[16] JFlex Lexical Analyzer Generator. http://jflex.de/index.html

[17] JCUP Parser Generator. http://www2.cs.tum.edu/projects/cup/

[18] CPN Tools. http://cpntools.org/

[19] K. Jensen, L.M. Kristensen, and L. Wells, "Coloured Petri nets and CPN tools for modelling and validation of concurrent systems. International Journal on Software Tools for Technology Transfer," 2007.

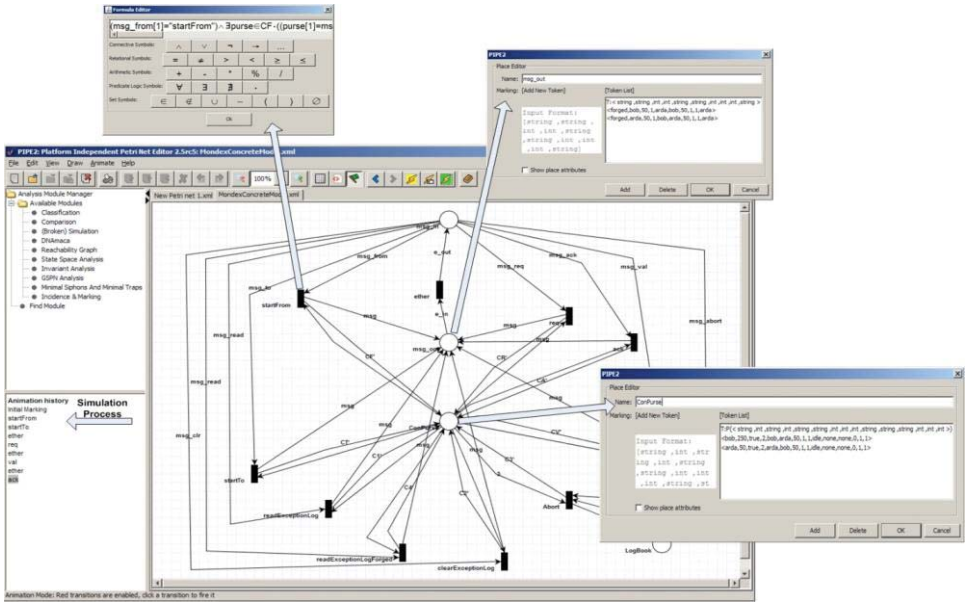


Figure 6 Screenshot of Mondex in PIPE+

A Novel Method for Formally Detecting RFID Event Using Petri Nets

Jinan Sun[1], Yu Huang[2,3], Xin Gao[1], Shikun Zhang[2,3], Lifu Wang[2,3], Chongyi Yuan[1,3]
[1]School of Electronics Engineering & Computer Science, Peking University, 100871, Beijing, China
[2]National Engineering Research Center for Software Engineering, 100871, Peking University, Beijing, China
[3]Key Laboratory of High Confidence Software Technologies of Ministry of Education, Peking University, Beijing, China
{sjn, hy}@pku.edu.cn

***Abstract*—Radio Frequency Identification (RFID) provides fast collection of large volume of data and can be used to identify physical objects with unique IDs. In order to provide semantically meaningful data to different applications, RFID data need to be processed to discover user-defined complex events. We propose a Petri net-based method for the detection of complex events in RFID. A model named ED-net is introduced to specify semantics of complex events, which is also taken as the basis for the implementation of an event detector. Formal model ED-net is an extension of ordinary Petri net, providing user-defined types, functions and expressions, which are suitable for the precise description of attributes and constraints of RFID complex events, with non-temporal, temporal and parameterized constraints. Through modeling all the events to be detected in one ED-net model, we avoid multiple detections of common sub-events of different complex events. Through the experimental evaluation, we verify the efficiency of our detection method. This paper is sponsored by the National Natural Science Foundation of China under Grant No. 60803014 & the National Research Foundation for Doctoral Program of Higher Education of China under Grant No.200800011017. Yu Huang is corresponding author.**

Keywords- RFID; Petri Net; Event Detecting

I. INTRODUCTION

Radio Frequency Identification (RFID) technology uses radio-frequency waves to automatically identify objects, which makes it possible to create a physically linked world where every object is automatically numbered, identified, cataloged, and tracked in real time. An RFID system generally consists of four parts: RFID tags, readers, middleware and application software. An RFID tag is uniquely identified by a worldwide unique ID, stored in its memory and defined by the electronic product code (EPC) standard [1]. Readers are capable of reading the information stored in RFID tags. RFID middleware systems are typically deployed between the readers and the applications in order to correct captured readings and provide clean and meaningful data to applications.

The RFID middleware plays the primary role in RFID data management. However, the characteristics of RFID data pose many challenges in the research of RFID middleware and complex event processing [2]. Firstly, the data generated from an RFID application are simple, and each RFID observation is of some form (epc, reader, timestamp); secondly, RFID data are temporal, dynamic and in large volume. They are generated dynamically and automatically, and must be processed in real time; thirdly, RFID data are inaccurate and have implicit semantics. Erroneous readings, such as missed or duplicate readings, have to be semantically processed. The information carried by RFID data also need to be analyzed to support advanced applications. Such information is often related to business knowledge and specific applications. Traditional event systems do not well support temporal characteristics of RFID events. Therefore, some novel and effective methods are needed for complex event processing in RFID technology.

The concept of complex event originated from active database. Lots of work has been done on complex event detection, both in active database and RFID applications. The work of [6] introduced a general purpose event monitoring system Cayuga, in which queries over event systems are expressed by Cayuga Event Language and implemented by Cayuga automaton. Petri nets are used for the modeling and detection of composite events for active object-oriented database system SAMOS [10]. The Snoop system [7] and EVE system [8] use graph-based approaches, and the work of [9] proposed EPS, in which a subscription tree used to process events.

Researchers also investigated specification languages and detecting methods for RFID complex events. The work of [11] proposed an event-oriented approach to process RIFD data by defining various constructors, including non-temporal and temporal constructors, to express the relationship among complex events. A tree-based algorithm [13] was advanced to improve existing algorithm RCEDA in [11], decreasing the time complexity of RFID complex event detection. SASE[12] executes complex event queries over real-time streams of RFID data, and a complex event language was proposed, supporting negation operating in sequences, parameterized predicates and sliding windows. But some operator-nested complex events were not referred in this language. Zhu [14] aimed at proposing a formal descriptive language $QDDC_{att}$ for complex events, supporting quantitative complex events.

Most of existing researches lack formal semantics to descript complex events, which may bring ambiguity and confusion in expressing and understanding complex events in RFID. Attributes (start time and end time) of a complex event are not defined strictly. Another problem is that existing detection methods are mainly developed for a single complex event, instead of a set of complex events. While in an RFID application, we often have more than one complex event to be detected, and these events share common sub-events. For

complex events $E_1 = (E_a;E_b) \wedge E_c$ and $E_2 = E_a \vee (E_d;E_e)$, they both contain a sub-event E_a. The detection of event E_a will be processed twice when the detections for E_1 and E_2 are carried on separately. This is time-wasting and should be avoided in real-time detection for complex events in RFID.

In this paper, we propose a Petri net-based method named ED-net for the description of complex events in RFID, which is convenient for describing temporal and parameterized constraints with locality property, token-flow mechanism and combinability. Moreover, ED-net has formal semantics which can guide us to design better software.

II. THE MODEL OF RFID EVENTS

An RFID event can be either a primitive event or a complex event. A primitive event is an RFID observation, which occurs at a point of time when an RFID tag is read by an RFID reader. A complex event is an RFID observation sequence, consisting of a set of primitive events and having special semantics. The time of a complex event could be considered as the point of time it is detected, or the entire time interval. The first case is often used in active database, while it will cause logical problems [3]. Therefore, we define the time of a complex event to be a time interval. In RFID event detection, both the temporal distance between two events and the interval of a single event are critical [5].

We introduce several signs and functions which will be used later. An event type is represented by E, and an event instance is represented by e; *t_begin(e)* returns the start time of an event instance *e; t_end(e)* returns the end time of an event instance e; *interval(e)* calculates the interval of an event instance e, and $interval(e) = t\_end(e) - t\_begin(e)$; $dist(e_1,e_2)$ calculates the distance between two event instances e_1 and e_2, and $dist(e_1,e_2) = t\_end(e_2) - t\_ end\ (e_1)$.The above functions are available for all the RFID events. Besides of these functions, users could define specific functions to distinguish primitive events. For example, *group(reader)* is used to represent to which group the reader of a primitive event belongs. Readers in the same group often have the same function; *type(epc)* is used to get the type of an object with tag *epc*.

Temporal constraints need to be taken into account when describing RFID events. The work of [11] discussed series temporal complex event constructors. Constructors for complex events in [11] are classifies into two categories: non-temporal complex event constructors and temporal complex event constructors. Non-temporal complex event constructors include OR($\vee$), AND($\wedge$), and NOT ($\neg$).$E_1 \vee E_2$ occurs when either E_1 or E_2 occurs. $E_1 \wedge E_2$ occurs when both E_1 and E_2 occur. $\neg E$ is usually combined with a temporal constraint, and it occurs when no instance of E occurs during a specific time interval.

Temporal complex event constructors specify temporal constraints of complex events, including the occurrence order of sub-events, distance constraint and interval constraint. These constructors can be nested in arbitrary order to describe various complex events. However, parameters are not included to express constraints on complex events. For example, operator *SEQ+* only requires one or more occurrences of an event, but the exact number of the occurrences is not specified. This might cause inaccurate description of complex events. Suppose we have a sequence of event instances: (e^1,e^3,e^5,e^6), where the superscript represents occurrence time of event e. According to the semantics of *SEQ+* operator, we can tell that complex event *SEQ+(E)* occurs more than once, (e^1,e^3), (e^3,e^5), (e^1,e^3,e^5), (e^1,e^3,e^5,e^6) could all be seen as the corresponding sets of events for complex event *SEQ+(E)*. Therefore, it is inconvenient to describe complex events with exact aggregation number, such as complex event E occurs when event E_1 has occurred ten times. In the following section, we will show how to model parameterized constraints with ED-net.

III. THE MODEL OF ED-NET

A. The static structure of ED-Net

Definition (ED-net) An ED-net (Event Detection Net) is a tuple $N = (\Sigma, P, T, A, C, G, B, E)$, and Σ is a finite set of non-empty types, called color sets; P is a finite set of non-empty places; T is a finite set of non-empty transitions; A is an arc set, $A \subseteq P \times T \cup T \times P$; C is a color function, $C : P \rightarrow \Sigma$; G is a guard function, $G : T \rightarrow \{Expr\}$, where $Expr$ is a boolean expression; B is a body function, $B : T \rightarrow \{Stat\}$, where Stat is a group of assignment operations; E is an arc function, $E : A \rightarrow \{AExp\}$, where $AExp$ is an expression whose value is a multi-set. The form of $AExp$ could be: $AExp := m'c|n'v|m'c + AExp|n'v + AExp$, where m and n are positive integers, c is a constant value of a specific color, and v is a variable. When m or n equals one, we just omit it in the expression. The sign '+' means addition of two multi-sets, as introduced in [15].

Fig. 1 shows an ED-net model and its instance. The model describes a complex event of the conjunction of two primitive events, with the constraint that readers of the two primitive events are in the same group.

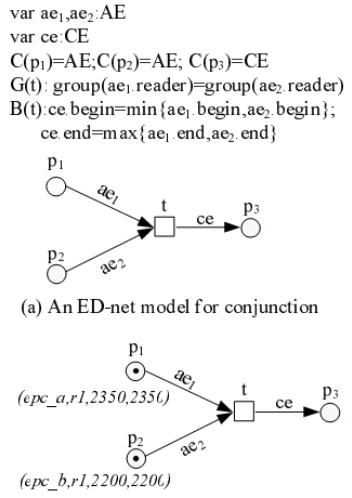


Figure 1. An example of ED-net model.

B. Dynamic behavior of ED-net

The state of an ED-net is represented by a marking that records the number and color of tokens in each place. A

marking is defined as a function $M : P \rightarrow \{C(p)_{MS} | p \in P\}$, where $C(p)_{MS}$ is the multi-set over the color set of place p. Take the instance of ED-net in Fig. 1 for example, its current state can be represented as a marking M: M = (1`(*epc_a*, *r*1, 2350, 2350), 1`(*epc_b*, *r*1, 2200, 2200), ∅). $M(p_1)$= {(*epc_a*, *r*1, 2350, 2350)}, $M(p_2)$= {(*epc_b*, *r*1, 2200, 2200)} and $M(p_3)$=∅ as place p_3 has no token.

The procedure of taking a step in an ED-net is as follows: First, check if a transition could be fired under current marking according to the firing rules introduced later. More than one transition may be enabled in one step, we randomly choose one to occur. Second, do the assignment operations according to the body of the transition, calculating the start and end time of the complex event. For example, for transition *t* in Fig. 1, value (*2200, 2350*) will be assigned to variable *ce*. Third, remove tokens from the input places of transition *t* and add tokens to the output places of transition *t*. The number and color of tokens to be consumed and produced are determined by corresponding arc expressions. According to the arc expression in Fig.1, tokens (*epc_a*, r_1, *2350*, *2350*) and (*epc_b*, r_1, *2200*, *2200*) are removed from p_1 and p_2 respectively and token (*2200*, *2350*) is added to p_3. Thus, the new marking of the ED-net is (∅, ∅, *1`(2200, 2350)*).

Firing conditions for a transition in P/T net [16] are also available in ED-net, which means that for a transition to be fired, there must be enough tokens in all of its input places. Besides of this, guard expression of the transition in an ED-net must be satisfied. We summarize the process of deciding whether a transition *t* could occur under marking *M* as follows: Firstly, check if all the input places of transition *t* are marked under marking *M*. We can see that p_1 and p_2 are both marked in Fig. 1; Secondly, If step 1 is satisfied, find a binding for variables appearing on all the input arcs of transition *t*. In Fig. 1, transition *t* has two input arcs p_1t and p_2t, and two variables ae_1 and ae_2 appear in the arc expressions. Under current marking, they could only be bound with (*epc_a*, r_1, *2350*, *2350*) and (*epc_b*, r_1, *2200*, *2200*) respectively. Multiple binding results could be found when tokens in input places are more than needed; Thirdly, Evaluate guard expression of transition *t* under a binding established in step 2. Only when the result is true, can transition *t* be fired. Otherwise, go to step 2 and choose another binding for corresponding variables. In case the guard expression cannot be satisfied under all the possible bindings, transition *t* cannot be fired.

C. ED-net models for complex events

Non-temporal complex events include conjunction complex event, disjunction complex event and negative complex event. The ED-net model for a conjunction complex event of two primitive events has been shown in Fig. 1. A negative event should be associated with a time-constrained event; otherwise it is meaningless to discuss the occurrence of an event during an endless interval. We will introduce the modeling of negative event in the model for an interval-constrained complex event (*WITHIN*).

A model for the disjunction of two primitive events is shown in Fig. 2. For convenience, we mark the name of the event under its corresponding place in the graphical representation of the model. Once one of places p_1 and p_2 is marked, which means that one sub-event is detected, transition t_1 or t_2 will be fired, and place p_3 will be marked with a token of color *CE* whose value is obtained from the corresponding primitive event. The model is also available for the disjunction of complex events; we only need to change colors of places p_1, p_2 and variables ae_1, ae_2 to be *CE*.

Temporal complex events refer to events with time-constraint, including the occurrence order of sub-events, distance of two events and the interval of an event. The work in [11] has listed a group of temporal complex event constructors covering most of the detecting situations. However, parameterized constraints are not included in the description of complex events. For example, constructors *SEQ+* and *TSEQ+* both express the periodic occurrence of one or more occurrences of an event, but neither specifies the exact number of occurrences. The imprecise description may cause unexpected results. Also, parameters are necessary and important in the specification of a complex event.

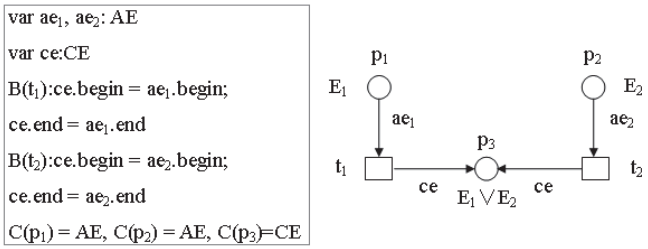


Figure 2. ED-net model for disjunction

Sequential constructor specifies that the occurrence of one event is after another. The semantics of "after" varies according to different understandings. When a user say event E_2 occurs after event E_1, all of these situations could be distinguished by the guard expression defined in the ED-net model. An ED-net Model for a sequence complex event E_1;E_2, and the semantics of "after" corresponds to the third situation, as the guard expression of transition t is $ce_1.end < ce_2.begin$.

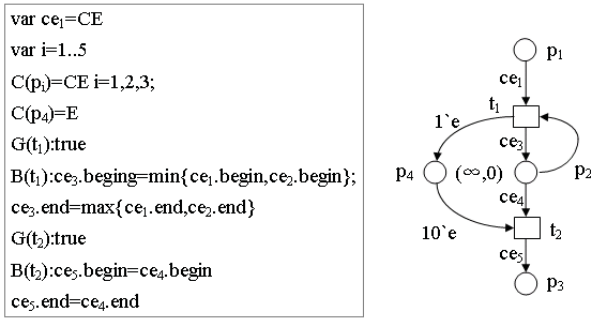


Figure 3. ED-net model for aggregation

The modeling of distance-constrained sequential complex event TSEQ(E_1; E_2,t_1,t_2) is similar to the model of sequential complex event, by changing the guard expression of transition *t* to be $(ce_1.end < ce_2.begin) \wedge (ce_2.end - ce_1.end > t_1) \wedge (ce_2.end - ce_1.end < t_2)$, which indicates that the distance of E_1 and E_2 is bounded by t_1 and t_2.

Fig. 3 shows an ED-net model for an aggregation complex event E, which occurs when its sub-event E_1 has occurred ten times. Places p_1 and p_3 corresponds to events E and E_1 respectively. As soon as E_1 occurs, place p_1 is marked. Then transition t_1 can be fired and a token e is produced in place p_4. The number of tokens in p_4 indicates the occurrence frequency of event E_1. When the number reaches 10, transition t_2 can be fired and ten tokens in p_4 are consumed according to the arc expression 10`e on arc p_4t_2.

We define the interval of an aggregation complex event to be from the earliest start time to the latest end time of all the sub-events. Instead of collecting all the sub-events and finding out the smallest and greatest values, we update the interval of complex event E each time a sub-event arrives, and when all the sub-events are detected, the latest value is the interval for E. In Fig. 3, transition t_1 updates the interval of complex event E according to the time of past occurrences of E_1 (from place p_2) and the time of E_1's new arrival(from place p_1). On the first arrival of E_1, transition t_1 will consume a token of (∞, 0) from place p_2, which is assigned on creation of the model. After the firing of t_1, token in p_2 will be updated to be (*ce1.begin, ce1.end*), which is exactly the time of the first occurrence of E_1:

$ce_3.begin = min\{ce_1.begin, ce_2.begin\} = min\{ce_1.begin, \infty\} = ce_1.begin$

$ce_3.end = max\ \{ce_1.end, ce_2.end\} = max\{ce_1.end, 0\} = ce_1.end$

After ten arrivals of E_1, transition t_2 occurs and place p_3 has a token whose color indicates the time of complex event E.

IV. Detecting Complex Events with ED-net models

In this section, we will introduce how to detect complex events based on ED-net. It is assumed that the RFID data have been filtered before the detection process and the inputs of the event detector is clean. The filtering step could eliminate redundant data [17] [18]. The detection process takes filtered RFID data as input, and sends signals to applications when an event is detected. As introduced in previous section, each event corresponds to a place in an ED-net, which is marked with a token when the event occurs. The ED-net model behaves dynamically by taking steps, and once an event place is marked after one step, associated actions defined in an application will be taken, such as sending an alarm or updating database.

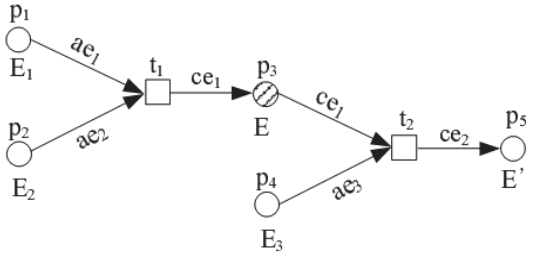


Figure 4. ED-net model for E and E′

The detection process could be divided into two parts: constructing an ED-net model for all the events to be detected and detecting complex events based on the ED-net model which takes RFID data as inputs. To improve the efficiency of the detection method, we combine ED-net models for different complex events and a step-by-step detection mode is applied. Details of the two parts are given below.

A. Constructing ED-net models

In the work of [11] detection is carried on separately. The graph-based computation model merges common sub-graphs, thereby avoiding multiple detections for sub-events. However, common sub-events in different complex events are not combined together. Therefore, they will be detected multi-times in separate detection processes. Instead of modeling all complex events separately, we combine the ED-net models for all the events to be detected in one ED-net model. Note that the final ED-net model may be composed of several independent parts which share no common sub-events.

Algorithm 1: Constructing ED-Net model

```
Input: complex events set S_E
Output: ED-Net model N
  foreach complex event e in S_E do
    if e has not been modeled in N then
       Get e's sub-events set C_E;
       foreach sub-event c_e in C_E do
       if c_e has been modeled in N then
    Add a place p' as a copy of place p;
       else
         Build c_e's ED-net model;
         Extend N with c_e's model;
       end
      end
      Build the ED-net model for event e
(taking its sub-events as inputs);
      Extend N;
    end
  end
return S_N
```

Main steps of constructing an ED-net model for a set of complex events are given in Algorithm 1. A complex event is often composed of several sub-events, which may also contain sub-events. The constructing of the complex event takes the places of its sub-events as inputs. Suppose we have complex events E=($E_1 \wedge E_2$) and E'=($E \wedge E_3$), assuming E_1, E_2 and E_3 are all primitive events. Fig. 4 shows the sketch of ED-net model for E and E'. The caption under place is the name of its corresponding event. The shadowed place p_3 corresponds to event E, and it is also a part of ED-net model for E'.

Different events often share common sub-events. Therefore, an event may be used in more than one detection model. In this case, we duplicate the place for the common sub-event. Event E participates in both complex events E_1 and E_2. Once an event E is detected in place p_1, the information will be duplicated to places p_2 and p_3, which will be used in the modeling of E_1 and E_2 respectively. In this way, event E only needs to be detected once, instead of twice in the separate detection for E_1 and E_2.

B. Detecting complex events with ED-net model

Detection for complex events behaves as Algorithm 2 lists. The detector works as long as RFID data are received. The marking of the ED-net model is a global variable, and whenever a primitive event is detected, the marking is updated. To state clearly, we use notations $\bullet t$ and $t\bullet$ to represent sets of input and output places of transition t respectively.

Algorithm 2: Detect complex events based on ED-net

```
Input: ED-net model N = (Σ,P,T,A,C,G,B,E)
M:Current marking of N;Enabled transition
set Te = ∅;
Repeat
  foreach t in T are marked do
    repeat
      Find b for variables related to t
    until G(t) is true under binding b;
          Add transition t to Te;
  foreach t in Te do
    Fire t;
    Evaluate output arc expressions of t
      foreach place p in •t do
         M'(p)= M(p) - E(pt);
      end
      foreach place p in t• do
         M'(p)= M(p) + E(tp);
         if p is a complex event
            Send M to applications;
         end
    foreach place p in P - (•t ∪ t•) do
       M'(p)= M(p);
    end
    Change current marking M to M';
    Remove transition t from Te;
  end
until no RFID data are received;
```

V. CONCLUSION

We have proposed a method for the detection of complex events in RFID, based on a formal model named ED-net. ED-net offers unified ways for describing both temporal and parameterized constraints of events and defining rules for calculating attributes of complex events. Color sets and functions are used to specify attributes and constraints of events. Constraints of a complex event are expressed by the structure of corresponding ED-net model and guard expressions associated with transitions. Attributes of a complex event can be obtained from those of its sub-events, and the calculation rule is specified in the bodies of transitions. Our detection method describes all complex events to be detected in one model; common sub-events are detected only once, instead of multiple times in independent detecting processes for these complex events, thereby improving efficiency of our methods.

We implement the prototype of ED-net with Java language in RFID test environment and compare it with traditional non ED-net algorithm such as RCEDA[11]. We produce the RFID complex events with 3 sub-events combined about 1000 per seconds, and the average latency of detecting complex events has been efficiently reduced in our experiments. In our experiment, the average CPU occupation time maintained an acceptable level with the average latency down in detecting.

Our future work is to promote our detection method in an RFID event detector and apply it in specific applications.

REFERENCES

[1] Epc tag data standards version 1.1. Technical report, EPC-Global Inc, 2004

[2] D. C. Luckham and B. Frasca, "Complex event processing in distributed systems," Technical Report CSL-TR-98-754, Stanford University ,1998

[3] A. Galton and J. C. Augusto, "Two Approaches to Event Definition," Proceedings of the 13th international Conference on Database and Expert Systems Applications, Lecture Notes In Computer Science, vol.2453, Springer-Verlag, London, 2002, pp. 547-556

[4] R. Derakhshan, M. E. Orlowska, Xue Li, "RFID Data Management: Challenges and Opportunities," IEEE International Conference on RFID, 2007, pp. 175-182

[5] RFID data management, aggregation and filtering. http://epic.hpi.uni-potsdam.de/pub/Home/Publications/RFID-PaperSS2007OleksandrMyly .pdf , 2007

[6] A. Demers, J. Gehrke, B. Panda, et al, "Cayuga: A general purpose event monitoring system," proceedings of the third biennial conference on innovative data systems research (CIDR), 2007, pp 412-422

[7] S. Chakravathy, D. Mishra, "Snoop: an expressive event spec¬ification language for active databases," Journal of Data & Knowledge Engineering, vol. 14(1), 1994, pp. 1-26

[8] A. Geppert and D. Tombros, "Event-based distributed workflow execution with EVE," Technical Report No.96.5, University of Zurich, 1996

[9] D. Moreto and M. Endler, "Evaluating composite events using shared trees," IEEE Proceedings of Software, 148(1), 2001, pp. 1-10

[10] S. Gatziu and K. R. Dittrich, "Detecting compostie events in active database systems using Petri nets," Proceedings Fourth International Workshop on Active Database Systems, Houston, TX, USA, 1994, pp. 2-9

[11] F. Wang, S. Liu, P. Liu, and Y. Bai. "Bridging physical and virtual worlds: Complex event processing for rfid data streams," In EDBT, vol. 3896 of Lecture Notes in Computer Science, Springer, 2006, pp. 588-607

[12] E. Wu, Y. Diao, and S. Rizvi, "High-performance complex event processing over streams," In SIGMOD Conference, ACM, 2006, pp. 407-418

[13] H. Liu, S. GOTO and J. Li, "The study and application of tree-based RFID complex event detection algorithm," Proceedings of the 2009 International symposium on web information systems and applications, 2009, pp. 520-524

[14] J. Zhu, Y. Huang and H. Wang, "A formal descriptive language and an automated detection method for complex events in RFID," Proceedings of the 33rd Annual IEEE International Computer Software and Applications Conference (COMSAC), vol. 1, 2009, pp.543-552

[15] K. Jensen, "Coloured Petri nets: basic concepts, analysis methods and practical use," vol. 1, Springer-Verlag ,1997

[16] T. Murata, "Petri nets: properties, analysis and applications," Proceedings of the IEEE, vol. 77(4), 1989, pp.541-580

[17] Y. Bai, F. Wang and P. Liu, "Efficiently filtering RFID data streams," In CleanDB Workshop, 2006, pp. 50-57

[18] B. Carbunar, M. K. Ramanathant, M. Koyutrk, C. Hoffmannt, A. Grama, "Redundant reader elimination in RFID systems," Proceedings of the Second Annual IEEE Communications Society Conference on Sensor and AdHoc Communications and Networks, (SECON 2005), 2005, pp. 176-184

Multithreaded Pointer Analysis Based on Petri Net

Fei Liu, Bixin Li

School of Computer Science and Engineering, Southeast University, Nanjing, China
Key Lab of Computer Network and Information Integration (Southeast University), Ministry of Education
Email: {fei_liu, bx.li}@seu.edu.cn

Abstract

This paper gives a novel method of investigating flow-sensitive pointer analysis for multithreaded program based on petri net. The method mainly borrows causal dataflow analysis idea from Azadeh Farzan. Petri net is used to describe control flow structure of multithreaded program. And pointer points-to information is propagated along causal dependencies of events in the partial order execution of petri net. The problem of pointer analysis is reduced to the coverability problem on the petri net.

***Keywords*—pointer analysis; multithreaded program; flow-sensitive; petri net**

I. Introduction

With development of multi-core technique and extensive use of program language supporting threads, multithreaded programs are becoming more and more common. However, many mature and traditional program analysis techniques mainly aim at sequential programs. Because of non-determination of multithreaded program semantics, analysis process of multithreaded program is more difficult, compared to that of sequential program. Interactions among multiple threads make it difficult to extend traditional program analysis techniques that are developed for sequential programs to multithreaded programs[5]. Research on analysis of multithreaded program is still immature.

Pointer analysis[4] is a fundamental static program analysis technique. With many years' research, there exist many publications about pointer analysis of sequential programs[1, 3]. Pointer analysis of multithreaded programs makes slow progress, compared to that of sequential programs. But there were still some work on multithreaded pointer analysis[5]. As far as flow-sensitive pointer analysis for multithreaded program is concerned, the main difficulty is the potential interferences among concurrent threads. Two threads interfere when a thread assigns to the shared pointer variable that another thread accesses(i.e., dereference, read or assign). Therefore, the key problem for designing flow-sensitive multithreaded pointer analysis algorithms is how to characterize inter-thread interference. This paper presents a novel method of investigating flow-sensitive pointer analysis for multithreaded program based on petri net. The method borrows causal dataflow analysis idea from[2]. It captures control flow structure using 1-safe petri net and explores partially ordered execution of petri net. The points-to information of pointer is propagated along causal dependency of events. Pointer analysis problem of multithreaded program is converted to the marking coverability problem of petri net.

The rest of paper is organized as follows. Section II introduces some preliminaries containing 1-safe Petri net and Mazurkiewicz trace. Section III presents multithreaded pointer analysis based on petri net. Section IV shows a case study and Section V gives some related work analysis. Section VI concludes.

II. Preliminaries

A petri net is a triple N=(P,T,F), where every element in set P is called a place, tokens in every place represents resources that each place owns, each element in set T is called a transition, set P and T satisfying $P \cap T = \emptyset$, and the binary relation F is called flow relation of net N, satisfying $F \subseteq (P \times T) \cup (T \times P)$. Net N is called 1-safe petri net if each place contains at most one token at any time. A trace alphabet is a pair (Σ,I) where Σ is a finite set

Supported partially by the National Natural Science Foundation of China under Grant No. 60773105 and no. 60973149, and partially Supported by Doctoral Fund of Ministry of Education of China under Grant No. 20100092110022, and partially by National High Technology Research and Development Program under Grant No.2008AA01Z113.

Correspondence to: Bixin Li, School of Computer Science and Engineering, Southeast University, Nanjing, China. E-mail: bx.li@seu.edu.cn

of events, every element representing action and relation $I \subseteq \Sigma \times \Sigma$ is called the independence relation, it being irreflexive and symmetric. $D = (\Sigma \times \Sigma) - I$ is called the dependence relation. A Mazurkiewicz trace is a behavior that describes a partially-ordered execution of events in Σ.

III. Multithreaded Pointer Analysis Based on Petri Net

This section mainly contains three parts. In the first place, the definition of multithreaded language PML(Pointer-included simple Multithreaded Language) is introduced. In the second place, the construction of petri net model of PML program is illustrated. In the last place, multithreaded pointer analysis based on petri net is proposed. For an arbitrary given PML multithreaded program P, the aim of pointer analysis is to determine the points-to set of pointer variable at some program point. The outline of the method of investigating flow-sensitive pointer analysis for multithreaded program based on petri net is as follows. Firstly, program P is transformed to P_transformed, which is semantically equivalent to program P. Secondly, petri net representation N of program P_transformed is constructed according to petri net representation of basic statement. Thirdly, CCD framework instance for multithreaded pointer analysis based on petri net is given. Fourthly, MOT(Meet Over All Traces solution) solution to reaching definition analysis of pointer variable is reduced to a coverability problem on the petri net which is solved by PEP(Programming Environment based on Petri Nets) tool based on partially ordered unfolding techniques.

A. Multithreaded language definition

The analyzed multithreaded program in this paper is based on simple multithreaded language PML(Pointer-included simple Multithreaded Language). The syntax of PML is shown in figure 1. Pointer_first_level in figure 1 is a set of pointer variable whoes type is declared as int *. And Pointer_two_level is a set of pointer variable whoes type is declared as int **.

B. Petri net model of PML program

For a given PML program P, this paper utilizes petri net to represent program control flow structure. In order to construct petri net model for PML program, there are two steps: (a)source code transformation of multithreaded program P keeping semantically equivalent, let transformed program be P_transformed; (b)model control flow structure of P_transformed using petri net.

P::=defn; threadlist
threadlist::=null | stmt || threadlist
defn::=int X | lock | int *p | int ** s |defn; defn
stmt::=stmt;stmt | X=e | *p=e| e=*p | p=&X| p=q | s=&p | p=*t | *s=q
|while(b){stmt} | if(b) then {stmt} else {stmt}
|acquire(l) | release(l) | skip
e::= i| Y| e+e| e-e| e*e| e/e
b::=true| false| e op e| b∨b | ¬b
$i \in Integer; X, Y \in Var; p, q \in Pointer\_first\_level;$
$s, t \in Pointer\_two\_level; l \in Lock; op \in \{<, \leq, >, \geq, =, ! =\}$

Fig. 1. Syntax of PML

1) Semantically equivalent source code transformation: For multithreaded program P, firstly, find all assignments statements in relation to dereference to two-level pointer variable, such as p=*t or *t=q statements. Secondly, do with the foregoing statements as follows: (using p=*t as an example to illustrate)

- scan source code of program P, for two-level pointer variable t, find all assignment statements where t is assigned. In the simple multithreaded language PML, pointer variable is one-level or two-level variable and PML currently doesn't consider type cast. So the value of two-level pointer variable is only changed by assignment statements such as $s = \&t$, or s=t. When the found assignments in the thread containing statement p=*t, the last assignment statement to t is selected. Otherwise, all assignment statements in other threads or some relevant assignment statements according to synchronization structure of multithreaded program are selected.
- convert statement p=*t to statement section(*) according to all selected assignment statements (t=&w, t=&v,$\cdots$,t=&q; let the number be N), as illustrated in figure 2. For statement *t=q, there exists corresponding statement section(**) similar to(*).
- replace all assignment statements in relation to dereference to two-level pointer variable t with statement section(*) or (**) firstly, then delete all assignment statements referring to variable t. The remainder is program P_transformed, which is semantically equivalent to program P. The pointer variables in program P_transformed are all single-level pointers. The assignment statements of pointer variable only contain following type: p=&X and p=q.

2) Petri net representation of program control structure: This paper models control flow structure of P_transformed using petri net. The transition corresponds to program statement, and the place is used to represent intra-thread control flow relation, inter-thread dependency relation and synchronization relation. Petri net representations of basic statements are illustrated in Figure 3.

```
{    int index =random(0,N-1);      //generate a number from 0 to N-1 randomly
     if (index= =0) then { t=&w;   p=w; }
        else if(index = = 1) then { t=&v;   p=v;}                    (*)
          else ……
        else if (index = = N-1) then { t=&q ; p=q; }
}
```

Fig. 2. Converting statement to statement section

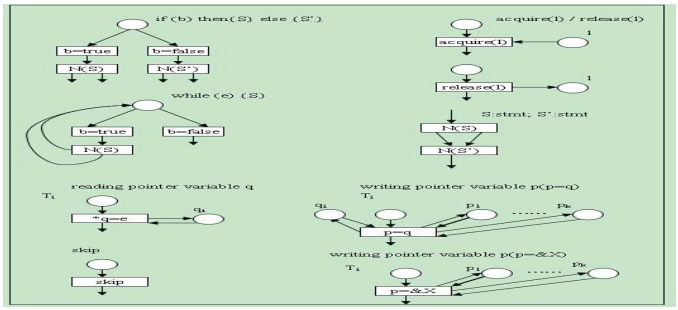


Fig. 3. Petri net representations of basic statements

C. Multithreaded pointer analysis based on petri net

The method of investigating flow-sensitive pointer analysis for multithreaded program based on petri net in this paper mainly borrows causal dataflow analysis idea from [2]. PML Program P is transformed to P_transformed, which is semantically equivalent to program P. Thus pointer analysis of program P is equivalent to that of program P_transformed. P_transformed doesn't contain procedure call. The statements that could change pointer variable points-to information can only be basic statements such as p=q and p=&X. Under this circumstance, pointer analysis for program P_transformed, that is, determining points-to set of pointer variable p at program point n, is equivalent to determining which assignment statements referring to variable p in program P_transformed could reach program point n, namely reaching definition analysis of variable p. Then reaching definition analysis of pointer variable p is solved using a concrete framework instance based on CCD(Concurrent Causal Dataflow analysis) framework proposed in [2].

1) CCD framework instance for multithreaded pointer analysis based on petri net: The CCD framework instance used for multithreaded pointer analysis is a quintuple$(N, \mathcal{S}, \mathcal{F}, \mathcal{D}, \mathcal{D}^*)$, including petri net model of P_transformed N=(P,T,F), property space $\mathcal{S} = (\mathcal{P}(\mathcal{D}), \subseteq, \cup, \emptyset)$, finite pointer points-to set $\mathcal{D}$, powerset$\mathcal{P}(\mathcal{D})$, $\emptyset$ representing initial pointer points-to set, $\cup$ determining how we will combine points-to information along program control structure reaching the same control point in a program. For program P_transformed, let pointers=$\{$p | there exists assignment statement to pointer variable p in P_transformed$\}$. Scanning program P_transformed and finding all pointer assignment statements similar to $p = \&X$, for example $p = \&X, q = \&Y, p = \&A, s = \&C$, and denoting p-to=$\{X, A, \cdots\}$, q-to=$\{Y, \cdots\}$, s-to=$\{C, \cdots\}, \cdots$. Finally merging foregoing (*)-to set as set pointers-to. Thus, $\mathcal{D}$=pointers $\times$ pointers-to = $\{$(p,X) | $p \in$ pointers $\wedge$ $X \in$ pointers-to$\}$. For any transition t, set $\mathcal{D}_t$ and $\mathcal{D}_t^*$ are defined. $\mathcal{D}_t^*$ represents points-to information relative to transition t, and $\mathcal{D}_t$ expresses points-to information that may modify when it executes. $\mathcal{D}_t$ and $\mathcal{D}_t^*$ satisfy condition $\mathcal{D}_t \subseteq \mathcal{D}_t^* \subseteq \mathcal{D}$. And $\mathcal{D} = \{\mathcal{D}_t \mid t \in T\}$ and $\mathcal{D}^* = \{\mathcal{D}_t^* \mid t \in T\}$. For each transition t of petri net, function f_t is defined in order to represent how points-to information of pointer variable changes when t is executed.

2) MOT solution to multithreaded pointer analysis based on petri net: It can be verified that CCD framework instance for multithreaded pointer analysis based on petri net proposed in this paper is a distributive instance. The multithreaded pointer analysis problem can be solved by utilizing algorithm proposed in [2].

IV. Case Study

This section gives a simple PML program to illustrate the method of multithreaded program pointer analysis proposed in this paper.

- There are three threads in the example program, as given in figure 4. Some statements in three threads are synchronized by lock variable. The dealt problem is to determining the points-to information of pointer variable p in statement t14 and statement t32.

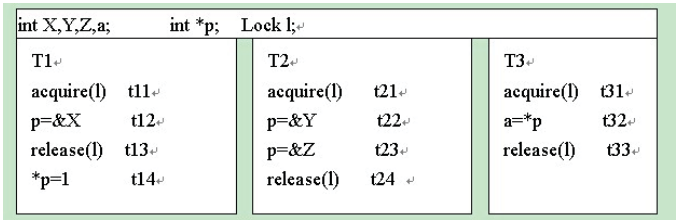


Fig. 4. A simple PML multithreaded program

- Petri net structure of program(Figure 5)
- Solution to MOT
  Points-to information of pointer variable at program point n is points-to information that holds before the execution of corresponding transition t of program's

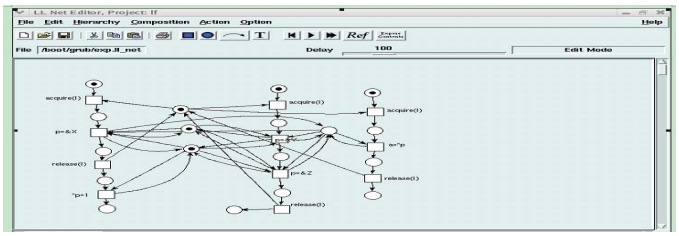


Fig. 5. Petri net structure of program

petri net. And the problem of computing MOT(t) solution is reduced to the problem of marking reachability on petri net.

From the experiment, we can acquire the conclusion that points-to information of pointer p holding before the execution of t14(*p=1) is d_1, d_2, d_3(let $d_1 = (p, x), d_2 = (p, y), d_3 = (p, z)$.), and that of pointer p holding before the execution of t32(a=*p) is d_1, d_3. Some related result are shown as figures 6,7.

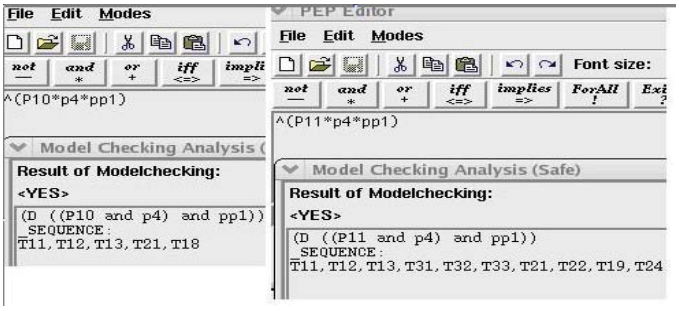


Fig. 6. Some related result

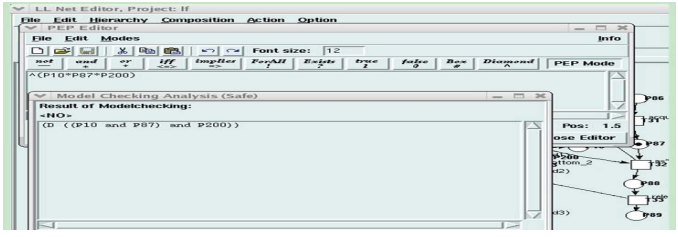


Fig. 7. Some related result

V. Related Work Analysis

Pointer analysis of multithreaded programs makes slow progress, compared to that of sequential programs. There were relatively little work on multithreaded pointer analysis. The first flow-sensitive pointer analysis algorithm for multithreaded program is proposed in[5]. It uses a parallel flow graph to represent multithreaded program. And no edges between concurrent threads are supposed. Similar to traditional sequential program flow-sensitive pointer analysis, dataflow equation systems for four basic pointer assignments and concurrent structure are established and are solved by fixed-point iteration algorithm. It handles many constructors and has many advantages, but it only analyzes programs with structured concurrent constructs such as **fork-join**, and it ignores synchronization constructs such as **lock**. Flow-sensitive pointer analysis for multithreaded program introduced in this paper is based on petri net.Petri net commendably represents synchronization constructs and unstructured multithreaded program. And petri net based pointer analysis provides a novel perspective to consider pointer analysis for multithreaded program. Multithreaded pointer analysis can be reduced to coverability problem on the petri net which is solved by PEP tool based on partially ordered unfolding techniques.

VI. Conclusion

This paper proposes a novel method of investigating flow-sensitive pointer analysis for multithreaded program based on petri net. The method has been applied to several simple designed programs and the experiment results show its effectiveness. It is motivated by causal dataflow analysis idea from[2]. It models multithreaded pointer program using 1-safe petri net, introduces CCD instance for multithreaded pointer analysis based on petri net, and solves the instance using petri net reachability analysis.

References

[1] L.O. Andersen. *Program analysis and specialization for the C programming language*. PhD thesis, 1994.

[2] A. Farzan and P. Madhusudan. Causal dataflow analysis for concurrent programs. *Tools and Algorithms for the Construction and Analysis of Systems*, pages 102–116, 2007.

[3] B. Hardekopf and C. Lin. Semi-sparse flow-sensitive pointer analysis. In *Proceedings of the 36th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 226–238. ACM, 2009.

[4] M. Hind. Pointer analysis: haven't we solved this problem yet? In *Proceedings of the 2001 ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering*, page 61. ACM, 2001.

[5] R. Rugina and M.C. Rinard. Pointer analysis for multithreaded programs. In *Proceedings of the ACM SIGPLAN 1999 conference on Programming language design and implementation*, page 90. ACM, 1999.

Facilitate IT-Providing SMEs in Software Development: a Semantic Helper for Filtering and Searching Knowledge

Riccardo Martoglia
DII – Department of Information Engineering
University of Modena and Reggio Emilia
Modena, Italy
riccardo.martoglia@unimo.it

Abstract— **Software development is still considered a bottleneck in the advance of the Information Society. The recently started FACIT-SME European FP-7 project targets to facilitate the use and sharing of Software Engineering methods and best practices among software developing SMEs. On top of an Open Reference Model (ORM) serving as an underlying knowledge backbone, specific filtering/search mechanisms will support the identification of adequate processes and practices for specific enterprise needs. In this paper, we focus on the proposal of knowledge-based text analysis and retrieval techniques which will form a key component of the advanced filtering mechanisms of the project. The proposed solution is designed to be more powerful and flexible than standard syntactic search techniques, but also to be easily applicable for any SME. The experimental evaluation on the preliminary implementation shows promising results.**

Keywords-software engineering; information retrieval; text analysis; semantic knowledge; semantic similarity.

I. INTRODUCTION AND MOTIVATION

Over the last years, Software Engineering (SE) research has provided more and more advanced and promising techniques for facilitating software development. In particular, the integration between Software and Knowledge Engineering has recently become very important, and several techniques possibly enabling better domain knowledge sharing and assisting developers in specific tasks such as component reuse [1] or software process assessment [2, 3], have been proposed to the research community [4].

Nonetheless, back to the "real" software development world, recent studies have shown that a large number of fundamental challenges still need to be faced. In Europe, software development is becoming a bottleneck in the development of the Information Society [5], while, on a global scale, the quality and productivity of work has not been able to keep up with the society software needs [6]. These issues are especially critical in the case of SMEs in the software development market: indeed, even if innovative SE methodologies are constantly devised and presented, many enterprises are usually not able to take full advantage from them since they generally lack the resources and knowledge needed for the internal deployment of the required methods and tools. Indeed, being software an often underpaid product, SMEs need to allocate mostly all of their available resources on its production rather than, for instance, on new technology training.

This is the challenging scenario of the recently started European FP7 3 years project "Facilitate IT-providing SMEs by Operation-related Models and Methods (FACIT-SME)". The main project goal is to facilitate IT SMEs in using SE methods for design and development, systematizing their application integrated with the business processes. Another fundamental goal is to provide efficient and affordable certification of these processes according to internationally accepted standards, and to securely share best practices, tools and experiences with development partners and customers. In order to achieve these goals, the project will develop a novel Open Reference Model (ORM) [7] for ICT SMEs serving as an underlying knowledge backbone and, on top of that, a customizable Open Source Enactment System (OSES) [8] will provide IT support for the project-specific application of the ORM. More specifically, the ORM will store existing reference knowledge for software-developing SMEs, including different engineering methods, tools, quality model requirements and enterprise model fragments of IT SMEs, in a computer-processable form. On top of the ORM repository, specific search mechanisms, which will be a key part of the OSES, will support the identification of adequate processes and data structures for a specific enterprise. Different application scenarios, identified with the support of the participating SMEs and enumerating the possible use cases of the FACIT-SME solution, will be dealt with. The most notable ones include supporting the organizations in their need to find a new methodology ("From Scratch" scenario) or to modify an existing one in order to better manage its software development projects ("From Methodology" scenario). In all cases, through a filtering phase, which takes as input company and project information and, for the second scenario, existing methodology descriptions, the organization will receive a set of suggestions in the form of the most relevant / useful elements and models in the ORM. Subsequent phases will also include helping the organization to easily check quality constraints and refining the models and results in order to adapt it to its needs. Besides five R&D partners providing the required competences, the project consortium also includes five SMEs operating in the ICT domain which will evaluate the results in daily-life application.

In this paper, we focus on the foundations we are laying for the filtering/searching mechanisms, carefully considering the actual user-targets these techniques will be aimed at. More specifically, we will primarily take advantage of textual

information, a vital knowledge source not only in the ORM defined in the project but also in the documentation already available in each enterprise. In this respect, we propose an innovative approach based on text analysis and semantic retrieval techniques leading to the following achievements:

- it is powerful enough to provide enhanced searching effectiveness over standard syntactic techniques;
- it is general and flexible as a basis of many functionalities offered by the OSES (i.e. for filtering software methodologies for software process assessment and improvement, quality requirements for helping in certification process, best practices for faciliting knowledge sharing, and so on);
- it is devised for IT SMEs, providing them with easy-to-apply methods that do not require big investments or knowledge prerequisites, allowing them to query for the information they need in the way they are used to;
- it exploits the large amounts of textual knowledge (i.e. methodology descriptions, and so on) already available in each enterprise, without requiring complex conversions toward complex structured formats which would be time and cost consuming.

Such approach forms the foundations of a **Semantic Helper** component which will be overviewed in Section II, while the analysis and semantic search techniques themselves will be deepened in Sections III and IV, respectively. Section V shows the promising results of a preliminary experimental evaluation, while Section VI concludes the also by briefly analyzing related works.

II. SEMANTIC HELPER OVERVIEW

The Semantic Helper will support other components of the FACIT-SME solution in filtering/searching/analyzing relevant information available in the ORM, including:

a) **assisted filtering / selection** of ORM elements given specific enterprise objectives (e.g. in "From Scratch" scenario to give pointers to useful information for certification status);
b) **assisted suggestion proposal** for a given enterprise methodology (e.g. in Scenario "From methodology" to help identifying relevant information or gaps between the given methodology and the ORM methodologies);
c) **automatic matching** between ORM documents (such as quality requirements and SE methodologies).

In order to facilitate such processes, a representation of the key parts of the ORM in a semantic and machine-processable way is needed. Given the predominant importance of textual information in the ORM, first of all we provide the Semantic Helper with appropriate **text analysis** techniques (see Section III), which are designed to automatically extract a shared "terminology" from the given set of **documents**. The extracted terminology is enriched with statistical and semantic information (i.e. links to thesauri and domain vocabularies, definitions, synonyms), in order to obtain a computer-processable semantic glossary Note that the analysis can be applied not only to documents coming from the ORM (e.g. about different quality requirements or software methodologies), but also to any "external" document or query submitted by an enterprise. In any case, once documents are reduced to a set of terms with associated information, appropriate **semantic similarity** techniques (detailed in Section IV) are exploited to easily identify relevant documents w.r.t. to a given query document, and to produce a list of suggestions ranked on the similarity (relevance) score.

III. TEXT ANALYSIS TECHNIQUES

A. Text Analysis and Keyword Extraction

The goal for text analysis and keyword extraction in the FACIT project is to design and develop an effective and easy-to-apply technique for automatically extracting terms (and their associated semantic information and statistics) from the submitted text documents. In particular, we wanted to devise a flexible technique to be exploited both for "off-line" analysis, thus working on the textual descriptions already available in the ORM, and for "on-line" querying operations, i.e. applied on the fly to the submitted query documents. Even if many packages are available for keyword extraction purposes, most of them do not allow sufficient configuration and extension options, making their integration with the future FACIT solution very complex. Therefore, we preferred to design a custom-made technique tailored to the FACIT environment. Here is a short summary of the steps performed, for each text document, in the text analysis phase:

1. **Tokenization**: the text is "tokenized" (words are identified, punctuation is removed);
2. **Stemming**: the tokens are "normalized" and "stemmed", i.e. terms are reduced to their base form (managing plurals, inflections, ...) (as we will see this will be very useful to enhance the effectiveness of the similarity computation phase);
3. **POS (Part of Speech) Tagging**: the tokens are "tagged" with Part of Speech tags (i.e. nouns, verbs, ...);
4. **Composite terms identification**: possible composite terms (such as "product action plan" or "product requirement") are identified by means of a simple state machine and of POS tags information;
5. **Filtering and enrichment:** by exploiting external knowledge sources, the most relevant terms are selected and they are associated to additional information (such as definitions, synonyms, …). More specifically we make use of the IEEE Software and Systems Engineering Vocabulary [1], a knowledge source covering specialist terms in the project area, and the WordNet[2] English thesaurus, possibly complementing the specialist source with general knowledge about English concepts;
6. **Term statistics and weights computation**: weights are computed for each of the terms, reflecting their

[1] http://www.computer.org/sevocab
[2] http://wordnet.princeton.edu/

TERM	WN	IEEE	SYNS	DEFS	IDF	DOC_LIST
acquirer	Y	Y	buyer, customer, owner, purchaser	(1) stakeholder that acquires or procures a product or service from a …	7,4961	['QM1372']
acquisition	Y	Y	outsourcing	(1) process of obtaining a system, software product or software service …	5,5491	['QM0392', 'QM0755', …]

Figure 1. An excerpt of the FACIT-SME semantic glossary (global view)

importance and meaningfulness in the text. As we will see, this information is fundamental in computing accurate text similarities (more on this in the following sections).

By applying batch text analysis to the documents currently available in the ORM, we achieved a first significant result in the FACIT-SME project, i.e. the automatic generation of a **semantic glossary**, representing a first step toward the sharing of the most important concepts available in the model and the automatic computation of text similarities. Thanks to the automatic text analysis procedure, this first draft can be easily updated/enriched in case of new content added to the ORM, while more fine-grained user interventions for adding/modifying/eliminating information are also possible. The following section describes the semantic glossary structure more in detail.

B. FACIT-SME Semantic Glossary

The Semantic Glossary consists of a **global view** (all terms in all documents, together with their statistics) and a **per-document view** (terms occurrences in each of the documents with their statistics). The **glossary global view** is an alphabetical sort of all the extracted terms, in a tabular form. Figure 1 shows an excerpt of the glossary global view. The format is:

TERM - the extracted term;
WN – whether the term is present in WordNet thesaurus;
IEEE – whether the term is present in the IEEE vocabulary;
SYNS – possible synonyms for the term (as extracted from the IEEE vocabulary and/or WordNet);
DEFS – possible definitions for the term (as extracted from the IEEE vocabulary and/or WordNet);
IDF – the inverse document frequency of the term in the collection;
DOC_LIST – a list of the documents IDs in which the term occurs.

The **glossary per-document view** is a list of all the term occurrences in the documents, sorted on the document ID, together with their statistics. Figure 2 shows an exercpt of the glossary per-document view. For each term (in each document) the report contains:

DOC – the document ID in which the term occurs;
TERM - the extracted term;
TF - the frequency of this term in the document, normalized by total number of terms in document;
WEIGHT - the TF*IDF weight of the term.

DOC	TERM	TF	WEIGHT(TF*IDF)
QM0001	iso	1	6,8024
QM0002	management	0,3333	0,6931
QM0002	quality	0,3333	1,0303

Figure 2. An excerpt of the FACIT-SME semantic glossary (per-doc view)

The glossary includes both synonyms/definitions and weight information, allowing, as we will see, the similarity functions of the Semantic Helper to draw useful knowledge from both the semantic and the classic text retrieval worlds. As to the semantic information, note that even if all the similarity techniques described in the next section are designed to work (and, as proved in Section V, provide encouraging results) without further intervention, the list of synonyms and definitions retrieved from the external knowledge sources could easily be refined manually by experts or automatically by means of techniques such as sense disambiguation [9, 10]. In addition, as in classic Information Retrieval, the importance (**weight**) of each keyword in each document is estimated. Beside term frequency (**TF**), we compute the inverse document frequency (**IDF**)[3] [11], which provides an estimate of the meaningfulness of each term. The weight is then computed as TF*IDF. In this way, very common terms which are present in a large number of documents have a lower weight and will give a lower contribution to the final similarity, since they are probably less meaningful.

IV. SEMANTIC SIMILARITY TECHNIQUES

As anticipated in the past sections, the need of effectively and efficiently computing similarities between documents is crucial to the project. To this end, we want to define a document similarity formula $DSim(D^x,D^y)$ which, given a source document D^x and a target document D^y, quantifies the similarity of the source document with respect to the target document. Being documents represented by sets of terms, semantic similarity computation becomes a matter of computing similarities between sets of terms. Therefore, document similarity will, in turn, use a combination of the scores provided by a term similarity formula *TSim* between the document terms. The computation of *DSim* between a given D^x and all the possible submitted D^y induces a **ranking** of the available documents with respect to the source one, thus predicting which documents are relevant and which are not with respect to D^x. For instance, by computing the document

[3] IDF is obtained by dividing the total number of documents by the number of documents containing the term and then by computing the logarithm of that ratio

similarities between a given quality requirement description and all the available software methodology descriptions of the ORM, the induced ranking will suggest all the software methodologies that could be relevant/related to the given quality requirement.

We provide different options with respect to the similarity formulas, so to be able to experimentally assess the ones most suited to the project. Equation (1) shows the standard document similarity formula between a given source document D^x and a target document D^y: the similarity is given by the sum of all term similarities between each term in D^x and the term (defined in (2)) in D^y maximizing the term similarity with the term in D^x:

$$DSim(D^x, D^y) = \sum_{t_i^x \in D^x} TSim\left(t_i^x, t_{j(i)}^y\right) \quad (1)$$

$$t_{j(i)}^y = \ argmax_{t_j^y \in D^y} \left(TSim\left(t_i^x, t_j^y\right)\right) \quad (2)$$

Note that (1) is not meant to be symmetric, instead it is conceived so to facilitate the ranking of the documents D^y with respect to document D^x. In case symmetry is needed, the summation in (1) can by extended to the terms of both documents. As to term similarity, the most basic option, only considering equal terms, is shown in (3), where the similarity *TSim* between two terms t_i and t_j is basically a Boolean formula valued 1 if the two terms are equal, 0 otherwise.

$$TSim(t_i, t_j) = \begin{cases} 1 & t_i = t_j \\ 0 & otherwise \end{cases} \quad (3)$$

Equation (4) proposes an extended (and possibly more effective) document similarity option, taking into account the weights as extracted by the text analysis process:

$$DSim(D^x, D^y) = \sum_{t_i^x \in D^x} TSim\left(t_i^x, t_{j(i)}^y\right) \cdot w_i^x \cdot w_{j(i)}^y \quad (4)$$

where $w_i^x = tf_i^x \cdot idf_i$; $w_{j(i)}^y = tf_{j(i)}^y \cdot idf_{j(i)}$. In this case, each term contributes to the final similarity with a different weight *w*, i.e. more frequent and more significant terms contribute more to the similarity between the two documents.

Let us now consider more advanced options for term similarity. Equation (5), besides equal terms, also takes **synonyms** and **semantically related terms** into account:

$$TSim(t_i, t_j) = \begin{cases} 1 & t_i = t_j \ \ or \ \ t_i \ SYN \ t_j \\ r & t_i \ REL \ \ t_j \\ 0 & otherwise \end{cases} \quad (5)$$

More specifically, the case of maximum similarity (value 1) is extended to the case where the two terms are synonyms (*SYN* relation). Moreover, the formula provides a further case where the two terms are not equal or synonyms, nonetheless they are in some way strongly related from a semantic point of view: such terms will contribute with a similarity of *r*, where $0<r<1$.

While the synonym information straightly comes from the text analysis phase, we consider two different ways of exploiting the extracted information and the external knowledge sources to determine wheter two given terms are semantically related. Equation (7) shows a possible way of computing the similarity by exploiting the glosses (definitions) of the terms:

$$t_i \text{ REL } t_j \iff \text{GSim}\left(\text{gl}(t_i), \text{gl}(t_j)\right) \geq \text{Th} \quad (7)$$

$$\text{GSim}\left(\text{gl}(t_i), \text{gl}(t_j)\right) = \sum \left|\text{ovl}\left(\text{gl}(t_i), \text{gl}(t_j)\right)\right|^2 \quad (8)$$

In this case, two terms are semantically related (*REL* relation) if the gloss similarity *GSim* between their glosses exceeds a given threshold *Th*. The Literature presents many possible ways of computing similarities between glosses. We found the extended gloss overlap measure [12] shown in (8) to be particularly effective in our context, especially with the IEEE vocabulary glosses. It quantifies the similarity between the two glosses by finding overlaps in them (the similarity is the sum of the squares of the overlap lengths). Other gloss similarity measures could also be exploited and investigated in the future, such as the gloss vector one described in [13].

Another possible way of computing semantic relatedness is to exploit the relations between terms coming from the WordNet thesaurus. Indeed, we adopt one of the most widely used methods in knowledge management, relying on the hypernymy relations of the thesaurus:

$$t_i \text{ REL } t_j \iff \text{HSim}\left(t_i, t_j\right) \geq \text{Th} \quad (9)$$

$$\text{HSim}\left(t_i, t_j\right) = \begin{cases} -\ln \frac{\text{len}(t_i, t_j)}{2H}, & \exists \ \text{lca}(t_i, t_j) \\ 0, & \text{otherwise} \end{cases} \quad (10)$$

In this case, two terms are semantically related if their hypernym similarity *HSim* exceeds a given threshold *Th*. In particular, the *HSim* shown in (10) derives from the works [9, 14] and computes a score which is inverserly proportional to the length of the shortest path connecting the (senses of the) two terms. H is a constant, which for WordNet is defined as 16. On the other hand, the similarity is 0 if the two terms are not connected in the WordNet hypernymy structures.

Note that the different document similarity and term similarity formulas presented in this section can be selected in a fully orthogonal way, so to be able to adapt in a flexible way to the specific settings and needs of the project.

V. Experimental Evaluation

In this section we present the results of the preliminary effectiveness evaluation we performed on the proposed techniques in the context of the project. We formed a collection of 1500 documents (i.e. textual descriptions) about quality requirements which will be part of the final ORM and derive from existing quality models such as CMMI [15] and ISO 9000 [16]. Then, we created different queries with reference to this collection; each query is either composed by a short text containing candidate keywords, so to simulate possible querying situations following the "From Scratch" scenario of the project (queries Q1-Q6, selected as the most representative ones), or by a whole new document ideally representing the description of an existing enterprise requirement or methodology, similarly to the "From

Query	Results			No sem syn/rel (baselines)			No kw sel (baselines)		
	Prec	Rec	F	Prec	Rec	F	Prec	Rec	F
Q1	1,000	1,000	1,000	1,000	1,000	1,000	0,011	0,420	0,022
Q2	1,000	1,000	1,000	1,000	1,000	1,000	0,005	0,330	0,010
Q3	1,000	1,000	1,000	1,000	0,969	0,984	0,946	0,240	0,383
Q4	0,947	1,000	0,973	1,000	0,079	0,146	0,921	0,321	0,476
Q5	0,878	1,000	0,935	1,000	0,077	0,143	0,986	0,235	0,380
Q6	0,923	0,949	0,936	1,000	0,333	0,500	0,967	0,369	0,534

Figure 3. Effectiveness analysis in terms of precision, recall and F-measure (standard results on the left, two baselines on the right)

Methodology" scenario (queries QT1-QT4). Each query will be submitted to the current implementation of the semantic helper (text analysis and similarity computation), so to generate a set of possible "suggestions", i.e. pointers to the relevant documents in the collection. In order to evaluate the effectiveness of our approach, for each query the output of the helper will be compared to a "gold standard", i.e. the relevant answers which were manually selected from the collection by experts in the field.

The first analysis we conducted was to assess the quality of the retrieved results in terms of precision and recall, which are typical evaluation metrics in the information retrieval field[4]. Figure 3 shows the results for Q1-Q6 (left part of figure). The shown results are those obtained with the gloss similarity as the similarity relatedness function (later we will discuss the WordNet hypernym-based one). Besides precision and recall, we also report, as a summarizing figure, their weighted harmonic mean (F-measure). Further, in order to emphasize the contribution of the different applied techniques to the achieved results, in the right part of figure we also present the results concerning two baselines, i.e. a standard retrieval method ignoring semantic synonyms and related terms information and another method not exploiting the text analysis phase (including stemming and composite terms identification). Let us now analyze the results in detail.

As we can see from Figure 3, the precision and recall levels achieved by the described techniques are very satisfying for all queries (equal or higher to 0.84 and 0.94, respectively). The processing of all queries greatly benefits from the text analysis phase: as the results of the second baseline show, without it recall levels significantly drop to 0.2-0.4 (for instance, different inflections of the same word are not correctly identified). Text analysis can also greatly benefit precision as, for instance, in Q1 and Q2: since they contain, among others, such composite expressions as "interface requirement" (Q1) or "configuration management system" (Q2), not correctly identifying them leads to a very large number of irrelevant retrieved documents (in the second baseline precision drops to less than 0.01, compared to 1 for the standard results). Differently from Q1 and Q2, queries Q3-Q6 also require synonyms and related terms management in order to provide satisfying answers: for instance, one of the key terms in Q3 is "supplier", a concept which is expressed as "vendor" in some of the documents (recall goes from 1 to 0.96 of the first baseline), while Q4 contains "purpose" which is

[4] Precision is defined as the fraction of retrieved documents which are known to be relevant, recall is the fraction of known relevant objects which were actually retrieved.

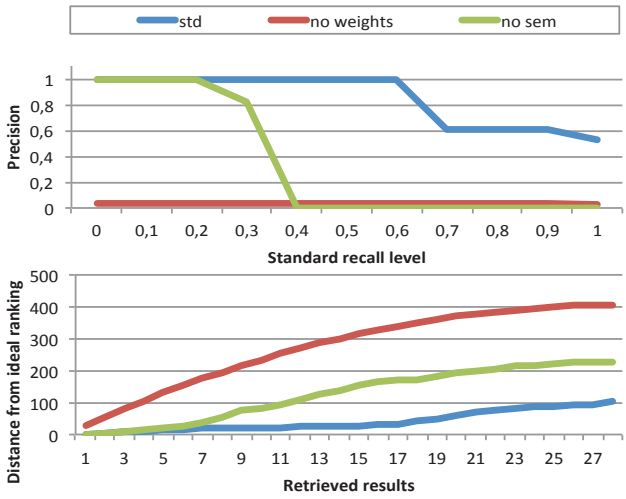


Figure 4. In-depth effectiveness analysis for query QT1: precision at standard recall levels (top) and distance from optimal ranking (bottom)

mostly expressed as "objective" in the collection (recall drops from 1 to less than 0.08). The same holds for the "related terms": by applying the gloss similarity semantic relatedness formulas exploiting the IEEE definitions, we achieve near-perfect recall levels (as opposed to the less than optimal ones of the first baseline) while also maintaining high precision levels. For example, most documents containing "review" are also relevant to Q5, which contains "audit", the ones containing "document" are also relevant to Q6 asking for "documentation", and so on. The WordNet based similarity proved equally useful as the gloss based one (for instance it correctly identifies the relatedness between "attribute" and "property", "procedure" and "process", and others), however we found that in some cases it may lead to several false positives, mainly due to the non-specialized nature of the employed thesaurus. For this reason, we decided to focus on the IEEE gloss similarity in these preliminary tests and to analyze the impact of the WordNet similarity more in detail in future tests.

We will now deepen the effectiveness analysis by considering queries QT1-QT4, in the form of actual text documents for which to find related documents in the collection. In this case, the queries are substantially more complex than Q1-Q6 and can possibly produce a very large number of results: for this reason, it is essential to evaluate not only which answers are returned but also their score and the induced ranking, so to assess whether the best suggestions are returned in the top positions and, thus, whether the proposed weighting scheme is effective. Figure 4 (top) shows, for QT1, the precision values obtained at different recall levels, i.e. when a given percentage of relevant documents have been found. The "standard" technique, which uses all the weights and semantic synonymy/relatedness information, is compared to non-weighted and non-semantic baselines. Notice that the standard technique achieves very high precision levels even at high recall levels: for instance, at recall level 0.6, the precision is still 1, while the baselines' precision levels have already dropped lower than 0.03. This confirms that our techniques are able to identify the most significant terms in the queries, without being misled by non-relevant ones. Figure 4 (bottom)

confirms the goodness of the retrieved results: for each alternative, the curve represents the normalized Spearman footrule distance [17] between the retrieved and the ideal ranking, i.e. the normalized sum of the absolute values of the difference between the ranks. Due to lack of space we do not show this detailed analysis for QT2-QT4, however we found that the good performance of QT1 is fully representative of all the queries. In particular, for QT1-QT4 the most relevant results are always among the first to be retrieved, and the precision at recall level 1 is always higher than 0.5 (orders of magnitude better than our baselines).

VI. Concluding Remarks

Several literature papers have highlighted possible benefits of combined knowledge engineering and software engineering approaches for specific SE tasks [2, 3, 4]. For instance, while standard reuse repositories are limited to plain syntactical search and generally suffer from low precision and recall [4], knowledge-based approaches such as [1] enhance the effectiveness of the component reuse task by proposing the usage of formal descriptions of components (in OWL) to be queried by specific graph query languages such as SPARQL. Other notable proposals have been presented, for instance, for facilitating software process assessment through formal descriptions of specific process improvement approaches such as CMMI [2, 3]. As already noted in [4], however, the discussion on integrating SE and KE approaches has been, in many cases, very academic, focusing on aspects like meta-modeling and neglecting applicability and usability.

The filtering/search approach we presented in this paper is designed to be effective and very easily applicable. In the FACIT-SME scenario, software-developing SMEs will be able to exploit all the advanced functionalities offered by these semantic foundations for a large number of tasks, such as software process and improvement and certification. Moreover, the approach does not have any prerequisite, such as the knowledge of complex formal representation/querying standards or the need of converting/updating the documentation already available in the enterprise. To this end, our proposal leverages on the strengths of both classic information retrieval (incorporating weight information [11]) and of knowledge-based techniques. In particular, semantic similarity techniques which already proved their effectiveness in a number of non-SE scenarios, from information disambiguation [9] to the querying of heterogeneous information in digital libraries and PDMSs [18], are adapted and extended in this new framework.

FACIT-SME has just started and the presented approach is the first step toward the project goals. Future work will include further analysis and refinements of the similarity techniques (especially for the WordNet-based ones), user feedback on the retrieved suggestions, Multilanguage information management and querying support, and the exploitation of other non-textual knowledge which will eventually be available in the ORM repository. Finally, the project evaluation phase, which will start soon, will involve user IT companies in actual scenarios in order to obtain useful opinions and suggestions about the quality of the proposed techniques and their improvement.

Acknowledgment

The research leading to these results has received funding from the European Community's Seventh Framework Programme managed by REA Research Executive Agency (http://ec.europa.eu/research/rea) ([FP7/2007-2013] [FP7/2007 - 2011]) under grant agreement n° 243695.

Our sincere thanks to Sonia Bergamaschi, Domenico Beneventano (UniMoRe), Gorka Benguria (ESI), Frank-Walter Jaekel (Fraunhofer IPK) and to the other project partners for their support to this research.

References

[1] C. Kiefer, A. Bernstein, J. Tappolet. Mining software repositories with ISPARQL and a software evolution ontology. In Int'l Workshop on Mining Software Repositories (MSR), 2007.

[2] K. Soydan, "An OWL Ontology for Representing the CMMI-SW Model". Proc. of 2nd Int'l Workshop on Semantic Web Enabled Software Engineering (SWESE), 2006.

[3] L. Liao, Y. Qu, H. K. N. Leung, "A software process ontology and its application". Proc. of the 4th Int'l Semantic Web Conference, 2005.

[4] H.-J. Happel and S. Seedorf, "Applications of Ontologies in Software Engineering". Proc. of 2nd Int'l Workshop on Semantic Web Enabled Software Engineering (SWESE), 2006.

[5] Aetic (Spain), Agoria (Belgium), AssInform (Italy) et al., "Position paper towards a European software strategy", presented to commissioner Viviane Reding on 24 October 2008.

[6] DG INFSO Internal Reflection Group on Software Technologies, "ITEA", April 2002.

[7] F.-W. Jaekel (Editor), "ORM Architecture and Engineering Models", FP7-SME FACIT-SME (FP7-243695), Deliverable, http://www.facit-sme.eu/FACIT-2-2010-10-18-IPK-deliverable_2_1-37b.pdf, Oct 2010.

[8] G. Benguria (Editor), "OSES Architecture and Component Specification", FP7-SME FACIT-SME (FP7-243695), Delivarable, Dec 2010.

[9] F. Mandreoli, R. Martoglia, "Knowledge-Based Sense Disambiguation (Almost) For All Structures". In Information Systems 36(2), 2011.

[10] L. Po, S. Sorrentino, "Automatic generation of probabilistic relationships for improving schema matching". In Information Systems, 36(2), 2011.

[11] G. Salton, C. Buckley, "Term-weighting approaches in automatic text retrieval". In Information Processing & Management 24(5), 1988.

[12] S. Banerjee and T. Pedersen, "Extended Gloss Overlaps as a Measure of Semantic". Proc. of the Eighteenth Int'l Joint Conference on Artificial Intelligence, pp. 805-810, 2003.

[13] S. Patwardhan, T. Pedersen, "Using WordNet Based Context Vectors to Estimate the Semantic Relatedness of Concepts". Proc. of the EACL 2006 Workshop Making Sense of Sense, pp. 1-8, 2006.

[14] C. Leacock and M. Chodorow, "Combining local context and WordNet similarity for word sense identification". In C. Fellbaum, editor, WordNet: An electronic lexical database. MIT Press, 1998.

[15] Carnegie Mellon University Software Engineering Institute, "CMMI for Development, Version 1.2" (pdf), 2006.

[16] ISO TC176, "DIS 9001:2000 Quality Management Systems - Requirement." (pdf), 1999.

[17] P. Diaconis, R. L. Graham, "Spearman's Footrule as a Measure of Disarray", Journal of the Royal Statistical Society 39(2), 262-268, 1977.

[18] F. Mandreoli, W. Penzo, S. Sassatelli, S. Lodi, R. Martoglia, "Semantic Peer, Here are the Neighbors You Want!". Proc. of the 11th Int'l Conf. on Extending Database Technology (EDBT), pp.26-37, 2008.

Inconsistency-Induced Heuristics for Problem Solving

Du Zhang

Department of Computer Science
California State University
Sacramento, CA 95819-6021
zhangd@ecs.csus.edu

Abstract

Inconsistency is ubiquitous in the real world, in human behaviors, and in the computing systems we build. Inconsistency manifests itself in data, information, knowledge, meta-knowledge, and expertise. The focus of this paper is on how inconsistency at the level of information or knowledge can be utilized as important heuristics in problem solving process. Using as an example a recent high profile question-answering system, the IBM's Watson computer system, we see the importance of reasoning about inconsistency in both Watson's tremendous success and its embarrassing moment in the first ever human-versus-machine Jeopardy! TV quiz show. We then describe a general framework for capturing inconsistency-induced problem solving heuristics. The take-home message is that inconsistency is a terrible thing to waste, and we should capitalize on what inconsistency reveals and utilize it to help improve our problem solving process.

Keywords*: inconsistency, heuristics in problem solving, inconsistency based heuristics.*

1. Introduction

Inconsistency is ubiquitous in the real world, in human behaviors, and in the computing systems we build. Inconsistency manifests itself in a plethora of phenomena at different levels in the depth of knowledge, ranging from data, information, knowledge, meta-knowledge, to expertise. Data inconsistency arises when patterns in data do not conform to an established range, distribution or interpretation. The exponentially growing volumes of data stemming from almost all types of data being created in digital form, a proliferation of sensors and sensor networks, and other sources such as social networks, complex computer simulations, space explorations, and high-resolution imagery and video, have made data inconsistency an inevitability. Information inconsistency occurs when meanings of the same data values become conflicting or when the same attribute for an entity has different data values. Knowledge inconsistency happens when propositions of either declarative or procedural beliefs, in explicit or tacit form, yield antagonistic outcomes for the same circumstance. Inconsistency can also emerge from meta-knowledge or from expertise. How to manage and reason in the presence of inconsistency in computing systems is a very important issue in semantic computing, social computing, and many other data-rich or knowledge-rich computing paradigms.

The focus of this paper is on how inconsistency at the level of information or knowledge can be utilized as important heuristics in problem solving process. Heuristics are strategies that rely on readily accessible but loosely applicable information to control problem solving processes [11]. Heuristic methods, such as a rule of thumb, an educated guess, an intuitive judgment, domain knowledge guided search, or common sense, are often deployed for problem solving processes where there does not exist an algorithmic approach or the search is exhaustive.

When confronted with inconsistency, human beings attempt to reason from inconsistency to consistency [7, 9]. Parallel to the cognitive consistency theory of human problem solving, generating and utilizing inconsistency induced heuristics is also conducive in computing system based problem solving process.

The rest of the paper is organized as follows. Section 2 offers a brief review on some related work. Section 3 describes the importance of reasoning about inconsistency through a recent high profile question-answering system, the IBM's Watson computer system, that demonstrates both its tremendous success and some embarrassing moment in the first ever human-versus-machine Jeopardy! TV quiz show. This case epitomizes the usefulness of inconsistency induced heuristics. In Section 4, we propose a framework where inconsistency in information or knowledge can be translated into useful problem-solving heuristics. Section 5 concludes the paper with remarks on future work.

2. Related Work

In cognitive science, the theory of *cognitive dissonance* underpins human behaviors toward

inconsistency [3, 5]. Human cognitions fall into irrelevant, consonant or dissonant categories. When confronted with dissonant cognitions or inconsistent knowledge, human beings will be in a cognitive, emotional, psychological, or behavioral dissonant state. As a result, individuals will experience some unpleasant psychological tension which has motivational properties not unlike those of hunger or thirst, and they will be motivated to reduce dissonance through behavioral changes.

There are cognitive dissonance related heuristics that are used to condition human behavior in the face of inconsistency, for instance, changing attitudes, beliefs or actions, justifying, blaming, or denying [3].

A number of reasoning paradigms in AI, such as resolution refutation or default reasoning, are based on recognizing and utilizing inconsistency as important heuristics [2].

As the basis for effective and powerful heuristics, inconsistency has also been incorporated into computing system based problem solving processes [19, 20]. Here we use digital forensics as an example to see how ample the opportunities are in taking advantage of inconsistency for problem solving objective. In the field of digital image forensics, there are a number of effective inconsistency based heuristics in detecting forgeries. The first type of heuristics is based on lighting inconsistency [4, 8]. In digital images, lighting inconsistency refers to a circumstance where shapes, colors, locations or directions, or number of sources of reflected light on objects are inconsistency. For instance, when specular highlights (small white specks of reflected light in people's eyes) in an image result in inconsistent lighting directions, number of sources, or shapes, it is evidence of image altering [4, 8]. So any presence of lighting inconsistency can be used as an effective heuristic to identify image tampering. The second type of heuristics is based on the local noise level inconsistency [10]. Normally in an authentic digital image, the noise is uniformly distributed across the entire image. Tools used to alter images often inject locally randomized noise to the forged regions so as to conceal the evidence of tampering, thus creating inconsistencies in the levels of noise in the image. The work in [10] described an approach that detects tampering through recognizing noise level inconsistencies in an image. The approach is based on decomposing an image into segments of various noise levels. The third type of heuristics is based on the blocking artifact inconsistencies [15]. In JPEG images, due to the fact that digital camera manufacturer and image processing software often utilize different JPEG quantization tables to balance between compression ratio and image quality, different blocking artifacts will be injected into the images as a result. When a digitally forged image is created from several sources, the resulting image invariably contains different sorts of compression artifacts. In addition, many image manipulation operations such as image splicing, re-sampling, or skin optimization, will generate differential blocking artifacts. These blocking artifact inconsistencies can serve as very effective heuristics in detecting tampered images.

Many other areas of endeavors can also benefit from recognizing the clues inconsistency helps reveal, and devising and utilizing inconsistency induced heuristics in the problem solving processes [19, 20].

3. What Watson Has Demonstrated?

A recent success story on utilizing inconsistency as an effective tool in problem solving is the IBM's Watson computer system. Watson was developed with the goal to compete at the human champion level in real time on *Jeopardy!*, an American TV quiz show that had its original series premiere in March 30, 1964, and that started its current series premiere in September 10, 1984. Early in 2011, Watson competed in the show's first ever human-versus-machine, two-game combined-point matchup. Broadcasted in three *Jeopardy!* episodes during February 14-16, 2011, Watson was pitted against two top human players, Ken Jennings, the record holder for the longest championship streak at 74 games, and Brad Rutter, the biggest all-time money winner on *Jeopardy!*. Watson was triumphant over the human players and walked away with the first prize of $1 million.

IBM's Watson computer system is an open domain question-answering system that was developed with advanced natural language processing, information retrieval, knowledge representation and reasoning, and machine learning technologies [6, 13]. Watson has a massive parallel computing infrastructure: 90 IBM Power 750 servers with each server having four POWER 7 processors, each POWER 7 processor having 8-core with each core capable of running 4 threads, and 16 terabytes of RAM. Underpinning Watson's open-domain question-answering capability is IBM's *DeepQA* technology that allows for question analysis and decomposition, hypothesis generation and filtering, evidence retrieval and scoring, merging and ranking answers, and confidence estimation (see Figure 1) [6]. Watson has access to its local resources of 200 million pages of structured and unstructured content including dictionaries, encyclopedias, news articles, and other reference materials [13].

A number of components in Watson use inconsistency as heuristics for hypothesis and evidence scoring. For instance, temporal reasoning is deployed in Watson to "detect inconsistencies between dates in the clue and those associated with a candidate answer" [6]. Another scoring component, geospatial reasoning, is capable of detecting spatial inconsistencies stemming from conflicting spatial relations such as "directionality, borders, and containment between geoentities" [6]. When scores for individual pieces of evidence are combined to produce overall evidence profiles for candidate answers, inconsistencies will be examined from dimensions such as name consistency or theory consistency [6].

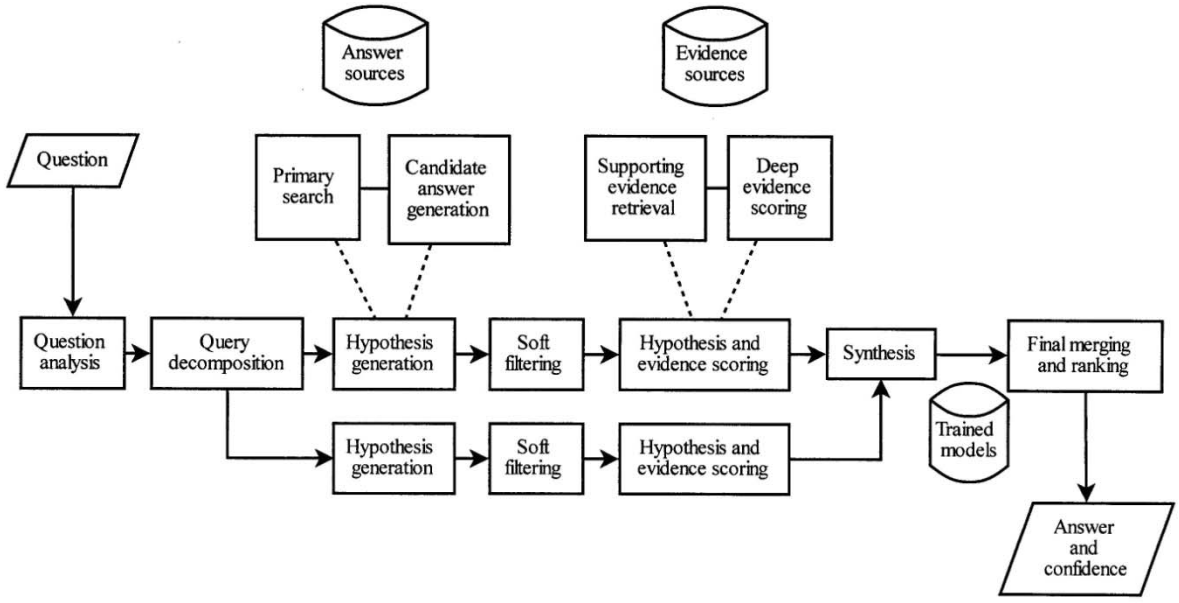


Figure 1. IBM DeepQA High-Level Architecture [6].

Despite the aforementioned inconsistency-based heuristics in place in its *DeepQA* engine, Watson flubbed during the Final *Jeopardy!* question at the end of the first match. The category for the Final *Jeopardy!* question was US Cities, and the answer was: "Its largest airport is named for a World War II hero; its second largest, for a World War II battle." Whereas both human players responded correctly with "What is Chicago?", Watson's response was a confused one: "What is Toronto?????".

The case of Watson's response of "Toronto" to the question category of "US Cities" constitutes what is referred to as anti-subsumption inconsistency in [16, 18]. Given a taxonomy of concepts, when there is no longer a subsumed relationship between a sub-concept (Toronto) and its super-concept (US Cities), this antagonistic circumstance is called anti-subsumption inconsistency. The manager of the Watson project at IBM Research, David Ferrucci, offered the following explanation on Watson's flub [14]:

> *First, the category names on Jeopardy! are tricky. The answers often do not exactly fit the category. Watson, in his training phase, learned that categories only weakly suggest the kind of answer that is expected, and, therefore, the machine downgrades their significance. The way the language was parsed provided an advantage for the humans and a disadvantage for Watson, as well. "What US city" wasn't in the question. If it had been, Watson would have given US cities much more weight as it searched for the answer. Adding to the confusion for Watson, there are cities named Toronto in the United States and the Toronto in Canada has an American League baseball team. It probably picked up those facts from the written material it has digested. Also, the machine didn't find much evidence to connect either city's airport to World War II. (Chicago was a very close second on Watson's list of possible answers.) So this is just one of those situations that's a snap for a reasonably knowledgeable human but a true brain teaser for the machine.*

What Watson has taught us is that there are still gaps in Watson's repertoire of hypothesis and evidence scoring components. The steps in soft filtering, deep evidence scoring, and final merging and ranking can be further strengthened by incorporating additional inconsistency based heuristics so as to avoid producing glaring conflicting answers.

4. Inconsistency-Induced Heuristics

Heuristics are criteria or approaches to be used to determine which course of action among several alternatives is the most promising one toward accomplishing some objective, and they are meant to strike a balance between keeping the criteria simple and keeping them discriminate fittingly between more promising and less promising choices [11]. Toward the aforementioned goals, inconsistency induced heuristics can be defined based on inconsistency categories, causes, morphology, and desired actions.

In this paper, we focus our attention on inconsistency at levels of information and knowledge in the depth of knowledge hierarchy of data, information, knowledge, meta-knowledge, and expertise. Table 1 summarizes the

causes of inconsistency and possible models of heuristics. Table 2 captures some of the most general categories and morphologies of inconsistency.

Table 1. Inconsistency Causes and Models.

Cause	*Ontology related reasons*
	Epistemic conflicts
	Conflicting defaults
	Lack of complete information
	Uncertainty
	Defeasible inheritance induced
	Assertion lifting
	Reliability of information sources
	Belief revision due to cognitive penetrability
	Deliberate act
	Concept forming process in description logic
Possible models of heuristics	*Closed-world reasoning*
	Mutually exclusive principle
	Locality of inconsistency principle
	Context principle
	Shortest path

Inconsistency can be caused by a whole host of circumstances. Ontologically, conflicting concept subsumptions, contradicting membership relations, or lack of constraints on ontology specifications can be causes for inconsistency. Epistemologically, different agents may hold beliefs that are inconsistent with each other. Conflicting defaults, lack of information, information source reliability can all contribute to inconsistency. In property inheritance, the case in which an inherited property can be overridden may result in inconsistency. When an agent's knowledge is compartmentalized, lifting assertions from one compartment to another can introduce inconsistency. Finally, inconsistency can also be the result of some deliberate act.

Heuristics in general can be derived or discovered under some simplified models [11]. The types of models that can be utilized in defining inconsistent phenomena induced heuristics include: closed world assumption to be used in calculating number of positive and negative pieces of evidence; mutually exclusive principle applied when sets of entities are mutually exclusive and jointly exhaustive; locality of inconsistency to be used to identify scopes or clusters of inconsistent phenomena [21]; context principle relying on analyzing the context for clues; and shortest path giving preference to shorter paths in inheritance networks [2].

When cast in the light of logical formalism, inconsistency can take on the following morphologies [16, 18]: (1) complementary: an atom and its negation; (2) mutually exclusive: mutually exclusive predicates that are syntactically different and semantically opposite of each other; (3) incompatible: an atom and the negation of its syntactically different but logically equivalent relation; (4) anti-subsumption: the antagonistic situation where a subtype is no longer subsumed by its supertype; (5) anti-supertype: the converse of the anti-subsumption where the negation of the supertype and one of its subtypes hold at the same time; (6) asymmetric: a symmetric relation and its negation both hold; (7) anti-inverse: a relation and the negation of its inverse relation exist; (8) mismatching: a concept and the negation of one of its conjunctive and substantiating sub-concepts hold; (9) disagreeing: propositions having reified but disagreeing quantities for the same attribute; (10) contradictory: relations containing attribute values that violate type restrictions or integrity constraints; (11) precedence: opposite precedence relationships being asserted for the same set of entities; and (12) inconsistent probability value type (*iProbVal*): if $\Delta \models L$, the probability value for L is outside the region specified by the extreme values of probabilities for Δ.

Table 2. Categories and Morphologies of Inconsistency.

Category	**Morphology**
Logical	*Complementary*
	Mutually exclusive
	Incompatible
	Anti-subsumption
	Anti-supertype
	Asymmetric
	Anti-inverse
	Mismatching
	Disagreeing
	Contradictory
	Precedence
	iProbVal
Temporal	*Partial temporal inconsistency*
	Complete temporal inconsistency
Spatial	*Topological inconsistency*
	Location/attribute inconsistency
	Structural/semantic constraint violations
Ontological	*Conflicting concept subsumptions*
	Contradicting membership relations
Argumentation	*Rebutting argument*
	Undercutting argument
	Counterargument
Contextual	*Inconsistent with context*
	Inconsistent to expectation

When the concept of time is incorporated, we have partial temporal inconsistency, and complete temporal inconsistency [17]. In the geospatial context, there are violations of geometrical properties and spatial relations (Topological inconsistency), spatial objects having conflicting geometric locations (location/attribute inconsistency), and structural/semantic constraint

violations [12]. In the ontological category, there are conflicting concept subsumptions and contradicting membership relations [20]. From the perspective of argumentation, inconsistency manifests itself in terms of rebutting argument which has a claim that is the negation of another argument's claim, undercutting argument which has a claim that contradicts some of the assumptions of another argument, and counterargument which is either rebutting or undercutting argument with regard to another argument [1]. Finally inconsistency can also arise from contextual related circumstances where responses to events are either inconsistent with the context or inconsistent with regard to expectation.

Before describing a general framework for inconsistency induced heuristics, we need to have some concepts in place. Let $\Im$ be a set containing (possible) conflicting circumstances or contradicting pieces of evidence and $\varphi(\Im)$ be a set of alternative outcomes or next steps in the presence of $\Im$. Let $\aleph$ denote elements in $\varphi(\Im)$:

$$\aleph = \{\delta_i \mid \delta_i \in \varphi(\Im)\}.$$

We use $\Re$ to represent a set of k attributes for the context of $\delta_i \in \aleph$:

$$\Re(\aleph) = \{\alpha_k(\delta_i) \mid \delta_i \in \aleph \wedge k \in [1,.., m]\}$$

Finally, ϖ is a set of weights for $\Re$:

$$\varpi(\Re) = \{\omega_k(\alpha_k(\delta_i)) \mid \delta_i \in \aleph \wedge \omega_k(\alpha_k(\delta_i)) \in [0, 1] \wedge \sum_{k=1}^{m} \omega_k(\alpha_k(\delta_i)) = 1\}$$

Now we are in a position to define a general framework for inconsistency induced heuristics.

Framework for Inconsistency Induced Heuristics

Input: $\Im$, $\aleph$, $\Re$, ϖ
Output: most desirable outcome: $\mathcal{M}$

- Identify category, morphology and cause of inconsistency[1]
- Select appropriate model of heuristics
- Calculate prognosis for each outcome
  Let ω_{ki} be the k^{th} weight for the i^{th} outcome;
  Let α_{ki} be the k^{th} attribute for the i^{th} outcome;
  for each $\delta_i \in \aleph$ do {
    $\text{prognosis}(\delta_i) = \sum_{k=1}^{m} \omega_{ki}\alpha_{ki}$
  }
- Determine the most desirable outcome
  $\mathcal{M} = \underset{i}{\text{argmax}}\{\text{prognosis}(\delta_i)\}$

return($\mathcal{M}$); □

Example. Let us use Watson's response to the airport clue as an example, and see how Watson could have avoided responding with an obvious wrong answer, had Watson been equipped with some scoring components based on inconsistency induced heuristics in its soft filtering, deep evidence scoring and/or final merging and ranking (Figure 1). To facilitate the discussion, assume the clue in the Final *Jeopardy!* question can be represented as follows where $\mathscr{C}$ is the domain of all north America cities and $\mathscr{A}$ is the domain of all airports in north America:

$$\exists x \in \mathscr{C}.\exists y \in \mathscr{A}.\exists z \in \mathscr{A}.\ [\text{city}(x) \wedge \text{us\_city}(x) \wedge \text{has\_largest\_airport}(x, y) \wedge \text{has\_2ndlargest\_airport}(x, z) \wedge \text{namedfor\_WWII\_hero}(y) \wedge \text{namedfor\_WWII\_battle}(z) \wedge \neq(y, z)].$$

Though no official analysis is available for Watson's performance in the televised *Jeopardy!* Show during February 14-16, 2011, from the initial explanation in [14], we can speculate that[2]

- Watson's hypothesis generation module produced several candidate answers including Toronto, Canada ("*the Toronto in Canada has an American League baseball team*"); Chicago ("*Chicago was a very close second on Watson's list of possible answers*"); and several US cities by the name Toronto ("*there are cities named Toronto in the United States*"). Let's assume that "Toronto, Iowa", "Toronto, Kansas", "Toronto, Ohio", and "Toronto, South Dakota" are the US cities Watson considered.
- At least three of the aforementioned cities survived the soft filtering, deep evidence scoring, and final merging and ranking stages to become candidates for the answer. These cities were in the following order of Watson's confidence: Toronto, Canada, Chicago, and a third city named Toronto in US.
- The other reasons given to explain Watson's behavior (weak connection between category and type of response desired, parsing of the language, and US Cities not appearing in the question) should not be as critical to the mistake as the lack of some inconsistency based sanity checking component to guarantee the consistency between the category and the candidate answers. When the system does not have adequate information or knowledge to respond to the question at hand, high precision becomes a more important issue.
- It seems that the "spatial containment" scoring component [6] failed in this case.
- Watson seemed to rely on other indirect evidence such as sport teams as part of evidence scoring (*Toronto in Canada has an American League baseball team*).

Using ap1, ap2, n1, n2, and bt to denote the following properties (attributes) for a city: has_largest_airport, has_2ndlargest_airport, namefor_WWII_hero, and namedfor_WWII_battle, and a major league baseball team, respectively, we have the following context for an

[1] Detecting and discerning category, morphology and cause of inconsistency are prerequisites to determining the type of heuristics to deploy. Space limit does not allow for elaboration in this paper.

[2] The quotes in italic were taken from David Ferrucci's explanations in [14] on why Watson responded with Toronto, Canada.

inconsistency induced heuristic (information in Table 3 was obtained via Google search: {Pearson, Downsview} for Toronto_Canada, {O'Hare, Midway} for Chicago, {Maquoketa, Clinton} for Toronto_IA, {Yates, Heir} for Toronto_KS, {Eddie Dew, Granatir} for Toronto_OH, and {Brookings, Clear Lake} for Toronto_SD).

Table 3. Possible Context for Sanity Checking.

city (candidate answer)	us_city	ap1	ap2	n1	n2	bt
Toronto_Canada	–	+	+	–	–	+
Chicago_IL	+	+	+	+	+	+
Toronto_IA	+	+	+	–	–	–
Toronto_KS	+	+	+	–	–	–
Toronto_OH	+	+	+	–	–	–
Toronto_SD	+	+	+	–	–	–

Since the set of incompatible evidence {city(Toronto_Canada), ¬us_city(Toronto_Canada)} would constitute an instance of anti-subsumption inconsistency [16, 18], hence we have $\Im$ = {city(Toronto_Canada), ¬us_city(Toronto_Canada)}. Given a possible set of weights for attributes in the context, ϖ ={0.5, 0.1, 0.1, 0.1, 0.1, 0.1} and let attribute values "+" and "–" be 1 and 0, respectively, then the contextual heuristic would return argmax{prognosis(δ_i)} = Chicago.

5. Conclusion

In this paper, we examined the general issue of inconsistency induced heuristics, and used the Watson question-answering system as a salient example to drive home the importance of taking full advantage of what inconsistency has to offer and capitalizing it to help improve problem solving processes. When utilizing inconsistency as effective heuristics, we need to understand various issues and dimensions antagonistic circumstances entail. As quoted in [7], Henri Poincare once said that contradiction is the prime stimulus for scientific research. Indeed, inconsistency is a terrible thing to waste.

Future work can be pursued in the following directions. Details of the proposed framework still need to be worked out. The specific patterns of heuristics under various models need to be fleshed out. How domain-specific information can be factored into the framework is another issue worth exploring.

Acknowledgement. We would like to express our sincere appreciations to the anonymous reviewers for their comments that help improve the content and the presentation of this paper.

References

[1] Besnard, Ph. & Hunter, A. *Elements of Argumentation*. MIT Press, 2008.

[2] Brachman, R.J. & Levesque, H.J. *Knowledge Representation and Reasoning*. Morgan Kaufmann Publishers, 2004.

[3] Cognitive Dissonance, Wikipedia, http://en.wikipedia.org/wiki/Cognitive_dissonance.

[4] Farid, H. Seeing Is Not Believing. *IEEE Spectrum*, Vol. 46, No.8, 2009, pp. 44-51.

[5] Festinger, L., *A Theory of Cognitive Dissonance*. Stanford University Press, Stanford, CA, 1957.

[6] Ferrucci, D., et al. Building Watson: An Overview of the DeepQA Project. *AI Magazine*, Fall 2010, pp.59-79.

[7] Gotesky, R. The Uses of Inconsistency. *Philosophy and Phenomenological Research*. Vol. 28, No. 4, 1968, pp.471-500.

[8] Johnson, M.K. & Farid, H. Exposing Digital Forgeries by Detecting Inconsistencies in Lighting. *Proc. of ACM Multimedia and Security Workshop*, 2005.

[9] Johnson-Laird, P.N., Legrenzi, P. & Girotto, V. Reasoning from Inconsistency to Consistency. *Psychological Review*, Vol. 111, No. 3, 2004, pp.640-661.

[10] Mahdian, B & Saic, S. Using Noise Inconsistencies for Blind Image Forensics. *Image and Vision Computing*, Vol.27, 2009, pp.1497-1503.

[11] Pearl, J. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley Publishing, 1984.

[12] Rodriguez, A. Inconsistency Issues in Spatial Databases. In L. Bertossi et al (eds.) *Inconsistency Tolerance*, LNCS 3300, Springer-Verlag, 2004, pp.237-269.

[13] Watson (artificial intelligence software), Wikipedia, http://en.wikipedia.org/wiki/Watson_(artificial_intelligence_software).

[14] Watson on Jeopardy, http://asmarterplanet.com/blog/2011/02/watson-on-jeopardy-day-two-the-confusion-over-an-airport-clue.html.

[15] Ye, S., Sun, Q. & Chang, E.C. Detecting Digital Image Forgeries by Measuring Inconsistencies of Blocking Artifacts. *Proc. of the IEEE International Conference on Multimedia and Expo*, 2007, pp.12-15.

[16] Zhang, D. Taming Inconsistency in Value-Based Software Development. *Proceedings of the Twenty First International Conference on Software Engineering and Knowledge Engineering*, Boston, July 2009, pp.450-455.

[17] Zhang, D. On Temporal Properties of Knowledge Base Inconsistency. *Springer Transactions on Computational Science* V, LNCS 5540, 2009, pp.20-37.

[18] Zhang, D. Toward A Classification of Antagonistic Manifestations of Knowledge. *Proceedings of Twenty Second International Conference on Tools with Artificial Intelligence*, Arras, France, 2010, pp.375-382.

[19] Zhang, D. Inconsistencies in Information Security and Digital Forensics. *Proceedings of the Eleventh IEEE International Conference on Information Reuse and Integration*, Las Vegas, NV, August 2010, pp.141-146.

[20] Zhang, D. Inconsistency: The Good, The Bad, and The Ugly. *International Transactions on Systems Science and Applications*, Vol.6, No.2/3, August 2010, pp.131-145.

[21] Zhang, D. On Localities of Knowledge Inconsistency. *International Journal of Software Science and Computational Intelligence*, Vol.3, No.1, 2011, pp.61-77.

Mapping CommonKADS Knowledge Models into PRR

Nicolas Prat
ESSEC Business School
Paris, France
prat@essec.edu

Jacky Akoka
CEDRIC-CNAM & Institut Telecom / TEM Research
Paris, France
akoka@cnam.fr

Isabelle Comyn-Wattiau
CEDRIC-CNAM & ESSEC Business School
Paris, France
wattiau@cnam.fr

***Abstract*—This paper aims at supporting the knowledge engineering process by proposing an approach to map CommonKADS knowledge models into specifications based on the Production Rule Representation (PRR) language. This approach starts by proposing a metamodel of CommonKADS knowledge models. We define the concept of inference group, required to perform the mapping transformations, and an algorithm that identifies inference groups automatically. We then proceed to the definition of transformation rules. The latter are applied to map CommonKADS knowledge models into a set of PRR production rulesets, combined with UML activity diagrams.**

Keywords – CommonKADS, Model Driven Approach, Production Rule Representation, metamodel, mapping

I. INTRODUCTION

It is well accepted that the cross-fertilization between software engineering and knowledge engineering can significantly improve the design of knowledge-based and expert systems. The aim of this paper is to present an approach enabling the mapping of CommonKADS knowledge models into specifications based on the Production Rule Representation language (PRR). The following aspects mainly motivate our work:

- The CommonKADS methodology is a widely used knowledge engineering method. Even if CommonKADS is more generally suited to systems where knowledge plays an important role, it does not necessarily lead to the development of expert systems. It focuses more on knowledge engineering issues. CommonKADS knowledge models are therefore specified at the analysis level. It can be considered as the Computational Independent Model (CIM) level of the Model Driven Approach (MDA [8]).
- On the other hand, PRR represents a relatively recent standardization effort conducted by OMG [10]. PRR is situated at the Platform Independent Model (PIM) level of MDA. We assume that the knowledge system will be implemented as production rules. PRR represents this knowledge in the design phase.
- There exists a possible convergence between CommonKADS knowledge models and PRR, since both are based on UML. However, the two are not completely related. Therefore, a mapping of CommonKADS knowledge models into PRR is an open research problem. We argue that such a mapping can be very useful.

To perform such a mapping, we propose an algorithm aiming at the definition of "inferences groups" as well as several transformation rules.

The rest of the paper is organized as follows. Section II is devoted to a presentation of related work. We describe in Section III the main concepts of the PRR metamodel. Our CommonKADS metamodel is presented in Section IV. The algorithm and the transformations used to map CommonKADS knowledge models into PRR are described in Section V. Finally, Section VI is devoted to the conclusion and future research.

II. RELATED WORK

CommonKADS [13] is a widely referenced methodology for designing knowledge-based systems [2,5,7,14,15]. As mentioned in the previous section, there exists a possible convergence between CommonKADS knowledge models and PRR, since both are based on UML. However, the two are not completely related. [1] offers an approach enabling a mapping between the knowledge model and JESS, based on a UML profile. However, the knowledge model presented in their approach is less detailed than the one presented in this paper. Moreover, this mapping is situated at the Platform Specific Model (PSM) level of MDA. Finally, let us mention that the description of the design level in CommonKADS [13] is relatively limited, especially compared to the richness of its analysis level as well as its knowledge model. The design level consists mainly in the "direct" mapping of concepts of the analysis level (including knowledge model concepts) as design classes. This approach does not help much the designer, hence the interest of a bridge with PRR in order to switch to the design level. [16] proposes to combine OWL, SWRL and JESS to build a knowledge level modeling. [17] describes and compares several rule languages with the objective of modeling

business processes. [4] is a recent and detailed state-of-the-art on rule-based systems.

III. MAIN CONCEPTS OF THE PRR METAMODEL

The Production Rule Representation language (PRR) [10] has been proposed by OMG for high-level (tool-independent) representation of rules. Fig. 1 is an excerpt of the metamodel of PRR, with the main concepts relevant in this research.

Production rules are the central concept of PRR. Production rules may reference and modify classes (possibly defined in a UML [11] class diagram). Rules are grouped into rulesets. A ruleset is a collection of rules with a particular mode of execution (operational mode). The mode of execution is either sequential or inferencing. In the *sequential* mode, rule execution order is determined by the sequence of rules in the ruleset. In the *inferencing* mode, the inference engine controls rule execution order. PRR currently supports forward chaining only, and suggests RETE [3] as a possible algorithm. A PRR rule may have a priority. Priorities are used for conflict resolution, i.e. to choose the rule to execute when there are several candidate rules.

A production rule is typically represented as *if [condition] then [action-list]*. An action may be the invocation of an operation associated with a class, the assignment of a value to an expression, or an action that updates the state of the rule engine (e.g. an action that asserts – creates – a new object).

Variables may be defined at the ruleset level or at the rule level. Rule variables are used for binding.

PRR rules can be represented formally, based on an extension of the Object Constraint Language (OCL) [9]. Imperative expressions are used to represent rules actions. Navigation call expressions are used to navigate within or between objects.

PRR is a powerful language for expressing rules, as illustrated in [12] in the domain of data warehousing and OLAP.

IV. MAIN CONCEPTS OF COMMONKADS KNOWLEDGE MODEL

CommonKADS knowledge models can be specified diagrammatically, and textually using the CommonKADS Modeling Language (CML). CML is a structured, semi-formal language. In [13], the syntax of CML is described, but the authors do not provide a metamodel of the CommonKADS knowledge model. Metamodels of the CommonKADS knowledge model have been proposed in the literature, but they are too succinct to be useful in our approach [1,6]. Our metamodel, defined as a UML class diagram, is presented in Fig. 2. It describes the main concepts of the CommonKADS knowledge model.

A knowledge model has three components: *domain knowledge*, *inference knowledge* and *task knowledge*. Domain knowledge represents the structure of the knowledge system, while inference knowledge and task knowledge represent its behavior. Another key difference is that inference and task knowledge are domain-independent, thus facilitating their reuse across several domains. For example, the assessment task may be used in several domains. The mapping of inference and task knowledge with domain knowledge is performed through the concept of knowledge role.

The structure of domain knowledge is described in a *domain schema*. This schema is represented as a UML class diagram. The main elements of a domain schema are concepts, relations and rule types. Concepts and relations correspond to the UML concepts of class (without operations) and association respectively. Rule types are specific to knowledge systems. They are prominently implication rule types, relating an antecedent concept to a consequent concept. The connection symbol specifies the semantics of the implication. The cardinalities of the antecedent (respectively the consequent) indicate the minimum and maximum number of expressions of the antecedent class (respectively the consequent class) in an instance of the rule type. Besides implication rule types, constraint rule types may be defined. Constraint rule types are internal to a concept. They can be considered as integrity constraints defined on this concept.

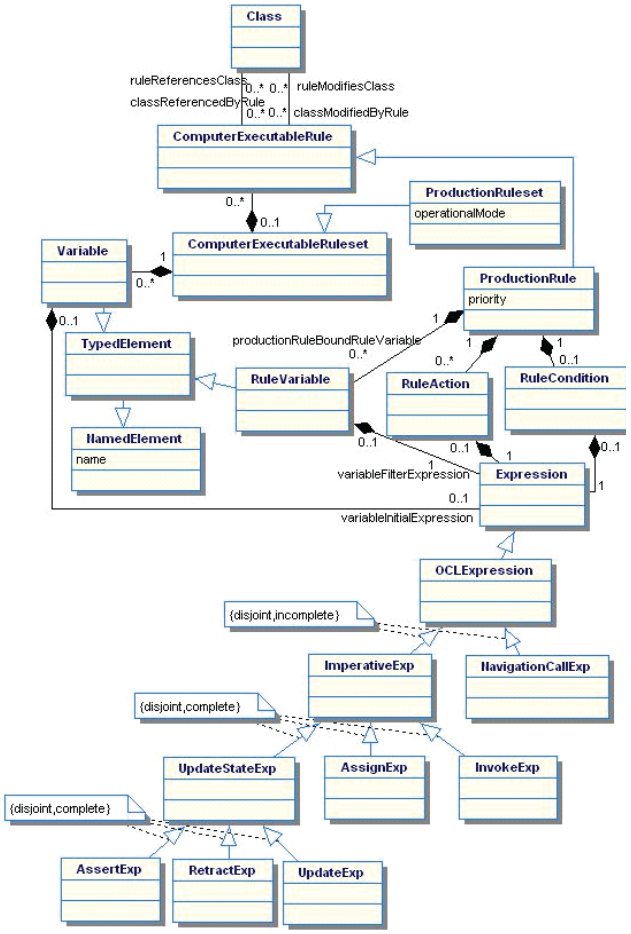


Figure 1. Main concepts of the PRR metamodel [10]

Knowledge bases are instances of domain schemas. They are primarily made of rule type instances.

The behavior of the knowledge system is represented with task knowledge and inference knowledge. Tasks, task methods, inferences and transfer functions are related through functional decomposition. A *task* may be performed through one (or possibly several alternative) *task method*(s) (OR-decomposition). A task method is then specified by its functions (inferences, transfer functions or tasks), related together by a control structure (AND-decomposition). The tasks of task methods are in turn OR-decomposed into task methods, etc… Inferences and transfer functions appear at the lowest decomposition level. An *inference* carries out a primitive reasoning step. Its internal structure may itself be complex but, in the knowledge model, it is represented as a black box. *Transfer functions* are used to communicate with the external world (information flows between the system and the user, at the initiative of the system or of the user). The transfer function "obtain", whereby the system asks the user to enter an information item, is frequently used. In this paper, we will focus on this transfer function.

Knowledge roles will be used to connect task and inference knowledge to domain knowledge. *Dynamic knowledge roles* are the inputs or outputs of functions. These roles will generally map to concepts in the domain schema. In the control structure of a task method, *intermediate knowledge roles* may be used (i.e. roles that are not input or output roles of the task specified by the task method). *Static knowledge roles* specify the collection of domain knowledge used inside an inference. A static knowledge role will generally map to an implication rule type in the domain schema.

Domain mappings perform the mapping of knowledge roles to the domain schema. In some cases, a role may map to a collection (a set or a list) of a concept.

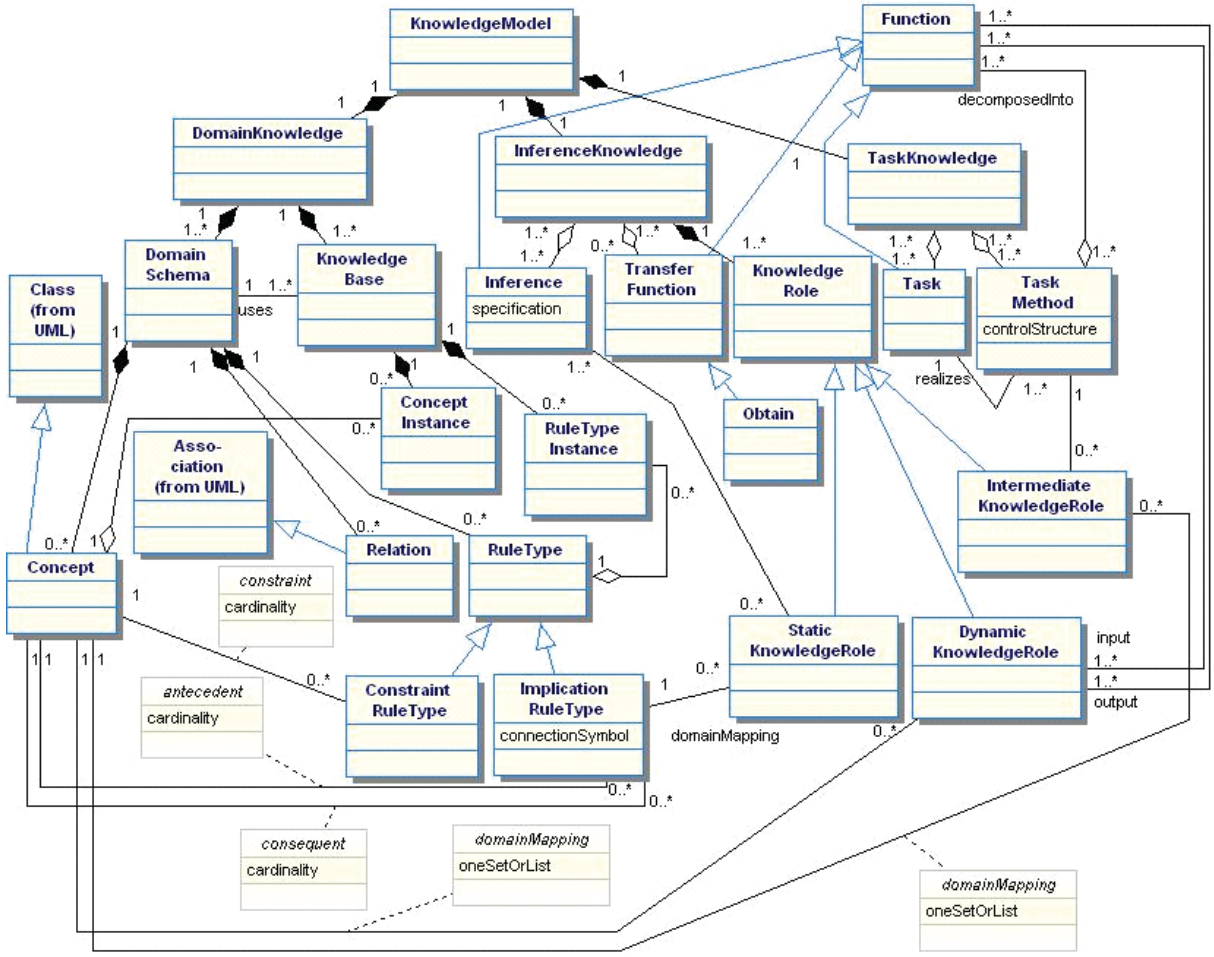


Figure 2. Metamodel of the CommonKADS knowledge model

V. MAPPING COMMONKADS KNOWLEDGE MODELS INTO PRR

A. Overview of our Approach

A specificity of our approach is to use PRR to represent knowledge in the design phase. Thus, our approach consists in mapping CommonKADS knowledge models into PRR. We assume that the knowledge system will be implemented as production rules in an expert system. In this context, the choice of PRR is appropriate.

Our approach is divided into two steps:

1. Before applying the mapping transformations, groups of inferences and transfer functions (more particularly, "obtain" transfer functions), are defined in the knowledge model. To this end, we propose an algorithm for defining such groups, called "inference groups". This algorithm operates on the control structure of task methods, represented as UML activity diagrams.
2. The transformations are then applied to map the knowledge model to the design level. The result is a set of PRR production rulesets, combined with UML activity diagrams.

B. Inference Groups

Before applying the mapping transformations, we need to group inferences and transfer functions (more specifically "obtain" transfer functions, on which we focus in this paper) into what we will call "inference groups". The rationale for this is to prepare for the mapping of inferences into PRR production rulesets. Inferences in CommonKADS are often more fine-grained than PRR can represent with a single production ruleset. It is therefore advisable, when possible, to group inferences and then map them into a single PRR production ruleset. This way, the sequencing of the inferences inside the group will be represented inside the PRR production ruleset, instead of having to represent this sequencing with a complementary formalism (in our case, UML activity diagrams). Furthermore, "obtain" transfer functions can in certain cases be mapped using variables in PRR production rulesets. However, to this end, they need to be grouped with an inference (or inferences) following them. The resulting inference group will be mapped into a single production ruleset in PRR. We present below the properties that must be satisfied by inference groups.

Property 1: An inference group is a group of two or more consecutive actions within the same control structure (i.e. inside the same "linear group of actions").

In our approach, we define a "linear group of actions" as one (or possibly more) action(s) not separated by a control structure. For example, actions separated by a decision node are in different control structures; actions in the "setup" part, the "test" part and the "body" part of an "until" loop each belong to different linear groups of actions. Our concept of "linear group of actions" is a specialization of the concept of ActivityGroup defined in UML [11]. Within a "linear group of actions", actions are linearly ordered. This follows from the fact that (1) these groups do not span across multiple control structures, and (2) we do not manage parallelism between actions (this could be the object of a future research).

We require actions of an inference group to be within the same control structure since the idea of defining an inference group is to map this group into a single PRR production ruleset, and control structures may not be used to combine the production rules within a production ruleset (as described in Section III, production rules in a production ruleset may only be executed sequentially or in forward-chaining mode).

Property 2: The actions of an inference group may only be inferences or "obtains". An inference group consists of one (or a series of) inference(s), possibly preceded by one (or several) obtains.

Inference groups may not contain tasks, because tasks are too high level to be mapped directly into PRR. In CommonKADS, a task needs to be further decomposed, as explained previously. In addition, inference groups do not integrate actions on roles (stereotyped actions <<WriteVariable>>, <<AddVariableValue>> and <<RemoveVariableValue>>), since these operations on variables (roles) are an important part of the global control structure of a task method, and therefore we avoid merging them with other actions into an inference group.

An inference group needs to contain at least one inference since a PRR production ruleset, into which the inference group will be mapped, is a collection of rules; and in CommonKADS, it is through inferences that rules are executed. The inference group may start with one (or several) obtain(s), which will be mapped using the concept of variable at the beginning of the PRR production ruleset.

Property 3: In an inference group, the outputs of "obtains" and of intermediary inferences are not used as inputs to actions or structured activity nodes outside the inference group.

When "obtains" and inferences will be merged into a single inference group, the output of the resulting inference group will be the output of the last inference of the inference group; the other outputs will no longer be represented explicitly, they will be internal to the inference group. Consequently, the outputs of an "obtain" or of an intermediary inference should not be needed outside the inference group.

C. The Inference-Group Building Algorithm

To identify inference groups within the activity diagram representing the control structure of a task method, the algorithm proceeds in two steps:

1. Grouping of actions into linear groups of actions.
2. Identification of inference groups within each of the linear groups of actions.

In step 1, the identification of linear groups of actions proceeds by identifying the first action of each group, and then recursively following the links (activity edges) to find the next action, until the last action of the group is met. Initially, all actions of the activity diagram are unmarked; actions are marked as they are incorporated into linear groups of actions.

An action is the first action of a linear group of actions if it is the target of an activity edge from a control node (e.g. a decision node), or if it is not the target (directly or through pins) of another action. Similarly, an action is the last action of a linear group of actions if it is the source of an activity edge to a control node (e.g. a merge node), or if it is not the source (directly or through pins) of another action. To find the following action of a given action, activity edges are followed. Since an action may have several indirect successors, we need to find the immediate successor. To this end, we use the constraint that the following action of an action may not be the target of another action that is not already present in the linear group of actions. Finally, in line with the definition of linear group of actions, we constrain an action and its immediate successor not to be separated by a control node (there should not exist a path of edges between an action and its next action such that this path contains a control node).

In step 2, inference groups are identified within each linear group of actions. This is done by traversing the actions of linear groups of actions in reverse order. Each time an inference is met, a candidate inference group is identified. The preceding inferences are recursively incorporated into the candidate inference group (as long as property 3 is satisfied); the preceding obtains are then recursively incorporated into the candidate inference group (as long as property 3 is satisfied). A candidate inference group is defined as an inference group if it contains at least two actions.

D. The transformations

Once the inference groups have been identified, the knowledge model is mapped to the design level, using the transformations described below. The result is a set of PRR production rulesets, combined with one UML activity diagram for each task method.

At the design level, we replace dynamic and intermediate roles by their domain mapping. This is at the expense of reusability, but reusability in CommonKADS is performed primarily at the analysis level through the reuse of template knowledge models. Furthermore, domain mapping makes activity diagrams more readable. We also simplify activity diagrams at the design level by keeping only the variables on which actions are performed (<<WriteVariable>>, <<AddVariableValue>> or <<RemoveVariableValue>>).

The transformations described below focus on the non-immediate mappings between the analysis level and the design level. (For example, implicitly, a concept, defined as a UML class in the domain schema, is mapped into a class at the design level; an action <<AddVariableValue>> in an analysis activity diagram is mapped into the same action in the design activity diagram, etc…)

Transformation T1: An inference or an inference group is mapped into an action, represented internally as a production ruleset.

Transformation T2: The input and output dynamic roles of an inference or an inference group are mapped into input and output parameters (pins) of the action resulting from transformation T1.

Transformation T3: The input dynamic roles of an inference or an inference group are mapped into parameters of the production ruleset resulting from transformation T1.

Transformation T4: For each static knowledge role of an inference or an inference group, the rule type instances of the implication rule type referenced by the static knowledge role are mapped into production rules in the production ruleset resulting from transformation T1.

Transformation T4.1: Each antecedent or consequent of the rule type (i.e. of the rule type of transformation T4), is mapped into at least one rule variable in each of the production rules.

Transformation T4.2: For each antecedent of the rule type, there are at least as many navigation call expressions in each of the production rules as indicated by the cardinality between the antecedent and the rule type. These navigation call expressions may be in the filter of the rule variable (or variables) representing the antecedent, or in the rule condition of the production rule.

Transformation T4.3: For each consequent of the rule type, there are at least as many rule actions in each of the production rules as indicated by the cardinality between the consequent and the rule type. These rule actions are of the assign type; they modify the class referenced by the consequent, or one of its subclasses.

Transformation T4.4: For each consequent of the rule type, an action of the assert type is created in each of the production rules to record the fact that the production rule has modified the consequent.

Transformation T5: For each input dynamic role of an inference, the class referenced by the role is used at least once in a rule variable filter expression in the production ruleset resulting from transformation T1.

Transformation T6: For each input dynamic role of an inference group, the class referenced by the role is used at least once (1) in a rule variable filter expression in the production ruleset resulting from transformation T1 or (2) in the initialization of a variable in the production ruleset resulting from transformation T1.

Transformation T7: Each "obtain" transfer function of an inference group is mapped into a variable in the production ruleset resulting from transformation T1.

Transformation T8: Each "obtain" transfer function outside an inference group is mapped into an action.

VI. CONCLUSION

In this paper, we have presented an approach for mapping CommonKADS knowledge models into specifications based on the PRR language. We have introduced the concept of "inference group", and defined an algorithm for automatically identifying inference groups, before proceeding to mapping transformations. In order to map knowledge models at the PIM level, we had to complete the concepts of PRR with UML activity diagrams.

Due to space limitations, it was not possible to illustrate our approach with an application. However, we have successfully applied the approach (including the mapping transformations) to a company buyout example, which is an example of assessment task.

Further research will concentrate on the following issues:

- Application of the mapping transformations to other scenarios (besides the company buyout example).
- Refinement of the transformations. In particular, in the current transformations, we do not take into account the distinction between sequential or inferencing operational modes, neither the concept of rule priority in the inferencing mode.
- Formal representation of the transformations with the QVT (Query/View/Transformation) language associated with MDA.
- Mapping of CommonKADS knowledge models into other languages than PRR at the PIM level (All CommonKADS knowledge is not necessarily meant to be represented as production rules).

VII. REFERENCES

[1] M. Abdullah, I. Benest, R. Paige, and C. Kimble, "Using Unified Modeling Language for conceptual modeling of knowledge-based systems", 26th International Conference on Conceptual Modeling (ER 2007), Auckland, New Zealand, November 2007, pp. 438-453.

[2] J. Andrade, J. Ares, R. García, S. Rodríguez, and S. Suárez, "A knowledge-based system for knowledge management capability assessment model evaluation", WSEAS Transactions on Computers, vol. 9, issue 5, May 2010, pp. 506-515.

[3] C. Forgy, "Rete: a fast algorithm for the many pattern/many object pattern match problem", Artificial Intelligence, vol. 19, issue 1, 1982, pp. 17-37.

[4] A. Giurca, D. Gasevic, and K. Taveter, Handbook of Research on Emerging Rule-Based Languages and Technologies: Open Solutions and Approaches, IGI Global, 2009.

[5] S.S. Hasan and R.K. Isaac, "An integrated approach of MAS-CommonKADS, Model–View–Controller and web application optimization strategies for web-based expert system development", Expert Systems with Applications, vol. 38, issue 1, January 2011, pp. 417-428.

[6] M. Moradi, M. Badja, and B. Vallespir, "Knowledge Based Enterprise Engineering (KBEE): a modeling framework for enterprise knowledge capitalization", in B. Vallespir and T.Alix (eds), Advances in Production Management Systems: New Challenges, New Approaches, International IFIP WG 5.7 Conference, APMS 2009, Bordeaux, France, September 21-23, 2009, Revised Selected Papers, IFIP Advances in Information and Communication Technology, vol. 338, 2010, pp.433-440.

[7] D. Nabil, A. El-Korany, and A. Sharaf Eldin, "Towards a suite of quality metrics for KADS-domain knowledge", Expert Systems with Applications, vol. 35, issue 3, October 2008, pp. 654-660.

[8] Object Management Group, MDA Guide version 1.0.1., document number omg/03-06-01, http://www.omg.org/cgi-bin/doc?omg/03-06-01.pdf.

[9] Object Management Group, Object Constraint Language (OCL) specification, document number formal/2010-02-01, http://www.omg.org/spec/OCL/2.2/PDF.

[10] Object Management Group, Production Rule Representation (PRR) version 1.0 specification, document number formal/2009-12-01, http://www.omg.org/spec/PRR/1.0/PDF.

[11] Object Management Group, Unified Modeling Language (UML) version 2.2. Superstructure specification, document number formal/2009-02-02, http://www.omg.org/spec/UML/2.2/Superstructure/PDF.

[12] N. Prat, I. Comyn-Wattiau, and J. Akoka, "Combining objects with rules to represent aggregation knowledge in data warehouse and OLAP systems", Data and Knowledge Engineering, in press.

[13] G. Schreiber et al., Knowledge Engineering and Management: the CommonKADS Methodology, MIT Press, 2000.

[14] J. Sigut, J. Piñeiro, E. González, and J. Torres, "An expert system for supervised classifier design: Application to Alzheimer diagnosis", Expert Systems with Applications, vol. 32, issue 3, April 2007, pp. 927-938.

[15] D. Sutton and V. Patkar, "CommonKADS analysis and description of a knowledge based system for the assessment of breast cancer", Expert Systems with Applications, vol. 36, issue 2, March 2009, pp. 2411-2423.

[16] K. Ye, J. Yan, S. Wang, H. Wang and B. Miao, "Knowledge level modeling for systemic risk management in financial institutions", Expert Systems with Applications, vol. 38, issue 4, April 2011, pp. 3528-3538.

[17] M. zur Muehlen and M. Indulska, "Modeling languages for business processes and business rules: a representational analysis", Information Systems, vol. 35, issue 4, June 2010, pp. 379-390.

A Virtual Catalyst in the Knowledge Acquisition Process

Geraldo Boz Jr,
Milton P. Ramos
TECPAR - Paraná Institute of Technology
Artificial Intelligence Division
Curitiba, Brazil
{gbozjr, milton.ramos}@tecpar.br

Gilson Yukio Sato,
Cesar A. Tacla
UTFPR - Federal University of Technology - Paraná
Curitiba, Brazil
{sato,tacla}@utfpr.edu.br

Julio C. Nievola,
Emerson Cabrera Paraiso
PUCPR - Pontifical Catholic University of Paraná
PPGIa
Curitiba, Brazil
{nievola, paraiso}@ppgia.pucpr.br

***Abstract*—Noctua is a tool to assist the Knowledge Acquisition and Collaborative Knowledge Construction processes. Noctua contains a virtual catalyst designed to facilitate the task of eliciting and validating knowledge. The virtual catalyst queries collaborators, proposing new knowledge, seeking confirmation to the knowledge already elicited, and showing conflicting opinions. Noctua takes into account collaborators' profiles in order to ask them questions related to each one's field of knowledge or interest. In this paper we present Noctua and the first experimentation we did using the tool.**

Keywords - knowledge aquisition; virtual catalist; collaborative knowledge construction

I. INTRODUCTION

In this paper, we present a web tool called Noctua (http://projetos.dia.tecpar.br/noctua/), which can be used for Knowledge Acquisition (KA) and Collaborative Knowledge Construction (CKC). Noctua has a virtual catalyst that aim at overcoming some obstacles inherent to the KA process such as lack of expert's time availability and the difficulty in eliciting and representing knowledge. The catalyst also helps surpassing barriers innate to the CKC such as authorship registration and knowledge validation. The idea is to insert into the process a virtual member who plays the role of the catalyst, i.e. someone "whose talk, enthusiasm, or energy causes others to be more friendly, enthusiastic, or energetic" [3].

Noctua operates with conceptual knowledge, represented by Knowledge Pages [1] and procedural knowledge, represented by Production Rules [4].

The tool tracks users' actions and log their collaborations. Additionally, the tool registers users' opinions confirming or refuting something. This information is used by the virtual catalyst to choose which questions to ask and which ones not to ask to each user.

II. COLLABORATIVE KNOWLEDGE CONSTRUCTION

This section presents the fundamental concepts for the construction of the tool presented in section III.

A. The Knowledge and Knowledge Acquisition

For the purpose of this paper, knowledge may be considered as information combined with experience, context, interpretation and reflection, as defined in [5]. Milton presents in [1] a technical definition according to which "knowledge is the ability/skill/expertise to manipulate/transform/create data/information/ideas to perform skillfully/make decisions/solve problems".

Knowledge Acquisition (KA) may be defined as "the transfer and transformation of problem-solving expertise from some knowledge source to a computer program" [6]. It is both an art and a bottleneck in the construction of knowledge-based systems [4].

The construction of a glossary of terms used by experts should be one of the first steps in the design of knowledge-based systems. According to Rolston [4], knowing experts' vocabulary is a fundamental task in KA. Silva Junior et al. [7] state that "an adequate design of a cognitive system depends on the existence of a common vocabulary".

One of the most popular KA techniques is the interview, despite criticisms on its efficiency [8]. It may be structured or unstructured, depending on its level of formalism and specificity.

Creating scenarios is another KA technique, in which experts are stimulated to explain their knowledge. Milton [1] suggests the creation of scenarios that depict or envisage real situations. Scenarios with potentially inconsistent or missing information may stimulate experts to question them or to identify inconsistencies that were not recognized until then. This can be used by the knowledge engineer to make expert's knowledge explicit.

B. Collaborative Knowledge Construction

Ramalho and Tsunoda [9] state that the Information and Communication Technologies created new spaces and forms for the construction of knowledge. Systems for collaboration (or collaborative software) make use of the Internet to foster communication and information organization, providing tools that facilitate the coordination inside groups of collaborators.

However, according to some researchers, the collaboration via web has some disadvantages. Pettenati and Ranieri [10] suggest that distance collaboration has deep social problems related to trust and reputation of the participants. It requires the development of a group culture and faces difficulties related to knowledge representation and management. The difficulty of representing the group and the competence of each member as well as the lack of face-to-face contact may weaken the sense of belonging and quickly lower the motivation to cooperate.

According to Nabeth et al. [11], the participation in communities of CKC is not, in fact, spontaneous, but driven by factors such as direct awards, gains from enhanced reputation or personal influence power, personal satisfaction with perception of their effectiveness and reciprocity. Participation requires a climate of trust, a sense of community, and a perception of recognition. These authors suggest a maximum effort to make visible the actions of each collaborator and their perceived value in the development of the collaborative process. This is what they call Social Translucence.

C. Collaborative Tool for CKC

According to Ackerman et al. [12], a system for CKC must have 3 basic characteristics: a tool for recording interviews, a discussion forum and a local memory. Noy et al. [13] presented a users' reviews that compare the characteristics of several collaborative tools. They highlighted as desirable:

- An easy to use web interface;
- The capacity to show the reliability of each knowledge piece and each collaborator; and
- The capacity to allow disagreements and discussions about the knowledge under construction.

Lomas et al. [14] considered as important characteristics the possibility of synchronous and asynchronous collaboration as well as information about the authorship. Other important features cited by them are: adequate communication tools, easy-to-understand interface, image sharing, collaborative construction of documents and social interaction.

Noy and colleagues presented in [15], desirable characteristics of tools for collaborative development of ontologies. We think that these features would be also welcome at any CKC tool:

- The capacity of tracking the changes undergone by the knowledge during its construction and keeping all the related comments and discussions;
- The capacity to save old versions of knowledge, with the possibility of further changing it and to discard newer versions, as well as to compare two versions;
- Automatic identification of conflicting knowledge and mechanisms to resolve conflicts;
- Users with the power of mentoring, who have the final word on possible conflicts of knowledge.

D. Negotiation and Construction of Consensus

Collaborative construction of knowledge requires that all collaborators understand the shared knowledge representation and be able to express themselves by using it, to show their agreement or disagreement with other participants and to evolve, somehow, to a consensus or a final decision [2].

A collaborative tool must have rules about how knowledge might be proposed, changed or deleted. Dieng et al. [16] describe the CO4 protocol in which, when someone proposes a change in the knowledge, if there is no disagreement, the modification is performed. On the other hand, if there is any disagreement, the modification is not made. All participants are invited to comment and submit alternative proposals. The discussion ends when the rejected proposal is removed or when the disagreement is withdrawn.

According to Herrera and Fuller [17], negotiation is a key aspect in CKC because building consensus among the participants is a condition for the evolution of the knowledge within its life cycle. Those authors constructed a negotiation model that encompasses some predetermined actions such as: request for explanation, suggestions for modification, and adoption of a position by vote.

III. THE TOOL: NOCTUA

This section presents the general characteristics of our tool called Noctua, highlighting the main characteristics of the virtual catalyst.

A. The Noctua's Projects

Noctua allows every user to create KA/CKC projects. Every project starts out empty. This allows collaborators inside each project to work in their own area of knowledge and to express themselves in their own specific way, focus and desired depth.

B. Classification and Representation of Knowledge

Noctua allows the construction of two kinds of knowledge, according to a classification presented by Milton [1]:

- Conceptual Knowledge: tells what something is;
- Procedural Knowledge: tells how to do something.

Noctua represents Conceptual Knowledge using Knowledge Pages (KP), which describe the knowledge by means of natural language texts and pictures.

Procedural Knowledge is represented by Production Rules (PR) in the following format:

if <list of conditions> then <list of conclusions>

Both representations were chosen because of their similarity to natural language which allows people unfamiliar to computers easily express their knowledge, as well as understand the information expressed in those formats.

C. Features of the Collaboration Tool

To overcome the difficulties presented in section II (subsection C), the Noctua allows the owner of each project to classify users as owners, tutors, and collaborators, each one with different permissions within a project.

Every collaborator may question the validity of a rule or entry. Questioned knowledge becomes invalid and must be discussed by the collaborators. If consensus is not reached, the matter may be finally resolved by a tutor.

Noctua allows all knowledge to be tagged. Collaborative tagging forms a folksonomy that reflects the collaborators' knowledge about the domain and can be helpful in building richer domain models in a consensual way [18].

Social Translucence is guaranteed by several aspects in the tool, such as: the registration and the disclosure of the authorship, chat between collaborators, and the disclosure of events such as creation and deletion of knowledge pieces.

Silva Junior et al. [7] establish a set of basic functions desirable in a groupware to support collaborative ethnography. If those functions are adapted to KA or to CKC, it is possible to see that Noctua embodies all of them:

- Creating, updating and closing KA/CKC projects;
- Recording users' profiles (and actively using them);
- Assigning users to activities (projects and roles);
- Recording notes and historical data (forums and instant messages);
- Creating documents (KPs and PRs);
- Supporting discussion and negotiation (by questioning knowledge and also in forums and instant messages);
- Supporting the awareness of the level of participation and contribution (by showing authorship statistics of each one's participation);

D. The Virtual Catalyst

The catalyst stimulates knowledge creation by asking collaborators questions and requiring their opinions (as shown further in this paper). Acting as a newcomer among experts, sometimes the catalyst asks impertinent questions, but sometimes its questions requires experts to rethink their concepts, so they may not only make their knowledge explicit but also wider.

The catalyst presents a screen containing a presumed knowledge (rule or some information within an entry). This knowledge, however, can be an exact copy of what is in the database or be a knowledge piece amended by the removal or the insertion of information related to other knowledge piece. In fact, when applied to rules, this technique creates scenarios and it can be used as described on section II-A.

Some of the possible forms of stimulation based on information in KPs are:

- Could you write something about <concept X>?
- What the text below is related to?
- In the entry on <concept X> it is written: <phrase A>. Do you agree with that?
- Which of the following statements applies to <concept X>?
- Is there a feature common to <concept X> and <concept Y>?

Concerning the rules, the virtual catalyst may act in several ways:

- Take a rule as it is (but not mentioning this to the user);
- Take two rules, mix some of their conditions and conclusions into a new tentative rule;
- Take a rule and delete some of its conditions;
- Take a rule and insert a condition or a conclusion from other rule.

The basic instigating question in these cases is: "See the rule below. Do you confirm it?"

Alternatively, a tentative set of conditions may also be used to ask the collaborator to think forward, not only confirming or refuting what is written, but trying to find a new valid conclusion. The question, in this case, is:

- Is there a possible conclusion if these conditions are fulfilled? (Then what?)

Asking for a new conclusion can also be done by using an established conclusion as a condition in a possible new rule. This leads to the creation of new layers of rules, i.e., rules whose conditions are conclusions from other rules.

IV. EXPERIMENTING WITH THE TOOL

Noctua was experimented by a group of ten people (faculty members from the Pontifical Catholic University of Paraná (PUCPR) and the Federal University of Technology - Paraná (UTFPR), and engineers from the Paraná Institute of Technology (TECPAR), all of them experts in computer programming. The experiment started with a face-to-face meeting in which the tool was presented to the participants. They defined the subject of the experiment: food and wine pairing. The idea was to describe wines and dishes in the Hyper Glossary and create rules for combining them properly.

The results are shown in Table I. The quantity of spontaneous knowledge eliciting (Spo) is compared to the quantity of those elicited by means of questions made by the virtual catalyst (Noc):

TABLE I. NOCTUA EXPERIMENT RESULTS

	Quantity	Noctua %	Spontaneous %
Entries	12	17	83
Questions on entries	15	---	---
Opinions on entries	12	17	83
Rules	38	16	84
Questions on rules	96	---	---
Opinions on rules	48	29	71

It is possible to observe that the participants used the catalyst only a few times. They allowed Noctua to ask them 111 questions (an average of 11.1 questions per person) and yet about 1 in every 6 rules and entries was obtained after the

questions asked by the catalyst. This indicates that the catalyst is a potentially useful tool.

By analyzing the rules produced by the group, we observed that the process had a critical flaw: it was not carried out a preliminary step in which the input variables could be defined. Each participant created his own set of input variables instead of collaborating to create one only set. Some of them considered dishes as input variables and wines as conclusions, others made the opposite.

We also observed that 90% of the rules and 67% of the entries have one only author. This shows that each participant did not collaborate much to add information to what was started by another. They agreed, after the experiment, that this may have happened due to none of them being an expert in wines, so they did not feel comfortable to change each other's contributions.

A new experiment in which people will build knowledge about their own expertise is underway with a group of nursing students and teachers.

V. CONCLUSION

This paper presented a tool for knowledge acquisition and collaborative knowledge construction that uses KPs and PRs to represent knowledge. The tool uses the construction of glossaries, virtual interviews (by means of instant messages) and creation of scenarios as knowledge acquisition techniques. The proposed interface does not require users to have any computer programming skills. Terms inside an entry are automatically hyperlinked to other entries so the set of KPs constitutes a hyper-glossary. Logical interconnections between PRs are recognized and displayed as hyperlinks. The rule base and the hyper-glossary form an integrated knowledge base. Noctua also offers forums and instant messaging tools in order to help solving conflicts of opinion and to make collaboration more effective.

The very innovation, however, is the insertion of a virtual catalyst element in the KA/CKC process. Regarding the PRs, the catalyst proposes new rules and changes in the existing rules by means of exploring new combinations of already elicited rule parts. As for the KPs, the catalyst acts in many ways such as the request for images or definitions and the attempt to associate concepts using mutual references or common characteristics. In both cases (rules and entries), the catalyst also helps the knowledge validation by asking each collaborator his opinion about what was posted by other collaborators.

New experiments will be conducted only with groups of specialists focused on a specific aspect of their own area of expertise. Knowledge Acquisition process will include an initial step for the definition of input variables. Noctua will be helpful in this phase too, allowing collaborative construction and validation of these variables.

Future work will allow users to set entry classes, each one containing certain mandatory items. By doing so, the entries in the Hyper Glossary will be better characterized as Knowledge Pages. We also intend to make available other forms of knowledge representation such as graphs showing the interconnections between entries and rules as well as timelines to show how the knowledge evolved along the collaboration process. Additionally, Noctua will be integrated with an inference engine so the rules can be tested on-line.

REFERENCES

[1] Milton, N.R. “Knowledge Acquisition in Practice”, Springer-Verlag London Limited, 2007.J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

[2] Suthers, D.D. "Collaborative Knowledge Construction through Shared Representations", in Proceedings of the 38th Annual Hawaii International Conference on System Sciences (IEEE), pp. 5a-5a, 2005.

[3] Dictionary.com, LLC. Copyright © 2010, "Dictionary.com", [Online]. Available: http://www.dictionary.com [Accessed Sep.18, 2010]

[4] Rolston, D.W. “Principles of Artificial Intelligence and Expert Systems Development”, McGraw-Hill Book Co, 1988.

[5] Davenport , T. H.; DeLong , D. W. and Beers, M. C. “Successful Knowledge Management Projects”, Sloan Management Review, 1998.

[6] Byrd , T. A.; Cossick , K. L. and Zmud, R. W. “A Synthesis of Research on Requirements Analysis and Knowledge Acquisition Techniques”, MIS Quarterly, 1992.

[7] Silva Junior, L.C.L.; Borges, M.R.S.; Carvalho, P.V.R. “Collaborative Ethnography: An Approach to the Elicitation of Cognitive Requirements of Teams”, Proceedings of The 9th International Conference on Computer Supported Work in Design (CSCW-D), 2009.

[8] McGraw, K. and Harbison-Briggs K. "Knowledge Acquisition: Principles and Guidelines". Prentice-Hall, Inc. Upper Saddle River, NJ, USA, 1989.

[9] Ramalho, L. and Tsunoda D.F. "A Construção Colaborativa do Conhecimento a partir de Ferramentas Wiki", nos Anais do VIII ENANCIB – Encontro Nacional de Pesquisa em Ciência da Informação, 2007.

[10] Pettenati, M.C. and Ranieri, M. "Informal learning theories and tools to support knowledge management in distributed CoPs", in Proceedings of the 1st International Workshop on Building Technology Enhanced Learning solutions for Communities of Practice, Greece, pp. 345-355, 2006.

[11] Nabeth, T.; Roda, C.; Angehrn, A. and Mittal, P. "Using artificial agents to stimulate participation in virtual communities", ADIS International Conference CELDA (Cognition and Exploratory Learning in Digital Age), pp. 2 5, 2005.

[12] Ackerman, M; Pipek, V. and Wulf, V. “Sharing Expertise - Beyond Knowledge Management”, MIT Press, 2003.

[13] Noy, N.F.; Chugh, A. and Alani, H. “The CKC Challenge: Exploring Tools for Collaborative Knowledge Construction”, IEEE Intelligent Systems, vol. 23, pp. 64-68, 2008.

[14] Lomas, B.C.; Burke, M. and Page, C.L. “Collaboration Tools”, Educause Learning Initiatives, 2008.

[15] Noy, N. F.; Chugh, A.; Liu, W. and Musen, M.A. “A framework for ontology evolution in collaborative environments”, Proceeding of The 5th International Semantic Web Conference (ISWC), 2006.

[16] Dieng, R.; Corby, O.; Giboin, A.; Golebiowska, J.; Matta, N. and Ribière, M. “Méthodes et outils pour la Gestion des Connaissances”, Dunod, 2000.

[17] Herrera, O. and Fuller, D.A. “Shared Knowledge: the Result of Negotiation in Non- Hierarchical Environments”, Proceedings of the CRIWG, pp 255-262, 2005.

[18] Tacla, C. A.; Freddo, A. R.; Paraiso, E. C.; Ramos,M. P. and Sato, G. Y. “Supporting Small Teams in Cooperatively Building Application Domain Models”, Expert Systems with Applications on ScienceDirect, Volume 38, Issue 2, pp. 1160-1170, February 2011.

A Real-Time Reliability Model for Ontology-Based Dynamic Web Service Composition

Harmeet Chawla and Haiping Xu
Computer and Information Science Department
University of Massachusetts Dartmouth
North Dartmouth, MA 02747, USA
{hchawla, hxu}@umassd.edu

MengChu Zhou
Department of Electrical and Computer Engineering
New Jersey Institute of Technology
Newark, NJ 07102, USA
zhou@njit.edu

***Abstract*—Ontology-based web service composition allows for integration of available web services in real-time to meet desired objectives. In order to evaluate the quality of composite web services at runtime, there is a pressing need to define a feasible real-time web service reliability model. In this paper, we present such a model. We first introduce a dynamic process model that supports the evaluation of web service reliability. Then we provide a hybrid reliability model for atomic web services by considering both software and hardware aspects of the services. In order to calculate efficiently the reliability of ontology-based dynamic composite web services, we present a recursive algorithm that evaluates the reliability of various service composition constructs in real-time. Finally, we use a case study to show how to compute and monitor the reliability of composite web services in real-time, and how our approach supports reliable ontology-based dynamic web service composition.**

Keywords-web service composition; ontology; reliability model; quality of service (QoS); dynamic process model; real-time.

I. INTRODUCTION

Web services are self-contained software components that can be published, discovered and invoked over the Internet. However, in many cases, a standalone web service is not sufficient to provide the needed functionality for certain user requirements [1]. This leads to the idea of composing different web services in order to meet such requirements. The process of web service composition can be either *static* or *dynamic*. In the former, the services are pre-determined during the design phase; while in the latter, only the service template can be initially defined, but the available web services associated with each constitutive component defined in the template must be determined at runtime. In order to discover, invoke, compose and monitor web services with a high degree of automation, we can use the semantic and ontological techniques [2]. In this paper, we adopt the semantic markup language for web services (OWL-S), which is an ontology language, to formally specify the semantics of web services. Such specification makes the definitions of web services machine-understandable. Decisions on adoption of a web service for service composition require matching not only the functional properties of the service, but also the nonfunctional properties such as service reliability. The functional and non-functional properties of a web service can be formally specified using an OWL-S profile. An OWL-S profile can be published and stored in an ontology-based UDDI (Universal Description, Discovery and Integration) such as an OWL-S/UDDI service registry so that when a service client searches it, services with matching profiles can be discovered and the corresponding grounding information can be retrieved. Since there may be more than one matched profiles published in an OWL-S/UDDI, a service client has to select one of them according to certain criteria such as the service reliability. This requires the calculation of service reliability in real-time. To achieve this, we first provide a hybrid reliability model for atomic web services, which considers both software and hardware aspects of the services. Then, to calculate the reliability of composite web services, we design an efficient recursive algorithm that evaluates various service composition constructs in real-time. By employing a real-time reliability model for service composition, our approach not only supports the selection of desirable web services for dynamic web service composition, but also provides an effective way to monitor the reliability of both atomic and composite web services in real-time.

Service reliability has been an important measure of the quality of web services for service composition. There are many different kinds of existing software reliability models for web services. Li *et al.* developed a user-oriented software reliability model for evaluating the reliability of web services [3]. Their approach can be used to evaluate the reliability of atomic web services based on an extended UDDI model, and to predict the overall reliability of a composite web service using a Business Process Execution Language (BPEL)-specified structure. Tsai *et al.* proposed a software reliability model that could dynamically evaluate the reliability of atomic and composite web services [4]. The model first calculates the reliability of atomic services by using group testing and majority voting, and then the overall reliability of a composite service by using an architecture-based model. The above approaches consider only the software aspect of web services and assume the reliabilities of the machines that host the web services are near perfection. Furthermore, they are typically based on BPEL-like architectures that require static binding of available web services with the service components defined in a process model at design time. Thus, they do not support dynamic web service composition. In contrast, our approach considers both software and hardware aspects of service reliability. Furthermore, since our approach supports ontology-based web service composition, it provides a feasible way for maintaining reliable composite web services at runtime.

There are also many previous efforts on ontology-based web service composition and formal modeling of dynamic service composition. Ma *et al.* introduced an ontology-based model for web service composition, called OMWSC [5]. They presented a goal-driven and ontology-based architecture that could support automatic composition of web services. Xiong *et al.* presented a service functional configuration net based on Petri nets for automatic service composition [6]. They described the configuration specification for component services through the structure of disassembly Petri nets, and obtained the optimal one using linear programming. Tan *et al.* introduced a formal method to derive possible web service composition candidates based on a service portfolio [7]. They first generated a service net (SN) containing all needed operations, and then used Petri net decomposition techniques to derive a subnet of SN that meets the business requirements. Although the above approaches support dynamic web service composition, most of them consider only the functional requirements for service composition, and none of them attempted to use service reliability as a major criterion for service selection. Different from the above approaches, we developed a real-time service reliability model for ontology-based web service composition. Thus, our approach supports dynamic composition of reliable web services at runtime.

II. ONTOLOGY-BASED WEB SERVICE COMPOSITION

A. Ontology-Based Web Service Composition

Specifying service related information semantically is the key to effective dynamic service discovery and service composition. Web service description language (WSDL) can be used to describe the syntax of a web service such as its input and output parameters, as well as related information such as the service provider and the service endpoint address; however, it does not support specifying the semantics of a web service. Therefore, WSDL has its limitations in supporting the dynamic service discovery, execution, composition and interoperation of web services. On the other hand, ontology-based techniques can not only be used to describe the syntax but also the semantics of web services. In ontology-based semantic modeling, the terms used in the concerned domain can be precisely defined, thus services can be matched based on their semantics rather than their syntax or keywords. In our approach, the process model is defined as a template, called a *Process Model Template* (PMT). In order to instantiate a PMT into an *Instantiated Process Model* (IPM), we need to search for available web services for its constitutive service components. For this purpose, each service component, which is also called a *simple component*, is associated with an OWL-S profile template. An OWL-S profile template is essentially an incomplete OWL-S profile with semantically defined input, output, preconditions and effects so that it can be matched with existing OWL-S profiles published in an OWL-S/UDDI. Based on the matched OWL-S profiles, a simple component can be bound to either an atomic or a composite web service. When a matched web service is a composite one, its process model must also be defined using a PMT, which can be instantiated further in the same manner. This procedure repeats until all matched web services become atomic. In this case, the process of instantiating the PMT is completed.

Since a profile template can be matched with more than one published OWL-S profiles, the most desirable one must be selected for execution. The criteria for service selection can be based on multiple features, such as provider, reliability, and price; however, for simplicity, in this paper, we only consider the reliability as the sole criterion for service selection. Since service reliability is a dynamic property, it requires the system be able to calculate it in real-time. This serves two purposes, namely the selection of web services for dynamic service composition and the real-time monitoring of web service reliabilities. With the monitoring function, a reliable service-oriented system can be maintained continuously – when the reliability of some web services drop to an unacceptable level, they can be replaced by other reliable ones at runtime.

B. Dynamic Process Model

A PMT is defined as a dynamic process model that consists of structural components, such as a *sequence* component and a *parallel* one. Each structural component contains simple components and possibly other structural ones. At runtime, a simple one can be bound to either an atomic web service or a composite one. The PMT is formally defined using Backus-Naur form (BNF) as in Fig. 1. As shown in the definition of PMT, the reliability requirements for a process model are described by using two parameters, namely `<desired reliability>` and `<marginal reliability>`. The former defines the desired reliability of an instantiated process model (i.e., a composite web service). A composite web service with at least the desired reliability is considered to be reliable. On the other hand, a composite web service with at least the `<marginal reliability>` but less than the `<desired reliability>` is considered as not reliable but is acceptable for a temporary usage. In this case, the application must try to search for more reliable ones in order to meet the reliability requirements of the composite web service. If it fails, a warning message must be sent to the user of the application. Furthermore, when the reliability of a composite web service becomes less than the marginal reliability, the composite web service becomes unacceptable and must stop its execution.

```
<PMT> ::= <pmt><desired reliability>
    <marginal reliability><process model></pmt>
<desired reliability> ::= <float>
<marginal reliability>::= <float>
<process model> ::= <process><start>
    <structural component><finish></process>
<structural component> ::= <sequence component>|
    <parallel component>|<loop component>|
    <choice component>
<sequence component> ::= <sequence><component>
    <component>{<component>}</sequence>
<parallel component> ::= <parallel>component>
    <component>{<component>}</parallel>
<loop component> ::= <loop><condition><component>
    </loop>
<condition> ::= <Boolean expression>
<choice component> ::= <choice><component>
    <component>{<component>}</choice>
<component> ::= <simple component>|
    <structural component>
<simple component> ::= <simple><component id>
    <owl-s profile template></simple>
<component id> ::= <string>
```

Figure 1. Definiton of PMT in BNF

A structural component can be one of the four major composition constructs, namely *sequence*, *parallel*, *loop*, and *choice* [4]. We now give a description of the major constructs defined in a PMT as follows.

Sequence In a sequence structural component, the constitutive components are executed in series. Fig. 2(a) shows an example with two simple components *A* and *B*, defined in a PMT. The directed arrow between *A* and *B* indicates the order of execution. When one of the constitutive components in a sequence construct is not functioning, the entire sequence structural component is not.

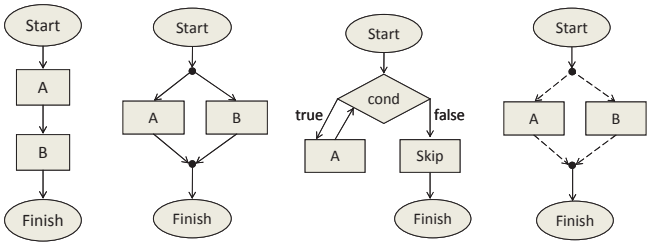


Figure 2. Examples: (a) sequence (b) parallel (c) loop (d) choice constructs

Parallel In a parallel structural component, two or more constitutive components can execute concurrently. The structural component terminates when all of its components have finished their execution. Fig. 2(b) shows a parallel structural component with two simple components *A* and *B*.

Loop A loop structural component refers to the repetitive execution of a simple or structural component. As shown in Fig. 2(c), when the condition *cond* is evaluated to be *true*, simple component *A* is executed repetitively. Only when it becomes *false*, the loop construct terminates. Note that the "skip" component is an empty component that is used to separate the loop construct from other components.

Choice In a choice structural component, only one of the constitutive components can be selected for execution. Fig. 2(d) shows an example of a simplified choice structural component with no guards (conditions). When guards are not defined, the component (*A* or *B*) to be selected for execution is determined manually by user inputs.

Fig. 3 shows an example of PMT with multiple structural components. The process model is defined as a parallel component with two sequence components defined as its constitutive ones, which can execute concurrently. The first sequence component consists of two components, namely simple component *A* and a choice structural component with two simple ones *B* and *C*. The second sequence component consists of two simple ones *D* and *E*.

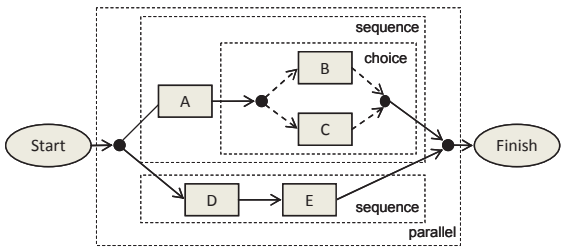


Figure 3. An example of PMT with mutiple structural components

C. Instantiation of PMT into IPM

When a PMT is instantiated into an IPM, each simple component defined in the PMT needs to be bound to either an atomic web service described by a WSDL file or a composite one specified by another PMT. Since such information must be recorded in the IPM when a matched web service is selected, we define an IPM as an extended version of a PMT with a set of placeholders for the mapping information that details how a simple component can be bound to a selected web service. At runtime, the placeholders are filled up with such detailed service information. The IPM can be formally defined using BNF as in Fig. 4.

```
<IPM> ::= <ipm><PMT><simple component mapping>
    {<simple component mapping>}</ipm>
<simple component mapping> ::= <simple mapping>
    <component id><placeholders for matched
    service></simple mapping>
<component id> ::= <string>
<placeholders for matched service> ::= <ph>
    <real-time reliability><service id>
    <service type>(<atomic service grounding
    info>|<composite service grounding info>)</ph>
<real-time reliability> ::= <float>
<service id> ::= <string>
<service type> ::= "atomic" | "composite"
<atomic service grounding info> ::= <wsdl file>
<composite service grounding info> ::= <ipm file>
```

Figure 4. Definition of IPM in BNF

As shown in the definition, the placeholders for a matched and selected web service are enclosed by the `<ph>…</ph>` tags. The item `<real-time reliability>` is the reliability of the web service that is bound to a simple component, which is calculated at runtime. The two parameters `<service id>` and `<service type>` are the identification and the type of the selected web service, respectively, where the service type can be either "atomic" or "composite." If the selected web service is an atomic one, the placeholder `<atomic service grounding info>` must be filled with the address of its WSDL file. On the other hand, if the selected one is a composite one, the placeholder `<composite service grounding info>` needs to be filled with the location of the IPM file that specifies the composite web service. The procedure for instantiating a PMT into a set of IPMs is defined in Algorithm 1. Note that it is defined recursively because a simple component can be mapped to a composite web service specified by another PMT file. In this case, that IPM file must also be instantiated by invoking the method *Instantiate-PMTintoIPM* recursively. As a result, the output of the algorithm is a set of IPM files organized in a tree-like structure. Furthermore, the instantiation process requires calculating web service reliabilities in real-time. This is because when more than one matched web services are discovered, their reliabilities need to be calculated in real-time, such that the most reliable one can be selected for execution. In order to calculate the reliability of a composite web service, an external method *CalculateReliability* (to be discussed in Section III-B) must be invoked. Note that calculating the reliability of a composite web service requires its IPM file as an input (line 17 of the algorithm), which has been generated after the recursive method call *InstantiatePMTintoIPM (ws.pmt)* in line 16.

Algorithm 1: Instantiate PMT into IPMs

Input: a *pmt* file to be converted
Output: a set of *ipm* files arranged in a tree structure.

```
1. InstantiatePMTintoIPM (File fname.pmt)
2.    copy fname.pmt to fname.ipm & create placeholders in fname.ipm
3.    create a PMT object pmt_obj from file fname.pmt
4.    foreach simple component sc in pmt_obj.process_model
5.       initialize sc.realtime_reliability to 0
6.       query OWL-S/UDDI using sc.profile_template
7.       foreach matched web service ws
8.          if (ws is atomic)
9.             extract reliability parameters from OWL-S profile
10.            calculate ws.reliability for atomic web service ws
11.            if (ws.reliability > sc.realtime_reliability)
12                sc.realtime_reliability = ws.reliability
13.               sc.service_id = ws.id
14.               sc.service_type = "atomic"
15.         else if (ws is composite)
16.            InstantiatePMTintoIPM (ws.pmt)   // create ws.ipm file
17.            ws.reliability = CalculateReliability (ws.ipm, true, null)
18.            if (ws.reliability > sc.realtime_reliability)
19.               sc.realtime_reliability = ws.reliability
20.               sc.service_id = ws.id
21.               sc.service_type = "composite"
22.      if (sc.service_type == "atomic")
23.         extract wsdl address from owl-s profile of sc.id
24.         set sc.service_type = "atomic"
25.         set sc.wsdl_file to the wsdl file of sc.service_id
26.      else
27.         set sc.service_type = "composite"
28.         set sc.ipm to the ipm file of sc.service_id
29.   save the service info of sc into its placeholders in file fname.ipm
```

III. Real-Time Service Reliability Evaluation

Service reliability represents an important attribute for the QoS of a web service deployed on a certain machine. In this paper, we take into account both hardware and software aspects of service reliability as both are needed to determine the reliability of a deployed atomic web service. Then based on the reliability of the participating atomic web services, we can calculate the reliability of a composite web service according to its dynamic process model.

A. A Hybrid Reliability Model for Atomic Web Services

Software reliability growth models (SRGM) are based on the assumption that the number of faults of a software system can be continuously reduced, which results in growth of its software reliability [8]. Although SRGM has been considered as one of the most successful techniques in software reliability engineering, it is most suitable for measuring and predicting the improvement of software reliability through the testing process [9]. In this paper, we assume that there are no features added and no faults removed once an atomic web service is deployed. In this case, the failure intensity of the software component (i.e., the atomic web service) will be constant. According to [10], the number of failures of the service in a given time follows a Poisson distribution. The corresponding formula to calculate software reliability can be defined as in (1).

$$R_{software}(\tau) = \exp(-\lambda\tau) \tag{1}$$

where λ is the constant failure intensity and τ is the execution time of the web service. When a web service is computation-intensive and executed continuously, τ can be further replaced by the elapsed time since the service is deployed.

On the other hand, hardware reliability can be represented by the two-parameter Weibull distribution [11]. The corresponding formula to calculate the reliability of a hardware component can be defined as in (2).

$$R_{hardware}(t) = \exp[-(\beta t)^{\alpha}] \tag{2}$$

where α is the shape parameter ($\alpha > 0$), and β is the scale parameter ($\beta > 0$). Note that the hardware reliability decreases with time. This is because after a certain age, the product enters its wear-out phase and the failure rate starts to increase.

Bowles tried to derive a combined hardware and software reliability model for networks [12]. According to [12], "The probability of successful operation of a device is the probability that the hardware does not fail and the probability that the software does not fail." Inspired by this idea, in this paper, we calculate web service reliability by considering both the reliability of the web service and the reliability of the machine where the web service is deployed. For simplicity, we assume that a web service is initially deployed on a new machine with the perfect reliability of 1. Thus, time t can be defined as $t = t_{current} - t_0$, where $t_{current}$ is the time when the reliability is calculated and t_0 is the time when the web service is deployed. Based on (1) and (2), the hybrid reliability model for atomic web service can be defined as in (3).

$$\begin{aligned} R_{service}(t) &= R_{software}(t) * R_{hardware}(t) \\ &= \exp(-\lambda t) * \exp[-(\beta t)^{\alpha}] \end{aligned} \tag{3}$$

where $R_{software}(t)$ is the reliability function of the atomic web service and $R_{hardware}(t)$ is the reliability function of the machine that hosts the atomic web service.

The needed parameters for calculating the reliability of a deployed atomic web service can be stored in an OWL-S profile as follows.

```
<profile: Reliability>
  <FailureIntensity datatype="float"> 0.0001
  </FailureIntensity>
  <Shape datatype="float">2</Shape>
  <Scale datatype="float">0.00002</Scale>
  <Date datatype="date">07/16/2010</Date>
  <Time datatype="time">14:30</Time>
</profile: Reliability>
```

Note that in the reliability portion of an OWL-S profile, we also include the parameters of the deployment date and time of the atomic web service because they are needed for calculating the web service reliability. Since the reliability of an atomic web service is time-dependent, the above parameters must be retrieved at runtime such that the reliability of the atomic web service can be calculated in real-time.

B. Reliability Model for Composite Web Services

The overall reliability of a composite web service depends on its structure, the degree of independence between service components and the availability of its constitutive web services. In order to calculate the reliability of a composite web service, we first need to consider the reliability of the major structural components defined in Section II-B. Based on

previous work [3, 4], we now define the reliability model for each structural component as follows.

Sequence The reliability of a composite web service composed of services in sequence can be calculated as in (4).

$$R_{sequence}(t) = \prod_{i=1}^{n} R_{service\_i}(t) \quad (4)$$

where the constitutive web services *service_i* ($1 \le i \le n$) are all independent of each other (i.e., the failure of one service does not lead to the failure of the others), and $R_{service\_i}(t)$ is the reliability function of each constitutive atomic or composite web service.

Parallel The reliability of a composite web service composed of services in parallel can be calculated as in (5).

$$R_{parallel}(t) = \prod_{i=1}^{n} R_{service\_i}(t) \quad (5)$$

where the constitutive web services *service_i* ($1 \le i \le n$) are all independent of each other. Note that the failure of any constitutive service results in the failure of the composite one because the successful termination of the latter requires the successful termination of all of its constitutive web services.

Loop The reliability of a composite web service composed of web services in a loop can be calculated as in (6).

$$R_{loop}(t) = \min_{0 \le i \le n}(R_{service\_loopbody}(t + i * \Delta t)) \quad (6)$$

where $R_{service\_loopbody}(t)$ is the reliability function of the loop body that can be an atomic web service or composite one; n is the number of iterations; and Δt is the execution time of each iteration. When $n*\Delta t$ is not a large value, the reliability of the loop structural component would be approximately the same as when it was executed the first time.

Algorithm 2: Calculate composite web service reliability

Input: an *ipm* file for a composite web service
Output: the real-time reliability of the composite web service

```
1.  CalculateReliability (File fname.ipm, Boolean initialization,
2.                        StructuralComponent structcom)
3.    initialize reliability to 1
4.    if (initialization == true)                    // step 1
5.      create an IPM object ipm_obj from file fname.ipm
6.      foreach simple component sc in ipm_obj
7.        if (sc.service_type == "atomic")
8.          calculate sc.realtime_reliability
9.        else sc.realtime_reliability =             // sc is "composite"
10.           CalculateReliability(sc.ipm, true, null)
11.   strc = ipm_obj.process_model                   // step 2
12.   if (strc is squenceComponent)
13.     foreach component com in strc
14.       if (com is simpleComponent)
15.         reliability *= com.realtime_reliability
16.       else reliability *= CalculateReliability(null, false, com)
17.   else if (strc is parallelComponent)
18.     foreach component com in strc
19.       if (com is simpleComponent)
20.         reliability *= com.realtime_reliability
21.       else reliability *= CalculateReliability(null, false, com)
22.   else if (…)       // other cases: redundant component,
23.   …                 // loop component, and choice component
24.   return reliability
```

Choice The reliability of a composite web service composed of web services in choice can be calculated as in (7).

$$R_{choice}(t) = \min_{1 \le i \le n}(R_{service\_i}(t)) \quad (7)$$

Since we consider the worst-case scenario, the reliability of a choice structural component equals the minimal reliability of the constitutive web services.

The algorithm for calculating the reliability of a composite web service is defined recursively as in Algorithm 2. Algorithm 2 involves two steps when calculating the reliability of a composite web service. In its first step, the reliabilities of all simple components are calculated. In a case when a simple component is bound to a composite web service, the method *CalculateReliability* must be invoked recursively with the parameter *initialization* being *true*. In the second step, the reliabilities of the structural components are calculated. Similarly, when a structural component contains another structural component as its constitutive component, the method *CalculateReliability* is also invoked recursively, but this time, the parameter *initialization* is set to *false* indicating that the algorithm is now processing its second step.

IV. CASE STUDY

To demonstrate the effectiveness of our approach, we utilize a case study of financial services, which involves investment in mutual funds and stocks. We define a process model for financial investment as a composite web service with a choice structure. As shown in Fig. 5, the choice is between buying mutual funds or stocks. The upper structural component represents mutual fund investment wherein the investor has a choice of investing in three types of mutual funds. They are *equity* that involves high risk, high gain funds, *debt* that represents low risk low gain funds, and *balanced fund* that gives almost steady gain with medium risk. In order to use the financial services, the investor needs to provide information about the type of fund, investment amount and personal information. For example, if the user wants to invest in equity funds, the *Equity* web service is selected and invoked, which provides a list of equity mutual funds along with their returns. Then the *SelectMutualFund* service is invoked to automatically choose the service with best returns for the user. Finally, the *BuyMutualFund* service will be invoked to buy the selected mutual funds. Similarly, the web services *Debt* and *Balanced* can be invoked to provide a list of debt funds and a list of balanced funds with returns, respectively. On the other hand, if buying stock is chosen, the process model has the *BuyStocks* service, where the input to this service is the stock to be bought, the number of stocks, and the rate at which the user wants to buy. When the specified stock is available at the preferred rate, the *BuyStocks* service is automatically invoked to buy the specified number of stocks.

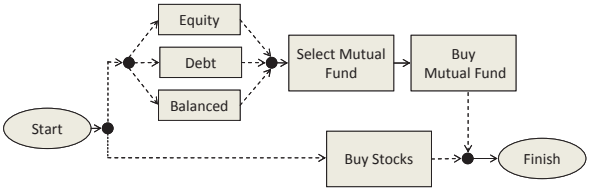


Figure 5. A process model for financial investment

In order to ensure the desired reliability of the financial services for investment, we develop a prototype service reliability monitoring tool. Fig. 6 shows the interface for monitoring the reliability of the financial investment service in real-time. Under the *ProcessModel* tab, it presents the predefined process model (as shown in Fig. 5) at the left hand side, and displays the real-time reliability of the composite web service at the right hand side. The interface also shows the status of the composite web service, which could be *normal*, *warning* or *unacceptable*. A *normal* status indicates that the reliability of the composite web service equals to or more than the desired reliability; a *warning* status indicates that the reliability is below the desired reliability but no lower than the marginal reliability. When the reliability falls below the marginal reliability, the status becomes *unacceptable*. Note that in this example, the *desired* and *marginal* reliabilities are set to 0.85 and 0.75, respectively. They can be easily re-configured by clicking on the *Configuration* tab. In addition, the interface also displays the current date and time as well as execution information of the composite web service. At the bottom part of the interface, all services bound to the simple components are listed. If the service is atomic (e.g., *EquityService*), the address of its WSDL and its real-time reliability are displayed (e.g., *http://192.168.1.112:8080/equity/EquityService?wsdl* and 0.92751); otherwise, if the service is composite (e.g., *BuyStocks1*), the address of the corresponding IPM file and its real-time reliability are displayed (e.g., *C://Files/BuyStocks1.owl* and 0.84917). The detailed information about a composite web service (e.g., *BuyStocks1*) can be retrieved by clicking on the *CompositeComponents* menu at the top of the interface. Note that when the real-time reliability of a composite service falls below the desired reliability, some of its constitutive web services with low reliabilities must be replaced by others in order to improve its QoS.

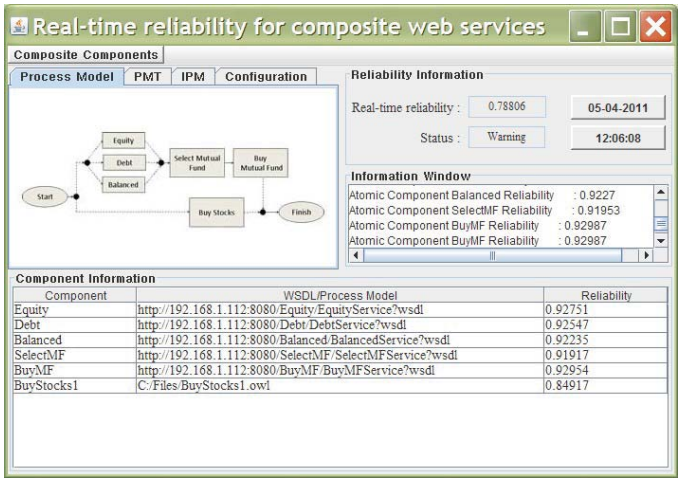


Figure 6. Interface for monitoring real-time web service reliability

V. Conclusions and Future Work

In this paper, we define a dynamic process model that consists of various constructs for web service composition. The dynamic process model is initially defined as a process model template, called PMT, where the constitutive components are not bound to any specific web services. At runtime, the PMT is instantiated into a set of instantiated process models or IPMs. During the instantiation process, web services that are matched with the simple components in a PMT are discovered and selected based on their real-time reliability values. In order to calculate the reliabilities of composite web services, we first present a hybrid reliability model for atomic web services. Then we define a real-time reliability model for a composite web service that aggregates the reliabilities of its constitutive components according to the definition of its dynamic process model. Our approach not only supports service selection for dynamic web service composition, but also supports maintaining a reliable composite web service at runtime by monitoring the service reliabilities in real-time. For future work, we will study existing software reliability models and propose the most suitable ones for atomic web services by considering additional factors such as software aging due to performance degradation or a sudden software crash. We will demonstrate how to automatically switch a web service to a reliable one when its reliability becomes unacceptable. By utilizing artificial intelligence techniques, we may further improve the service reliability model for automatic adjustment of its parameters at runtime. In addition, integrating the proposed reliability index into some existing approaches [5, 6] can also be considered as a worthy future direction.

References

[1] J. Rao and X. Su, "A survey of automated web service composition methods," in *Proc. of the First Int. Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004)*, San Diego, CA, USA, pp. 43-54, Jul. 2004.

[2] D. Martin, M. Paolucci, S. McIlraith, M. Burstein, D. McDermott, D. McGuinness, *et al.*, "Bringing semantics to web services: the OWL-S approach," in *Proc. of the First Int. Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004)*, San Diego, CA, USA, pp. 26-42, Jul. 2004.

[3] B. Li, Z. Su, Y. Zhou, and X. Gong, "A user-oriented web service reliability model," in *Proc. of the IEEE Int. Conf. on Systems, Man and Cybernetics*, Singapore, pp. 3612-3617, Oct. 2008.

[4] W. T. Tsai, D. Zhang, Y. Chen, H. Huang, R. Paul, and N. Liao, "A software reliability model for web services," in *Proc. of the 8th IASTED Int. Conf. on Software Engineering and Applications (SEA 2004)*, Cambridge, MA, pp. 144-149, Nov. 2004,

[5] J. Ma, Y. Zhang, and M. Li, "OMWSC - an ontology-based model for web services composition," in *Proc. of the 5th Int. Conf. on Quality Software (QSIC 2005)*, Melbourne, Australia, pp. 464-469, Sept. 2005.

[6] P. C. Xiong, Y. Fan, and M. C. Zhou, "Web service configuration under multiple quality-of-service attributes," *IEEE Trans. on Automation Science and Engineering*, vol. 6, no. 2, pp. 311-321, Apr. 2009.

[7] W. Tan, Y. Fan, M. C. Zhou, and Z. Tian, "Data-driven service composition in building SOA solutions: a Petri net approach," *IEEE Trans. on Automation Science and Engineering*, vol. 7, no. 3, pp. 686-694, Jul. 2010.

[8] M. R. Lyu, "Software reliability engineering: a roadmap," in *Proc. of the Workshop on Future of Software Engineering (FOSE'07)*, Int. Conf. on Software Engineering, Washington, DC, pp.153-170, 2007.

[9] H. Pham, *System Software Reliability*, Springer-Verlag, London, pp. 152-177, 2006.

[10] J. Musa, *Software Reliability Engineering*, McGraw-Hill, New York, pp. 310-311, 1999.

[11] A. Hoyland and M. Rausand, *System Reliability Theory: Models and Statistical Models*, John Wiley & Sons, New York, pp. 37-40, 1994.

[12] J. B. Bowles, "A model for assessing computer network reliability," in *Proc. of the IEEE Conf. on Energy and Information Technologies in the Southeast (SoutheastCon)*, Columbia, SC, USA, pp. 603-608, Apr. 1989.

Learning action models with indeterminate effects

Jie Gao
School of Information Science and Technology
Sun Yat-Sen University
Guangzhou , P.R.China

Hankz Hankui Zhuo
School of Information Science and Technology
Sun Yat-Sen University
Guangzhou , P.R.China

Dao-jun Han
School of Information Science and Technology
Sun Yat-Sen University
Guangzhou , P.R.China

Lei Li
School of Information Science and Technology
Sun Yat-Sen University
Guangzhou , P.R.China

Abstract—AI planning requires action models described by a formal action plan description language as input. However, building action models is a time-consuming task. Previously, some algorithms are based on the assumption that knowledge of states in the middle of observed plans could be complete and effects of actions could be deterministic. However, the assumption is not necessarily established in practice. In this paper, we assume knowledge of states in the middle of observed plans could be incomplete, and effects of actions are uncertain. We develop a novel algorithm to learn action models with conditional effects and probabilistic effects automatically, and evaluate the algorithm in two domains.

Keywords-Action model learning; Machine learning; Knowledge engineering; Automated planning

I. INTRODUCTION

Currently, automated planning is a hot research topic in artificial intelligence and increasingly applied in practice. The intelligent planning systems need to be provided with action models as input, however it's extremely difficult and time-consuming to build action models manually, even for those experts in relative fields. Therefore, it will save lots of manpower cost if action models can be obtained automatically from existing planning traces.

Recently, researchers have proposed some algorithms to learn action models, which assume that action models have determinate effect. However, effects are normally indeterminate in practice. For example, in slippery-gripper domain, whenever the gripper is dry or wet, there are two possible effects which are holding block or not holding block. In this action model, when gripper is dry, the possibility of holding block is 0.95, and that of not holding block is 0.05; when gripper is wet, the possibility of holding block and not holding block are both 0.5. In traditional planning research, we often assume that action models with conditional effects and probabilistic effects could be built manually by experience, but it's actually very difficult even for experts.

For solving this problem, we propose a new algorithm to learn action models with conditional effects and probabilistic effects automatically, which is called AMLCP (Action Model Learning with Conditional effects and Probabilistic Effects). The algorithm can be applied to learn action model with conditional effects and probabilistic effects from the given planning traces with incomplete state information. The obtained action model, which is shown to be approximate to the correct action model, can be provided to domain experts, and applied in the automated planning system for saving the manpower cost greatly.

In conclusion, compared to the previous action model learning algorithms, AMLCP made the following contributions: (1) obtained action models could have conditional effects and probabilistic effects. In practice, the effects of action are usually uncertain and conditional, with multiple possibilities. AMLCP can obtain action models with conditional effects and probabilistic effects; (2) state information of the planning traces could be incomplete. It is quite difficult to obtain complete state information in reality. AMLCP can be applied with incomplete state information.

The rest of the paper is organized as follows. In section 2, we introduce the related work. In section 3, we present the steps of the algorithm AMLCP in detail. In section 4, we construct experiments in three planning domains to estimate the error rates of learned action models by AMLCP. In section 5, we summarize the paper and discuss our future works.

II. RELATED WORK

Recently, researchers have proposed some algorithms to learn action models. According to whether state information is complete, these algorithms could be divided into two parts. Some algorithms are, to learn action models from plane traces with complete state information[1-8]. Gil et al. [1] build EXPO system, bootstraped by an incomplete STRIPS-like domain description with the rest being filled in through experience. Oates et al.[2] use a general

classification system to learn preconditions and effects of actions. Schmill et al.[3] learn action models by approximate computation in relative domains. Wang et al.[4] propose an approach to learn action model automatically by observing planning traces and refine the operators through practice in a learning-by-doing paradigm. Pasula et al. [5,6] present how to learn stochastic action models without conditional effects.

The other algorithms, are to learn action models with incomplete state information[7-13]. Yang et al. [7] build action model learning system-ARMS by using SAT, which can handle the case with incomplete state information, and acquire STRIPS action models. Amir et al. propose the algorithm SLAF[8-11] to learn action models with incomplete state information, which resembles version spaces and logical filtering and identifies all the models that are consistent sequence of executed actions. Lai et al. [12] construct AMLS system to learn action models from planning traces with incomplete state information, under the framework of genetic algorithm. For acquiring action models with better expression ability, Zhuo et al. [13] build a novel learning algorithm LAMP, which applies Markov logic networks to obtain action models with quantifiers and logical implications.

III. Action Model Learning with Conditional Effects and Probabilistic Effects

A. Definition of action model learning with conditional effects and probabilistic effects

A STRIPS-like planning problem with conditional effects and probabilistic effects can be defined as a four-tuple $<S,s_0,s_g,O>$, where S represents a set of states, and each state is a set of propositions. s_0 represents the initial state and s_g represents the goal state which is the final state following with a series of states transition, starting with initial state. O represents a set of action models with conditional effects and probabilistic effects, composed of a three-tuple $<a, PRE, CPEFF>$, in which a represents an action schema with action name and parameters, for example (*pickup ?b*); *PRE* represents preconditions. *CPEFF* represents conditional effects and probabilistic effects, formally expressed as $<(p_{i1},c_1,e_{i1})...(p_{ij},c_1,e_{ij})\ldots(p_{in},c_1,e_{in})>$, in which c_1 represents the i^{th} condition composed of literal and conditions c_1 $(1 \le i \le k)$ are mutually exclusive, the corresponding j^{th} effect is represented by e_{ij} with probability p_{ij}, which is the conjunction of literal. A possible action sequence is denoted as $<a_1,a_2,\ldots,a_n>$, transferring from initial state s_0 to goal state s_g. Furthermore, we call $(s_0,a_1,s_1,a_2,\ldots,s_n,a_n,s_g)$ as a planning trace, where the middle state s_i might be null, and a_i represents action schema.

Action model learning with conditional effects and probabilistic effects can be described as follows. Given planning traces set T, propositions set P as input, algorithm AMLCP outputs all the action models with conditional effects and probabilistic effects in A.

Table1 The Input Of Algorithm AMLCP

Input: Predicates *P*			
(block ?b) (gripper ?g) (gripper-dry ?g) (holding-block ?b) (block-painted ?b) (gripper-clean ?g)			
Input: Action Schemas *A*			
(pickup ?b ?g) (dry ?g) (paint ?b ?g)			
Input: Plan Traces *T*			
	Trace 1	**Trace 2**	**Trace 3**
Initial state	*(gripper G) (block B) (gripper-clean G) (gripper-dry G)*	*(gripper G) (block B) (gripper-clean G)*	*(gripper G) (block B) (gripper-clean G)*
Action 1	(paint B G)	(pickup B G)	(pickup B G)
Observation 1		*not(holding-block B)*	*(holding-block B)*
Action 2	(pickup B G)	(dry G)	(paint B G)
Observation 2			
Action 3		(pickup B G)	
Observation 3			
Action 4		(paint G)	
Goal state	*(gripper-clean G) (holding-block B) (block-painted B)*	*not(gripper-clean G) (holding-block B) (block-painted B)*	*not(gripper-clean G) (holding-block B) (block-painted B)*

Table 2 Output Of Algorithm AMLCP

Action schema	(pickup ?b ?g)
Preconditions	*(block ?b) (gripper ?g)*
Conditional effects and probabilistic effects	<(0.95 *(gripper-dry ?g)* (and *(holding-block ?b)*)), (0.05 *(gripper-dry ?g)* (and*(not(holding-block ?b))*))> <(0.5 (not *(gripper-dry ?g)*) (and *(holding-block ?b)*)), (0.5 (not *(gripper-dry ?g)*) (and*(not(holding-block ?b))*))>

Table 3 Framework Of Algorithm AMLCP

Input	1. A set of plan traces *T*; 2. A set of action schema *A*; 3. A set of predicates *P*.
Output	A set of action models with conditional effects and probabilistic effects.
Steps	1. Encode each plan trace as a set of propositions; 2. Generate candidate formulas, using *A* and *P*; 3. Apply MLNs to learn weights of all the candidate formulas; 4. Choose some of candidate formulas according to given threshold, and convert weighted candidate formulas to action models with conditional effects and probabilistic effects as output.

B. Framework of algorithm AMLCP

The motivation of our algorithm AMLCP is to transform action model learning problem into weights learning problem in MLNs, and obtain action models with conditional effects and probabilistic effects. The frameworks of algorithm AMLCP is shown in table 3.

C. Steps of algorithm AMLCP

In the following subsections, we will show a detailed description of each step of the algorithm AMLCP.

1) Step 1:encode plan traces

In the first step of algorithm AMLCP, we encode all the plan traces as a set of proposition databases *DBs* with plan traces *T* as input. Firstly, we use propositions to represent each state of plan traces.Secondly, we can consider an action as the transition of states, then action can be encoded as the conjunction of propositions.

2) Step 2: generate candidate formulas of each action

In STRIPS model, if a predicate is a negative effect of an action, then the predicate should be a precondition of the action; and a predicate can not be both positive effect and negative effect of an action. Considering the two characteristics, we describe an action model in two parts: preconditions and effects.

a) Preconditions

If predicate p is a precondition of action a, then p must be satisfied when the action a is executed, which can be described formally as:

$$\forall i,\bar{x},\bar{y},a(\bar{x},i)\rightarrow p(\bar{y},i), \quad (1)$$

where $\bar{x}$, $\bar{y}$ are parameters, and i is the state symbol.

b) Conditional effects.

If predicate p is a positive effect of action a with condition c, then p should be added to the next state after the action a when condition c is satisfied, which can be described formally as:

$$\forall i,\bar{x},\bar{y},a(\bar{x},i)\rightarrow p(\bar{y},i)\wedge p(\bar{y},i+1)\wedge c(\bar{z},i), \quad (2)$$

where $\bar{x}$, $\bar{y}$, $\bar{z}$ are parameters, and i is the state symbol. If predicate q is a negative effect of action a with condition c, then q is satisfied when a is executing and condition c is satisfied, but not satisfied after action a, which can be described formally as:

$$\forall i,\bar{x},\bar{y},a(\bar{x},i)\rightarrow q(\bar{y},i)\wedge\neg q(\bar{y},i+1)\wedge c(\bar{z},i) \quad (3)$$

Applying formulas (1)-(3), we can acquire candidate formulas of preconditions and conditional effects.

3) Step 3:Learn weights of candidate formulas

According to reference [16], Markov Logic Networks L consists of a set of pairs (F_i,ω_i), where F_i is a formula in first-order logic and ω_i is a real number. With a finite set of constants $C=\{c_1,c_2,\cdots,c_n\}$, it defines a Markov network $M_{L,C}$ as follows:

- $M_{L,C}$ contains one binary node for each possible grounding of each predicate appearing in L. The value of the node is 1, if the grounded predicate is true, and 0 otherwise.
- $M_{L,C}$ contains one feature for each possible grounding of each formula F_i in L. The value of this feature is 1 if the ground formula is true, and 0 otherwise. The weight of the feature is ω_i associated with F_i in L.

We apply Alchemy system[17] to learn weights of candidate formulas, by using weighted optimized pseudo log-likelihood. For each atomic formula, if it appears in *DBs*, then it corresponds to $x_i=1$, otherwise 0.

4) Step 4: Obtain action model with conditional effects and probabilistic effects

In the candidate formulas of preconditions, we choose those formulas with weights greater than some threshold as a set and convert it into the preconditions of action model. Similarly, we can choose some candidate formulas of conditional effects and calculate their corresponding probabilities. Finally, we can obtain action model with conditional effects and probabilistic effects.

Table 4 The Acquired Action Model

Action schema	pickup(?b ?g)
Preconditions:	*block(?b), gripper(?g)*
Probabilistic conditional effects:	*<(0.87 (gripper-dry ?g) (and (holding-block ?b))), (0.13 (gripper-dry ?g) (and(not(holding-block ?b))))> <(0.48 (not (gripper-dry ?g)) (and (holding-block ?b))), (0.52 (not (gripper-dry ?g)) (and(not(holding-block ?b))))>*

IV. EXPERIMENTS AND ANALYSIS

A. Datasets and evaluation criteria

To evaluate the algorithm ALMCP, we collected plan traces from planning domains slippery-gripper and sandcastle. The two domains have the characteristics we need to evaluate ALMCP algorithm, which means both of them have indeterministic effects. Using probabilistic planner Probabilistic-FF, we generated 20-100 planning traces from the two domains as training data.We consider the given action models as correct ones, and then use the correct action models to evaluate the error rates of learned action models.

B. Experimental results

Fig.1 and Fig.2 show performance of algorithm AMLCP with respect to the number of plan traces and thresholds used in selecting candidate formulas which are set to be 0.001,0.01,0.1,0.5, respectively.

From the results of the two experiments, the error rates are affected by the number of given plan traces. Generally, error rate decreases when the number of plan traces increase. When the number of plan traces is smaller, the error rate is higher. When the number of plan traces increases to some extent, the error rate decreases rapidly, but eventually it goes down slowly. It means that the difference between

learned action models and correct ones is obvious when information is limited, but the difference will decrease with enough information. It can be speculated that learned action models will be approximate to correct ones with enough number of plan traces.

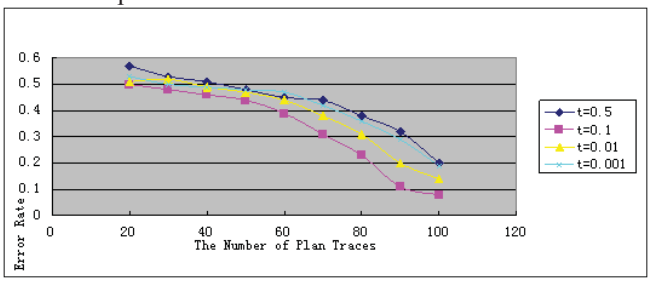


Fig.1 Error rates with respect to different number of plan traces, in slippery- gripper domain

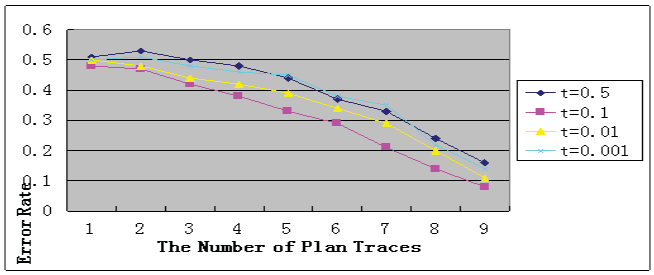


Fig.2 Error rates with respect to different number of plan traces, in soadcastle domain

V. CONCLUSION

In this paper, we proposed a novel algorithm AMLCP to learn action models with conditional effects and probabilistic effects from a set of observed plan traces where some intermediate states might be partially observable. Our AMLCP algorithm applies MLNs to learn action models automatically. The experimental results in two indeterminate planning domains show that AMLCP algorithm is quite effective. In the future, we could expand the algorithm to improve its expression ability and reduce error rates of learned action models.

ACKNOWLEDGMENT

Hankz Hankui Zhuo thanks China Postdoctoral Science Foundation funded project(Grant No.20100480806) and National Natural Science Foundation of China (61033010) for suport of this research. Lei Li thanks Macao FDCT 013/2010/A for support of this research.

REFERENCES

[1] Yolanda G. "Learning by experimentation: Incremental refinement of incomplete planning domains". In Proceeding of the Eleventh International Conference on Machine Learning(ICML 1994), 1994. 87~95.

[2] Tim O, Paul R. C. "Searching for planning operators with context-dependent and probabilistic effects". In Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI 1996) 1996. 865~868.

[3] Matthew D. S, Tim O, Paul R. C. "Learning planning operates in real-world, partially observable environments". In Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems (AIPS 2000), 2000. 246~253.

[4] Xuemei W. "Learning by observation and practice: An incremental approach for planning operator acquisition". In Proceeding of the Twelfth International Conference on Machine Learning (ICML 1995), 1995. 549~557.

[5] Hanna M.P, Luke S. Z, and Leslie P.K. "Learning and planning with probabilistic relational rules". In Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS 2004), 2004. 73~82.

[6] Hanna M.P, Luke S. Z, and Leslie P.K. "Learning symbolic models of stochastic domains". Journal of Artificial Intelligence Research, 2007. 309~352.

[7] Qiang Y, Kangheng W, and Yunfei J. "Learning action models from plan examples using weighted max-sat". Artificial Intelligence, (2-3), 2007. 107~143.

[8] Eyal A. "Learning partially observable deterministic action models". In Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI 2005), 2005. 1433~1439.

[9] Dafna S, Allen C, and Eyal A. "Learning partially observable action models: Efficient algorithm". In Proceedings of the Twenty-First National Conference on Artificial Intelligence(AAAI 2006), 2006. 920~926.

[10] Dafna S, Eyal A. "Learning partially observable action schema". In Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI 2006), 2006. 913~919.

[11] Megan N, Adam V, and Eyal A. "Reasoning about partially observed actions". In Proceedings of the Twenty-First National Conference on Artificial Intelligence(AAAI 2006), 2006. 888~893.

[12] Lai ZF, Jiang YF. "Learning action model in AI Planning Based on Genetic Algorithm". Chinese Journal of Computers, 2007,30(6):945~953.

[13] Zhuo HH, Yang Q, Derek H. H and Li L. "Learning complex action models with quantifiers and logical implications". Artificial Intelligence, 147(18). 1540~1569.

[14] Matthew R and Pedro D. "Markov logic networks". USA:Department of Computer Science and Engineering, University of Washington, 2004.

[15] Stanley K, Parag S, Matthew R, Pedro D. "The Alchemy System for Relational AI". University of Washington, Seattle, 2005.

Fraud Detection in Selection Exams Using Knowledge Engineering Tools

Marcus de Melo Braga
Post-Graduate Program in Knowledge Engineering
Universidade Federal de Santa Catarina (UFSC)
Florianópolis, Brazil
marcus@egc.ufsc.br

Mario Antonio Ribeiro Dantas
Department of Informatics and Statistics
Universidade Federal de Santa Catarina (UFSC)
Florianópolis, Brazil
mario@inf.ufsc.br

***Abstract*—This paper proposes a method for fraud detection in automated selection exams, using knowledge engineering tools for identifying groups of answers with a strong indication of fraud, based on probabilistic evidence. Founded on an analysis of the wrong answers of the various candidates, the proposed method enables identification of suspicions, and evidence of fraud attempts through finding candidates with a significant number of identical wrong answers. This method of using knowledge engineering tools can be employed in various types of selection examinations, adapting to the characteristics and features of each one and contributing to the achievement of fair exams and free of fraud attempts.**

Keywords: knowledge engineering tools; fraud detection; selection exams; multiple-choice questions.

INTRODUCTION

The need for detection of various types of fraud in several fields of application has grown in recent years, possibly due to the great developments in information and communication technologies (ICT) that provide citizens with diverse resources and technological facilities for access to databases of public and private organizations.

The growing complexity of equipment and services provided by new technologies hinders the identification of possible failure points or some susceptible to fraud attempts. In several areas of activities, there is a lack of study and research that enables the creation of methods and techniques to prevent fraud and to reduce financial losses caused by this type of crime. Knowledge Engineering (KE) provides us with tools that can assist in detecting and preventing fraud, helping us to fight this type of crime.

The application of knowledge engineering tools in fraud detection is already common in many fields of study. It occurs mainly in the following areas: mobile communications (cellular), finance, electronic commerce, healthcare, credit cards and government organizations [1], [2]. A systematic review on the subject of this research has not identified a study published in the databases Scopus (www.scopus.com) or ISI Web of Knowledge (apps.isiknowledge.com) associating the keywords "detection," "fraud," "exam," and "examination." Based on the review conducted in the two largest databases, we can state that we did not find any study of application of KE tools to detect fraud in selection examinations. Therefore, we propose a new possibility for application of these tools.

Some related studies apply others' software techniques for fraud detection. In [3], artificial immune systems are used to identify fraud attempts in online credit cards operations. Data-mining techniques can also be applied for fraud detection in e-Commerce [4], mobile communication networks [5], and in conventional telephony [6]. Other studies specifically propose statistical methods for fraud detection in several areas [7], [8].

The model presented in this paper for the application of KE tools in fraud detection in admission exams is structured into eight sections: Section II makes a brief introduction to selection examinations processes, presenting their main characteristics. Section III presents the main types of fraud that can occur in a contest aimed at giving an overview of the method to be proposed in this paper. Section IV examines the various types of questions that can be adopted in an examination. Section V presents a statistical study of the probability analysis of identical wrong answers (IWA) to a specific type of question: multiple-choice. In Section VI, we present a proposed method for fraud detection in automated selection processes. Section VII presents and discusses the results, and finally in Section VIII, we give conclusions and suggestions for future work.

II - SELECTION EXAMS

A. Definition

A selection examination process can be defined as an activity consisting of several sub-processes or steps, with the aim of selecting candidates through the application of questionnaires, tests, and other selection tools to fill a limited number of vacancies.

B. Phases of a selection exam

The phases, steps or sub-processes of a selection examination are necessary to meet statutory deadlines for publication, call for entries, preparation of tests, allocation of material and human resources, realization of exams, test

This study was supported by CAPES, the Brazilian Government entity for the training of human resources.

reading, test processing, and disclosure of results for the subsequent filling of vacancies by admission or hiring. Figure 1 shows the complete cycle of the phases of a selection process.

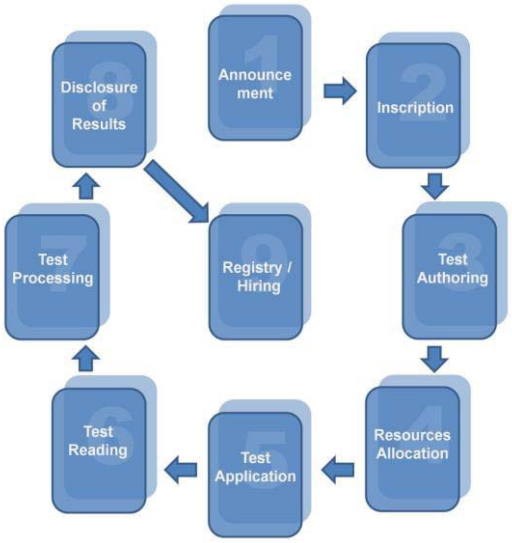


Figure 1. Phases of a selection exam.

The limitation of the vacancies is precisely the main motivation of the contest; if there were no limits to the vacancies or the number of qualified candidates for completion was not significant, selection would have been unnecessary.

The process initiates with the disclosure activities. The purpose of the announcement phase is to enroll the largest number of candidates, giving chances to all interested in attending the event. After complying with legal requirements for the announcement, the next phase opens up the inscription or registration for all applicants who may be qualified for the selection exam. Once the inscriptions are finished, in the next phase, we make the arrangements for the preparation of the tests that will be applied to candidates; this is done through enjoining experts or specialized companies for the authoring of tests and examinations. At the phase of resource allocation, the candidates are distributed by each exam place; all the materials and human resources are allocated for the next phase, such as rooms, equipment, proctors, and coordinators. In the test application stage, there are major activities related to the security aspects of a selection process, including review of all the candidates with metal detectors during their access to the exams places, planning and proctoring the application of the tests, and the electronic monitoring of all the examination locations.

Once the tests are completed, all material is collected in a safe place − usually the institution's information technology department − where the test reading phase is done with automatic readers, scanning responses of all candidates' tests for later assessment. In the test processing phase, all test answers are processed using the templates and automated processing systems for getting the results of the tests, creating the scores for the candidates, and generating their respective places (classifications). Finally, after verifying and auditing the whole process of assessment, final reports are issued for disclosure of the results toward the registry or hiring of selected candidates.

III. Fraud in Selection Exams

Currently, security is one of the most relevant points in managing selection processes, given the considerable number of technological tools that tempt people to perform activities of fraud, especially in contests where there is a great competition of candidates per position.

Security issues in exams can now be handled from the time of enrollment by consulting special database lists maintained by various Brazilian governmental institutions that contain the names of all candidates already identified in prosecution of fraud attempts, throughout the national territory. Despite having their entries approved, such candidates can be identified from the time of enrollment and allocated in special rooms to be monitored in their activities during application of the tests.

At the stage of resource allocation, some security measures can also be taken, trying to find candidates with relatives in the same contest and separating them in different rooms or places for application of the tests, avoiding their proximity.

However, it is at the phase of test application in which security issues should have greater weight because it is precisely at that point that the largest number of fraud attempts occurs. All measures already mentioned − such as the use of metal detectors and electronic monitoring of electronic transmissions − are key in this phase.

At the stage of test reading, it is also necessary to adopt preventive measures, with rigid control of access to the processing data center and precautions against possible reading processing errors.

Finally, in the phase of disclosure of results, safety measures must be adopted with respect to the risks of information leakage or premature disclosure of the results, avoiding harming the reliability of the whole process.

Our practical experience in conducting selection exams over two decades shows that occurrences of fraud attempts are fairly common on any type of competition or selection process.

The attempts of fraud range from traditional and frequent acts of trying to cheat or plagiarize during the test, copying the responses of a competitor in the same room, and even the most daring attempts, carried out by means of electronic transmission. All these attempts must be considered and scrutinized by the authorities responsible for conducting the selection process, and all the omissions in these security cases are inadmissible. Absolutely nothing can justify the absence of mechanisms for its prevention or precaution.

IV. Types of Questions

For a better understanding of the proposal for fraud detection presented in this article, it is necessary to describe the types of questions usually adopted in examination tests.

At first, the main types of questions used in tests can be grouped into three categories:

- Multiple-choice questions.
- True/false questions.
- Open questions.

Open questions significantly hinder the attempts of fraud since they require a response not coded and difficult to be transmitted − although not impossible. Open questions are the most difficult to process when adopted in contests with a large number of participants. Another disadvantage that this type of question presents is how to standardize criteria for evaluation, aiming to avoid disparities in the assessment by reducing the subjectivity and inconsistency among the different evaluators.

True/false questions inhibit the adoption of the fraud detection strategy proposed in this paper. With this type of test question, it becomes difficult to identify the "signature" or the statistical evidence of fraud, as will be shown below. For this type of question in particular, other strategies should be established for the automatic detection of fraud, taking advantage of their peculiarities.

Multiple-choice questions are still frequently adopted in selection processes for their tradition and simplicity [9]. One of their main advantages is the ease of representation on printed answer sheets that can be read by automatic readers. This procedure enables the processing of the large volumes of data collected in medium and large examinations.

The method for fraud detection proposed in this study assumes that the type of questions adopted in the contest is multiple-choice. This type of question presents an interesting feature that can be exploited for the automatic detection of fraud: We observe that when two or more candidates try to cheat on a test, the probability that their errors are equal is high; that is, the wrong answers are the same and the alternative letters answered incorrectly are also equal. Taking this observation as a basis, the method proposed in this paper makes a study of the statistical probability of simultaneous occurrence of errors on the same questions and the same alternatives to substantiate the mathematical evidence of a fraud attempt, further proposing KE tools to identify the occurrence of this problem, detecting and fighting some fraud activities in selection exams.

In the following section, we create a probability analysis of the simultaneity of the same wrong answers with the same wrong alternative letters in a test of multiple-choice questions using a total of 40 questions for a case study.

V. Probability Analysis

To illustrate the method proposed in this paper, we assume that the test being examined has 40 multiple-choice questions with 5 alternatives (ABCDE). A sample of the possible answers can be seen in Figure 2.

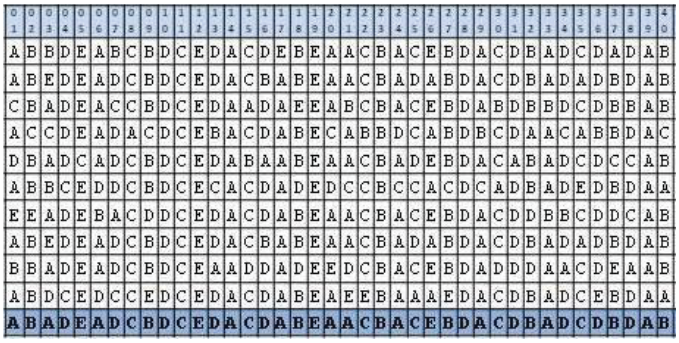


Figure 2. An example of answers of multiple-choice questions.

The first line of the figure shows the position of each of the 40 questions in the test paper. The bottom line, displayed with highlighted background, corresponds to the template of these multiple-choice questions. The 10 lines between are examples of candidates' responses.

To further facilitate visualization, we can eliminate all the correct answers of the candidates, since they are of no interest to our study. Figure 3 shows only the wrong answers, already highlighted.

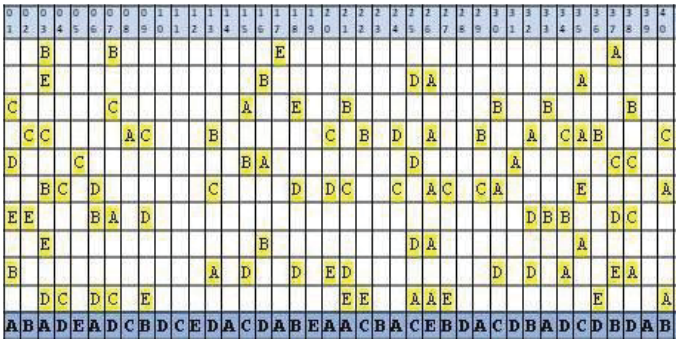


Figure 3. Only the wrong answers are displayed.

The research question for the analysis of probabilities is: What is the probability that two or more candidates err in a multiple-choice test with 40 questions (such as ABCDE) in the same question numbers and answering the same wrong alternative letters? Our hypothesis is that after a certain number of wrong answers exactly alike, the probability of occurrence is close to zero, providing formal evidence that there is a strong suspicion of a fraud attempt.

This probabilistic evidence substantiates the fraud detection method proposed in this study, applying KE tools for identifying high similarities in the wrong answers of the candidates in a selection exam that meets these characteristics and type of question.

C. *Calculating the probability*

For didactic purposes, it is necessary to define the concept of IWA. In this study, IWA are those special occurrences when two or more candidates miss the same questions, answering the same wrong alternative letters, resulting in a remarkable coincidence. One can intuitively predict that this coincidence has a significantly low probability of occurrence (i.e., it is almost impossible), and this fact sets up a strong indication of anomaly. We can interpret this occurrence as a "signature" of

the fraud attempt, a feature that makes it unique, singular, and − rightfully so − it should be investigated.

The probability of the simultaneous occurrence of IWA in an examination test can be expressed as follows:

Let:

a - the number of alternatives in each question and

k - the number of IWA,

then the probability (P) can be expressed as:

$$P = \left(\frac{1}{a^2} \right)^k$$

This equation represents a simplification of the problem and is based on the following assumptions:

- The questions are independent; that is, their responses do not interfere with the responses to other questions.
- The probability of each alternative (ABCDE) is equal.

Without this simplification, the solution would be far more complex, requiring a more detailed study of all proposed questions of a specific test to determine the probability of each question and each alternative individually. The proposed equation also implies that the probability of IWA does not depend on the size of the universe (i.e., number of candidates who have taken the test).

This equation demonstrates clearly that as the number of IWA increases, the value of its probability decreases significantly. We can perform the calculation in a small interval to analyze the behavior of the calculated probability. Table 1 shows the calculation of probabilities for the range of 1 to 10 IWA, considering a test with multiple-choice questions with five alternatives ("ABCDE," with $a = 5$).

TABLE I. PROBABILITIES OF IWA IN A MULTIPLE-CHOICE TEST WITH 5 ALTERNATIVES

IWA	P (%)
1	4.000000000000
2	0.160000000000
3	0.006400000000
4	0.000256000000
5	0.000010240000
6	0.000000409600
7	0.000000016384
8	0.000000000655
9	0.000000000026
10	0.000000000001

From Table 1, we conclude that, if the number of IWA is equal to or greater than 3, the probability is already minimal (0.0064%).

VI. PROPOSED METHOD

We have applied a SQL (Structured Query Language) script and relational databases as a traditional Software Engineering tool for helping to detect some IWA. Nevertheless, the results using this technique were not always accurate due to different combinations of IWA and the absence of some IWA in some candidate' responses.

There are many Knowledge Engineering tools that can be applied to detect this kind of fraud attempt with better accuracy. Among others, the main KE tools that can be used in this case are:

- Artificial neural networks (ANN).
- Artificial immune systems (AIS).
- Data mining.
- Case-based reasoning (CBR).

The proposed method is founded on applying case-based reasoning (CBR) as a data-mining tool. CBR is an artificial intelligence technique applied in solving problems and achieving learning, based on past experience such as the use of known cases to solve new cases [10].

CBR methodology fits the purpose of fraud detection proposed in this paper by presenting some features relevant to the problem presented. Among these characteristics, we can highlight the following [11]:

- A CBR system should be able to handle incomplete and highly specific queries.
- It can suggest appropriate cases, even when not all attributes are provided.
- It presents retrieved cases in a rational way; that is, avoiding excessive responses and respecting a limit of retrieved cases specified by the user.

These characteristics favor the application of CBR methodology in this type of fraud detection, since it allows retrieval of similar cases differently from traditional information retrieval, done through use of queries in relational databases.

The proposed method consists in designing a system of CBR for the identification of candidates with IWA above the minimum value of statistical probability (IWA = 3) for the retrieval of similar cases in an IWA file. The system starts out with a file containing all the wrong answers of several candidates, making an automatic sequential search of all similar cases until there is a recovery of more than one case where this similarity occurs. All cases recovered with high similarity will be listed for further investigation of attempted fraud.

As the volume of responses in a test can be quite high, it is necessary to adopt some criteria to restrict the sample studied in order to optimize queries. The first criterion to be adopted is to restrict the sample studied by eliminating the responses of candidates whose number of hits in the test is less than a minimum passing grade. Thus, all responses that have a

number of errors exceeding a certain percentage will be automatically discarded since these candidates failed to pass the competition. This measure considerably reduces the number of cases to be searched.

Another measure of restraint refers to the retrieval of cases in the database. It makes no sense to retrieve cases in which the number of IWA is less than a certain percentage; for example, 60%. The idea is to eliminate instances in which the number of IWA does not reach a certain percentage, which undermines the suspicion of fraud. Thus, we consider only the responses of the candidates whose percentage of IWA is equal to or exceeds 60%.

Looking closely at Figure 4, we note that there are some instances of IWA between the lines, as can be seen in Table 2.

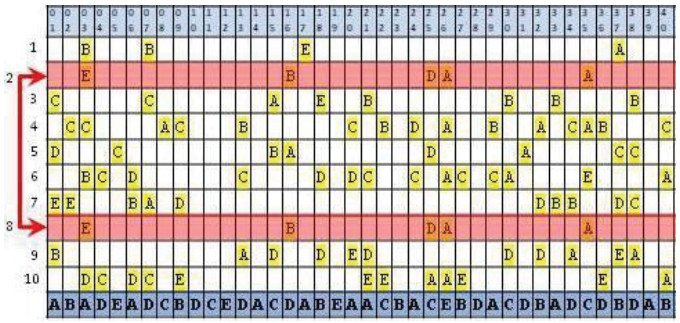


Figure 4. Relevant similar cases in the case base.

TABLE II. OCCURRENCES OF IWA IN THE FIGURE 4

Lines with IWA	Nº of IWA	% of IWA
2 and 8	5	100%
6 and 10	4	29%
7 and 9	1	10%

Although we have identified three instances (2-8, 6-10, 7-9) in which there is similarity among the cases of wrong answers, not all are relevant. If we consider the percentage of IWA in each of the events, we see that only one of them is of interest: line 2 with line 8 (Fig. 4).

The other events are not depicted with evidence of fraud since the percentage of IWA is less than a significant percentage. We can establish a minimum threshold value for this percentage (e.g., 60%), whereas in some cases the candidate who received a given template (crib note) can recognize that one of the answers is wrong and increased his or her number of correct answers or may even have tried to answer some questions on his or her own (chance), and it still is wrong, leading to wrong answers on the same issues but with different alternatives from the original source of cheat or crib.

The proposed method provides an algorithm to scan the responses meeting the restriction criteria predetermined. The process is sequential and begins with the first line, retrieving all its similar cases, then moving on to the next line (2) and doing new queries only from this starting point, reducing in every turn the universe to be searched and optimizing the process of the sequential search. Some CBR tools allow a case to be used as a query, facilitating the research process.

In a real application, we need the candidate identification number in addition to the answers of the multiple choice questions. This will enable us to identify those candidates who have a certain number of IWA.

To automate the fraud detection in an IWA file of multiple-choice questions, it is necessary to make a data-mining tool using an automatic scanning procedure on the IWA case base. As some CBR systems do not allow the creation of scripts to program this automatic scanning, one way to implement such a data-mining procedure would be through the creation of a similarity function, "Sim ()" in a relational database software, enabling the creation of an SQL script to perform the automatic scanning of all the IWA stored in the database, calculating their similarities, identifying which answers show a high similarity and, finally, retrieving them for inspection.

The similarity metric that best fits the proposed model is the average of similarities implemented in the CBR tool. Several simulations were conducted to find the best option for the case studied, including the codification of the alternatives as numeric values (ABCDE as 12345). None of them has better results than the model adopted.

VII. RESULTS AND DISCUSSION

The model simulated in the CBR tool has produced some excellent results in recovering similar cases of IWA. For testing the model, a query was created that had one more wrong answer than the similar cases in the case base, just in the initial position of the string (Fig. 5).

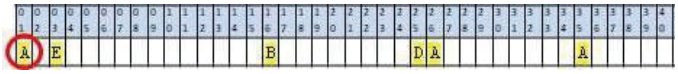


Figure 5. Graphical representation of the query.

This insertion of one more wrong answer in the initial position would make impossible the retrieval of similar cases by means of a SQL query in a relational database, since the blank character (or space) has a smaller value for its internal representation than the letter "A"; this fact increases the distance between the cases registered in the database and the query made. However, the model created with the CBR system proved excellent performance by retrieving all similar cases. Table 3 shows the similarity values calculated by the CBR tool in all the retrieved cases of the case base.

TABLE III. VALUES OF THE SIMILARITIES OF THE RETRIEVED CASES

Case	2	8	4	6	10	5	1	3	7	9
Sim()	1,00	1,00	0,50	0,25	0,25	0,20	0,00	0,00	0,00	0,00

The query on the case base done with the data shown in Figure 5 gives us the results returned by the CBR system used in this study, as presented in Figure 6.

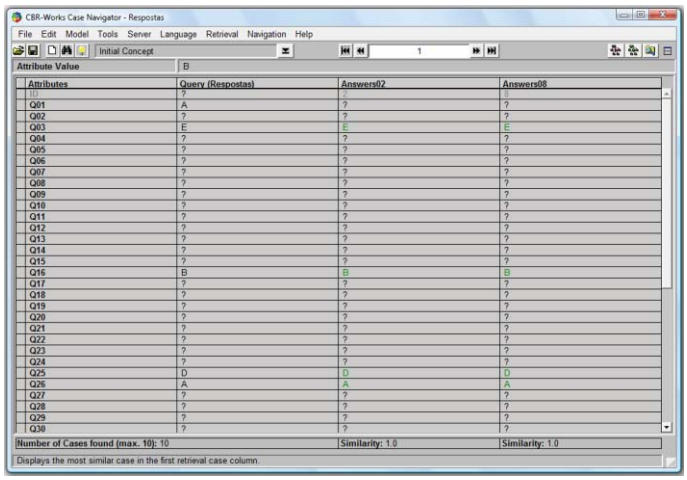


Figure 6. Cases retrieved bythe CBR tool.

Even when the query was submitted with one more wrong answer purposely introduced in the first column, it is clear that case numbers 2 and 8 were recovered with maximum similarity. The remaining cases that did not show a significant number of IWA had low similarity, proving the efficiency and effectiveness of the proposed model.

VIII. Conclusions and Future Work

The methodology of CBR showed excellent performance in the detection of IWA, identifying all the cases in the case base used as a test. While there are some differences between the values of the query with the values found in the case base, the CBR software used in this study correctly retrieved all cases that showed the highest similarity.

We conclude that the method proposed in this study can be applied in detecting and preventing fraud in selection examinations that use multiple-choice questions, allowing the adoption of countermeasures in a timely manner, thanks to its ease of modeling and application.

One limitation of this study is the lack of a more detailed analysis of the behavior of the CBR software with respect to its performance with a case base with thousands of answers from a large group of applicants. However, based on other authors' similar experiences with using bases of more than 200,000 cases [12], the CBR systems have a satisfactory performance. Thus, it is expected that the CBR solution here proposed presents a good performance even in the case of larger bases.

Future research could explore the application of the model proposed in this study for other kinds of questions and tests, adapting the methodology of CBR to the specificity of these applications. Others investigators can also expand this study, making the analysis of the behavior of other CBR software or others KE tools for a greater volume of data.

Acknowledgment

We want to thank Dr. Marcelo Sobottka of the Department of Mathematics at the Federal University of Santa Catarina (Brazil) for his valuable help in determining the probability of IWA in a test of multiple-choice questions.

References

[1] N. Laleh and M. A. Azgomi, "A taxonomy of frauds and fraud detection techniques," in *Communications in Computer and Information Science*, vol. 31, pt. 9, S. K. Prasad, S. Routray, R. Khurana, S. Sahni, Eds. ICISTM 2009, Berlin: Springer-Verlag, 2009, pp. 256–267.

[2] R. J. Bolton and D. J. Hand, "Statistical fraud detection: A review," Statistical Science, vol. 17, no. 3, pp. 235–255.

[3] A. Brabazon, J. Cahill, P. Keenan and D. Walsh, "Identifying online credit card fraud using artificial immune systems," Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2010).

[4] M. Lek , B. Anandarajah, N. Cerpa and R. Jamieson, "Data mining prototype for detecting e-commerce fraud," 9th European Conference on Information Systems, Bled, Slovenia 2001, pp. 160–165.

[5] B. Kusaksizoglu, "Fraud detection in mobile communications networks using data mining," Ph.D. thesis, Department of Computer Engineering, Bahcesehir University Istanbul, 2006.

[6] C. Schommer. (2009). *Discovering fraud behavior in call detailed records* [Online]. Available: http://wiki.uni.lu/mine/docs/Schommer.DataMiningAndFraud.pdf.

[7] J. Li, K. Y. Huang, J. Jin and J. Shi, "A survey on statistical methods for health care fraud detection," Health Care Management Science, vol. 11, No. 3, 2008, pp. 275–287.

[8] C. C. Albrecht. (2008). *Fraud and forensic accounting in a digital environment* [Online]. Available: http://www.theifp.org/research-grants/IFP-Whitepaper-4.pdf.

[9] D. McSherry, "A case-based reasoning approach to automating the construction of multiple choice questions," Lecture Notes on Computer Science, vol. 6176, I. Bichindaritz and S. Montana, Eds. ICCBR 2010, Springer-Verlag, 2010, pp. 406–420.

[10] M. M. Richter, "Case-Based Reasoning Technology, From Foundations to Applications", Lecture Notes in Computer Science, Springer-Verlag, Berlin, 1998, vol. 1400, pp. 1-15.

[11] W. Wilke, M. Lenz, and S. Wess, "Intelligent sales support with CBR," in *Case-Based Reasoning Technology from Foundations to Applications*, M. Lenz, B. Bartsch-Spörl, H. D. Burkhard, S. Wess, Eds. Berlin: Springer-Verlag, 1998, pp. 91-113.

[12] M. Lenz and H. D. Burkhard, "Case retrieval nets: Basic ideas and extensions," in Proceedings of the 20th Annual German Conference on Artificial Intelligence: Advances in Artificial Intelligence, London, Springer-Verlag press, 1996, pp. 227-239.

An approach for retrieval and knowledge communication using medical documents

Rafael Andrade, M. A. R. Dantas
Post-Graduate Program in Knowledge Engineering and Management (EGC)
Federal University of Santa Catarina, UFSC
Florianopolis, Brazil
andrade@telemedicna.ufsc.br, mario@inf.ufsc.br

Fernando Costa Bertoldi, Aldo von Wangenheim
Department of Informatics and Statistics (INE)
Federal University of Santa Catarina, UFSC
Florianopolis, Brazil
{bertoldi, awangenh}@inf.ufsc.br

***Abstract* — The great number of information available in different data sources requires increasingly of search engine system to retrieve as many relevant documents as possible. Clinical medical records contain a great number of information, normally written in free-text form and without a linguistic standard. The physicians do not write the patient's reports using style elements. Consequently, to retrieve the knowledge from these data is not an easy task for the search engine. In this paper we present the development of a model that allows recovering knowledge from textual information in medical documents. Query expansion techniques, which apply knowledge detection assets from the DeCS ontology and language dictionaries, will be used. The goal is to expand the user search and to create a knowledge base to allow its reuse. In order to improve the search results, semantic annotations and negation detection will be used to process medical texts. The case study presented at the end shows that the proposed model was able to achieve a mean accuracy of 90% in its first ten results, while the Boolean model was limited to only 60%. The conclusion is that the user will not need to search in different databases to find the necessary information.**

Keywords-Informartion Retrieval; Query expansion; Negation detection; Medical ontologies

I. INTRODUCTION

The electronic medical information is increasingly present in all hospital and medical clinic. The large number of data that contain medical information is available for researchers, medical institutions, patients and all types of people interested in this kind of information. More and more people can receive and share their information without any restrictions. However, the great number of information available in different data sources requires the use of more intelligent retrieval techniques, focusing on information content and semantic [1].

To demonstrate this requirement, we present a scenario from a typical medical expert task: some common tasks of medical research and clinical care involve, for instance, the checking of exam results, comparison with other reports or statistical analysis of disease patterns in medical records. When a large volume of information is captured and analyzed, the automation of this task is crucial for efficient processing. Medical concepts have to become easily extractable from medical records and they can be compared with other information in order to allow more effective searches[2]. Although the information is available from different forms, the medical texts need to be interpreted by computers in order that information can be processed and effectively shared. To enable this process, the users need to make use of tools in order to increase the accessibility and the data management [3]. On the other hand, to interpret medical text is a difficult task, but it can be easier in comparison as narrative speech, because the medical vocabulary is more restricted. Clinical medical records contain a great number of information, normally written in free-text form and without a linguistic standard. The physicians do not use elements of style for write the patient's reports. A physician can write a report using many ways, and each of them has its own writing style [4]. However, to manipulate these amounts of information is certainly one of the greatest challenges for modern health care search engine systems.

In this paper we describe the use of the knowledge extraction technique from medical ontology and negation detection procedures to define semantic annotation by expand the user's query and to improve the search engine process on medical reports. We describe how to connect the named entities from semantic descriptions on medical ontology DeCS [5] and expand the user's query from DeCS terms. Furthermore, we describe a method that automatic negation detection in medical reports using Natural Language Processing (NLP) techniques [6]. We understand that the use of these three techniques (semantic annotation, query expansion using medical ontology and detecting negated findings and diseases in medical reports), allow for semantically enriched Information Retrieval (IR), expands the user's queries with medical terms and improve the quality of a search in the search engine. In this paper, the search quality is typically described by the use of precision at ten (P@10) metrics.

This paper presents the problem of retrieval information from medical documents on Section 2. A background and related works for semantic search, medical ontologies, Semantic Annotation, Semantic repository and detecting negated phrases techniques are presents on section 3. In Section 4 we present a schema to define automatic annotation from medical ontology DeCS, expand the user query and detect

negative expressions on medical text reports. And finally, the conclusions and future works are discussed in Section 5.

II. PROBLEM

Just because the information does not have a specific standard, the information retrieval (IR) process from patient's medical records is in the most of cases, inefficient. The traditional search engine by using Boolean techniques, do not exploit all the potential existence in this domain-specific knowledge. Although the medical domain provides controlled vocabularies and tools that can be used to search and index documents according to a conceptual hierarchy (e.g. MeSH, DeCS, UMLS, etc.), these engines do not recovery the hierarchical relations among the concepts and semantic such the parents and children links. Sometimes, the medical expert needs to retrieval more relevant information about a specific term or expression. Therefore, this method does not necessarily improve the traditional methods of text search [3]. In order to all related information will be retrievable, it is necessary that the search engines are able to understand the user request and expand the search range, but without losing the quality of search described by precision and recall metrics.

III. BACKGROUND

Before we present our approach, we need to introduce some techniques most used for effective search and retrieval medical information in semantic context. We explain in more detail the methods used for semantic search using query expansion and medical ontologies on Subsection A. In particular, the Subsection B introduces the use of semantic annotation techniques in order to improve on free text search retrieval performance. The detection-negated phrases on medical reports are introduced in Subsection C.

A. Query expansion and ontologies

To solve the problem of semantic search, effective techniques are being widely developed to search and retrieve semantically the medical knowledge [1], [2], [3], [7], [6]. The resource most used is ontology. Ontology is a formal specification of shared conceptualization in order to represent concepts, relationships and rules that manage the relationships. It consists in a way to represent semantic relationships, such objects and relationships in a particular domain [8] and [9]. Ontologies and semantic approaches are used for the integration and IR from different databases [10]. The ontology structure is an important criterion to organize the knowledge and determine the semantic reusability and data interoperability [1].

An important use of the ontology is the query expansion from free textual documents in order to improve IR systems [10]. To expand a query, the user is guided to reformulate the search by add new meaningful terms to the initial query. The system extracts the similar terms from ontology and generates a new query to increase IR systems. The objective here is combining two independent subsystems to retrieval textual and visual information, using the integration of medical knowledge with term expansion they improve the results compared with the two techniques in separated way.

Other works address the problem of query expansion in order to enrich the information retrieval system [1], [7], [11]. A way to expand a query is the use of Term Frequency – Inverse Document Frequency (tf-idf) technique. In order to expand the search, this method includes new terms to the request using words or phrases having meanings similar or related to the original request [11]. In order to evaluate the importance of query expansion and manual indexing, Abdou and Savoy [11] develop a new query expansion model and evaluate the performance of ten different IR models, including probabilistic, language and vector-space models. The authors perform two IR tests (With and Without MeSH ontology) from Medline collection in order to measure retrieval performance. The method presented by authors is 170% more precise when compared to the classical tf-idf vector-space model. Including the MeHS when indexing scientific articles this work improve more 8.4% performance on the best IR scheme. Other way to expand a query is the use of ontologies. Bhogal et al.[7], presented a range of definitions of information retrieval focused on the use of context for query expansion. They have discussed the problem of the use of ontologies for a range of information retrieval tasks and their use in the area of query expansion. For more information see [7].

B. Semantic annotation

Important studies show that extract semantically the information providing semantic annotation using references from ontologies and knowledge databases, can improve more precision on IR systems [2], [12], [13]. In this research area, semantic text annotation is effective texts processing that map corresponding concepts from ontologies and annotate these concepts in the document [2]. Annotations generate and use a set of metadata that provide reference to named entities from ontology mentioned in the medical documents [12]. This metadata generate a knowledge database in order to support semantic search in document repositories. The unstructured medical text gain an important metadata supplied by the ontology. The Semantic annotation enable computer to understand the semantic meaning of a data set and increase the quality of retrieved information and system interoperability [14]. Gschwandtner et al. [2], presents a specific automatic semantic annotation system that maps the concepts from medical ontology to free medical text. They customize other annotation systems (annotating web-pages) to understand a specific medical domain. They develop an application that generates a map of concepts from medical terminologies and the medical concepts included in documents are annotated semantically by a metadata. The medical experts can visualize, control and correct all kinds of annotated information for processing a medical document. Her works show that mapping medical concepts from ontology can provide semantically accurate information for text processing and helps to removing ambiguity from different meanings.

Lourenco et al. [13], identifies relevant terms in documents based on a lexicon Named Entity Recognition (NER) process. The goal is annotate the occurrences of biological classes from abstracts or full-texts within PubMed library and implement the semantic indexing of documents and terms. The technique is used to extract the information from medical libraries, tokenize (pre-processing documents) and apply a lexical dictionary in order to perform the recognition of named entities. In their results obtained, the system is able to reduce significantly the

number of irrelevant documents without significant loss of irrelevant documents. Although the technique used was different, the two works aim to improve the indexing and semantic information retrieval from medical documents.

C. *Negative expressions*

Retrieval negative expressions is as important as retrieval any other information from the database. A negative expression can invalidate a search. Therefore, the system engine has to decide whether this expression contained in the database will be excluded or not from the search [15]. Negative phrases/expressions are used by doctors to write patient's diagnosis, medical procedures as well it helps the medical expert to identify the occurring symptoms and diseases from medical documents [6]. For instance, in medical document is common diagnosis that contains text like "the patient does not have hypertension", or "the patient has hypertension". These expressions create a problem to the search engine. If the engine detects the first expression as true, all documents that contain this kind of expression will be retrieved and the result will contain false information. Because of this, the process of negation detection requires a lot of knowledge about the language to correctly identify negated words or terms from an expression. The detection of negative statements in medical databases can reduce the search space thereby rendering the search process more agile.

Gingl et al. [6] developed a method to classify and detect negated information occurring in Clinical Practice Guidelines (CPG) on syntactical level using grammatical information of the English language. Her studies show that grammatical elements are used to decide whether a phrase is negated or not. The negation classification allows medical expert to decide which therapies or treatment options are best or not applied to the patients. Their syntactical methods detection improves the values in precision and recall. One previous study identifies concepts from a medical ontology UMLS and using a lexical scanner in order to recognize and classify a large set of negated patterns occurring in the text [15]. The authors developed a program based on existing technology for implementing parsers based on context-free grammar. The results of your study show that the system presented has a recall and precision between 91.8 and 95.7 percent in detecting negations on medical documents.

IV. METRHODS AND MODEL DESCRITION

Although, most of this works apply different techniques in order to retrieval the information from medical documents, all of these works have a very particular function: improve the search engine process on medical data. We understand that is not easy to improve the current process, but based on the union from these three previews presented techniques, we refine the actual approach.

Our work differs from the previous works, because we aim to provide to the medical user a document range much more extensive and effective. We want to present most relevant documents so that the user does not need to dispend too much time to find information or the user does not need to look in different databases in order to find the information. We aim also to provide a semantic list created from a knowledge database that is frequently updated in order to permit the user find the information in an intuitive way. The approach presented here is based on the query expansion technique using ontologies that defines the use of terms in hierarchical tree and by the use of synonyms. In order to improve the search, semantic annotations will be used in medical texts. This technique includes the use of named entities and the detection of negative phrases to increase the research universe and reduce the number of less relevant responses. We will create a metadata repository which be used to generate concepts from the ontology, extract the information from medical documents and make the text pre-processing.

In figure 1, we describe how the term is indexed and how the users receive de results. The most important component is the Query Engine module. It is responsible to access the semantic knowledge database and the text extraction modules (Semantic Annotation, Negation Detection and Query Expansion). The core of the query search engine is composed for two main elements: Indexing and Search. The Analyzer process is the conversion of texts in terms. The terms are used to determine which documents match a query during the search. The Analyzer is the component of the analysis process that performs a series of operations to facilitate the indexing. It converts the tokens to lowercase letter, removes stressed characters, and removes common words, like articles and pronouns (stop words) and extracts words root (stemming).

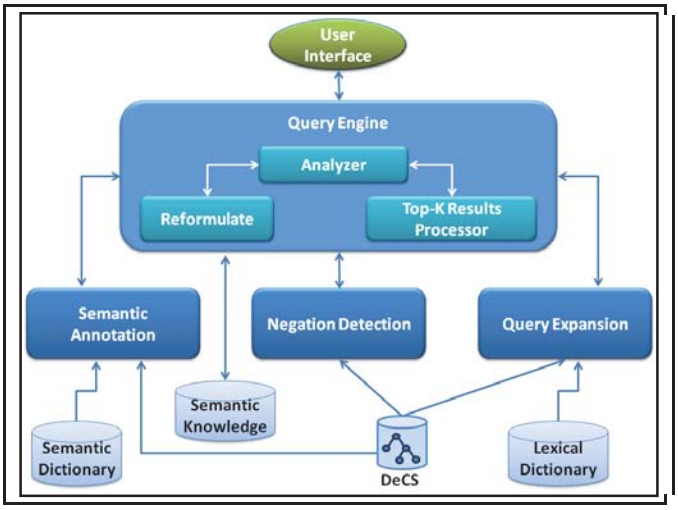


Figure 1. Semantic Information Retrieval from medical databases.

The Analyzer also uses NLP to extract terms from DeCS ontology, in order to classify and define the relationship between two terms, and extract the negative sentences. In our system, we assume that the knowledge database should be built and associated from information sources, which utilize medical ontologies, lexical dictionaries, and the analyzer component in order to describe concepts that appear in documents stored on medical database.

The Reformulate component is responsible to connect others text extraction modules, reformulate the query and store the results on the Semantic Knowledge database. The semantic annotation, query expansion and Negation Detection modules are used by the Top-k Results Processor in order to retrieval and ranking the information. The ranking method is an adaptation of the vector space model (Van Rijsbergen, 1975),

which defines keywords weights that appear in the document. The weights are computed automatically based on the frequency of instances in each document. The number of occurrences for a document instance is defined by the number of the times that an instance appears on text

The Negation Detection module is a NLP technique, which finds negations sentences into the medical text report in our database. As described in [6], we need to detect five negation classes (adverbial negation, intra-phrase triggered negation, prepositional negation, adjective negation and verb negation) in order to improve parameters of recall and precision by the user query. The first step in the detection process is to create a list of negative terms (not, ever, nerver, none, neither…). As a single word does not define a negative term, we need to analyze an expression. Furthermore, most of medical findings do not have 100% sure if a expression is affirmativ or negative, we created a list "hypothetical" terms, like, *cannot assert*, *cannot exclude, cannot completely rule out*…

Then we map biomedical text founded in DeCS ontology in order to create a negative expression dictionary with this five classes of negation expressions. We discovery this negated expressions on the medical report in order to refine the user query by sending to the query engine module. As a result, we generated a list that has been indexed with the terms found in order to facilitate further search. Figure 2 shows a list of excerpt from negative expressions indexed by the algorithm in the Portuguese language.

206876|Bloqueio divisional ântero-superior de ramo esquerdo, **<hypothetical>** nao se pode excluir **</hypothetical>** **<finding>** fibrose inferior **</finding>**
200330|Alterações da repolarização ventricular em parede inferior com onda T **<negation>** negativa **</negation>** em D3 e a Vf compatível com **<finding>** isquemia subepicárdica **</finding>**
295168|**<hypothetical>** Não se pode descartar **<hypothetical>** **<finding>** fibrose inferior **</finding>**
84079|**<finding>** taquicardia supraventricular **<negation>** não**</negation>** -sustentada **</finding>**

Figure 2. Stretch of annotated medical findings using our system.

The query expansion process uses the DeCS ontology by adding more medical information in the user research. The DeCS ontology is used for indexing the medical report text in database, because this ontology contains concepts, relations of synonymy and related concepts. This facilitates the expansion and extraction of terms from the DeCS Descriptors. If the set of terms have relation with the user query and DeCS descriptors, we generate a new query, which contain all terms presents on DeCS synonymous. Moreover, we use the lexical dictionary in order to find new synonymous expressions on medical reports. For instance, when a medical search for the disease "asthma", the DeCS ontology find four answer terms: "Asthma", "Asthma, Exercise-Induced", "Dyspnea, Paroxysmal" and "Asthma, Aspirin-Induced". If we use the lexical dictionary, we expand the results including three new terms: "Bronchitis", "Infectious bronchitis virus" and "Bronchitis, Chronic". It is happened because in our dictionary the term "Asthma" has relationship with the term "Bronchitis", but in DeCS ontology this connections does not exist in the same hierarchical tree.

To define semantic annotation, we use a lexicon dictionary, a list of stop words, and a dictionary of negative sentences for Portuguese language. Moreover, we use the DeCS ontology to create an automatic semantic annotation on text and to store on semantic repository. The purpose of the semantic repository is not only exact terms retrieval contained in the queries, but also retrieval related entities, as synonyms and related terms stored hierarchically on medical ontology. Using structural document similarities, we identified named entities, associate these entities with concepts, and define the semantic annotation. These annotations are stored on semantic repository for further future research.

V. RESULTS

In order to validate a knowledge base we needed a people who have a high knowledge degree in a specific area and ability to transfer this knowledge. These people are called domain specialist. The specialist help to structure the knowledge and allows to minimize the indexing errors of computer systems.

In this work we use an specialist in medical domain in order to validate the creation and the annotation of the knowledge base. This validation enable to the user to represent and communicate the knowledge in future search.

In order to validate our database, the specialist have select randomly 172 reports from computer tomography. Figure 3 shows the result of evaluating from the medical especialist in CTs findings. The specialist found 142 reports categorized as affirmative; 19 reports categorized as negative; four as ambiguous reports and seven reports are not found. In this case, we can see that only for these 172 reports, a traditional system would index those 19 reports (11%) as true and the traditional search engines would be unable to define if a report should be part of the response or not.

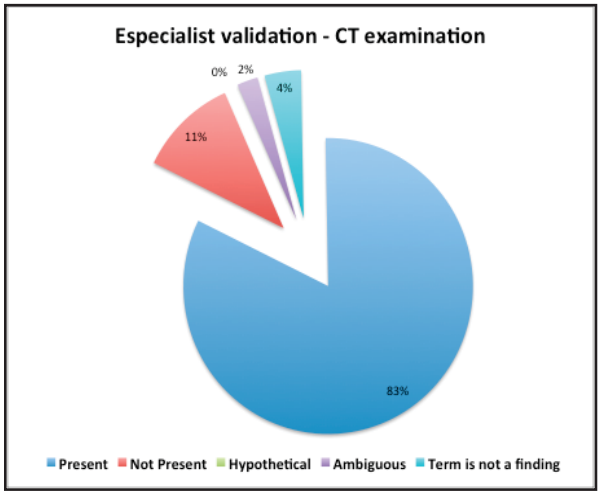


Figure 3. Validation reports for CT examination by medical specialist.

Once developed the prototype, several experiments were conducted to verify the conduct of the search system in real situations and in this case we verify the compliance with the model described here. The work tests were developed to validate the recovery process and communication of knowledge in health domain.

In this scenario, the user has done a search for reports that contain terms used in the ontology. We also investigated reports that contain negative expressions and reports that contain expressions, which are not known by the ontology. The aim of this study is show to the user the queries that he has required and similar terms. If the system does not find a result, the word or expression will be recorded in our daily database and indexed in the next interaction. This information is extremely valid, because the system can "learn" new terms, according users will use the system. And for users, they can learn more about the terms used by other professionals.

To illustrate the operation of our system, we developed a study case that considered four sentence searches:

Q1: "Presence of thyroid nodules"

Q2: "Absence of Lithiasis"

Q3: "popcorn calcification in the brain"

Q4: "Right MCA Aneurysm"

The query Q1 was done with terms that are present in domain ontology. In this case, the documents that are expected to answer to Q1 are all documents that contain *"thyroid nodules presence"* + *"Presence Thyroid Gland Nodule"* + *"presence Thyroid Neoplasms"* + *"presence Thyroid Disease"* + *Gland Thyroid presence"* (result of the research expansion). As the term "thyroid nodules" is known in ontology, the system returned all the documents that had some relation with the set of term listed in the expanded query. Still, the results may not contain negative expressions, because the user did not select the specific field to search for phrases with a negative sense. The queries results are described in Table 1.

TABLE I. CONSULTATION USING THE DEVELOPED METHODOLOGY

Query	Results	Time in ms	P@10
Q1 "Presence of thyroid nodules"	221	557	0,7*
Q2: "Absence of Lithiasis"	432	612	1,0
Q3: "popcorn calcification in the brain"	109	736	0,9*
Q4: "Right MCA Aneurysm"	79	589	1,0
* In this query we found hypothetical terms that can not be considered valid			

In order to validate this case study and define the system's accuracy, we used the metric of P@10. For each query defined in the beginning of this section, the result is shown in Table 1. For a better search, the queries was conduct after the validation by the specialist and the connection of the daily use terms with the ontology terms. The average precision of all queries was 0.9000.

The proposed methodology evaluated the accuracy of the first ten results (P@10) for the two research models: the traditional method (described here as IR) and the new model (IR+QE+Neg). We can notice that in all the submitted queries, the new research model had much better results than the traditional method. For the user to get a satisfactory result in the traditional method, we needed to perform various different queries, already in IR+QE+Neg method was carried out only one query for retrieve the results. Figure 4 show only the best results with the traditional IR compared with IR+QE+Neg model.

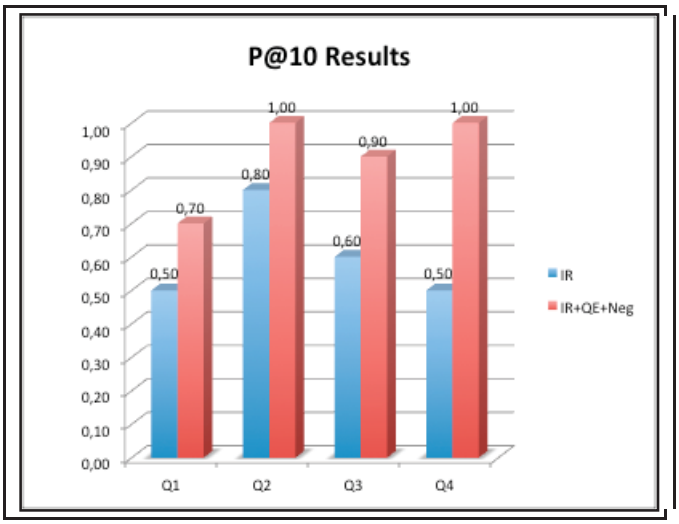


Figure 4. Comparison of traditional method with the proposed model.

With the exception of Q1, all other queries obtained accuracy equal to or above 90%. The accuracy rate is a little lower in Q1 due to the fact that the responses contain many hypothetical expressions. In this way, we cannot confirm the accuracy of the reports. But even so, the accuracy in Q1 was still much higher than the traditional IR method (50%).

As the average precision of the two models can be computed, we can also compare our model with the most important articles showed in section II. Figure 5 shows a comparison between our proposed systems with four others IR models. However, the average precision can only be considered in the expansion methods and in our method. Likewise, the overall precision was only available in works of detecting negative expressions. Because this work involves the use of two different models, it was difficult to define an efficient compared against the models available in literature. Still, the proposed model showed accuracy well above what is available in the literature.

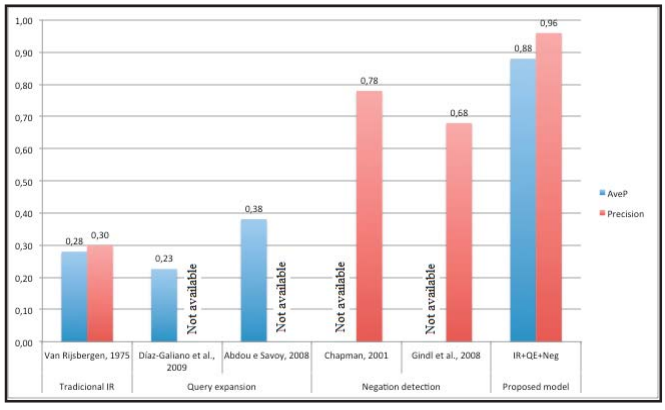


Figure 5. Results of average precision and overal precision compared with others important works.

The model presented by Díaz-Galiano *et al*. [1], uses the database of the Cross Language Evaluation Forum (CLEF) in 2005 and 2006. The base has 50,000 annotated images, which are available for testing accuracy. The other works have used a proper database for measuring the performance of its experiments. The traditional method was used to compute the 352 sentences accuracy rating, which reached 0.30. Dias-Galiano *et al.* [1] using 50,000 reports came annotated media accuracy of 0.23. Abdou and Savoy [11] were able to reach 0.38 of precision in their responses using a set of 1,000 reports. Chapman [16] and Gindl *et al.* [6] show only the total precision in their experiments. Chapman used a database of 1058 reports and came to a precision of 0.78. Gindl *et al.* [6] developed their research in a base of 558 awards and reached the total of 0.68 accuracy. Since our model, which uses three RI presented techniques, came to an accuracy rating of 0.88 and overall accuracy 0.96.

VI. CONCLUSIONS

In this paper we described a method that enables semantic search on medical text reports using ontologies in specific domains. We implemented a parser that extracts information from a database, normalize and store into an index database. In some searches conducted, it is possible to measure the functionality and applicability of the proposed architecture. Also we had used DeCS ontology for clinical toxicology area in order to classify the available information and establish well-defined relations between this information. The use of this ontology will optimize the process of care, increasing reliability and efficiency of medical professional.

In our approach the queries will be generated from keywords by a natural language query, or by form-based interface where the user can explicitly select ontological items. When a user performs a search, the queries are executed against the semantic repository, which returns a list containing instances that satisfy the search. If the required information is not indexed on semantic repository, the engine performs the search again directly to the medical database. The result is an index that is stored on semantic repository and is available for further queries. Before sending an answer to the user, the system retrieves the information, ranking and creates the top-k result list.

We also discussed three general data integrations approaches that use the DeCS ontology to provide access to medical database. According to an evaluation on the use of these three techniques, we believe that the use of this semantic information retrieval could assist the user to improve the results in your research, by expanding the query and enrich the values from precision and recall.

In future we aim to develop a semantic index that interprets the information received from a medical database. We will develop basic NLP techniques, in order to construct the semantic queries. An ontological process will be provided to map the word on the text with the ontology DeCS concepts.

For the handle of negative statements, we will develop a text-mining algorithm to extract medical terms from our medical diagnosis database. Then we need to classify the text, e.g., “the patient does not have hypertension”, “the patient has hypertension”, or “the patient has high blood pressure”. Furthermore, we need to make a relation between the terms on medical ontologies, with the terms founded in the medial report (e.g., symptoms and human body parts). The goal in the future is to define high weights for negative expressions, generate a dictionary that will be use for faster search and improve more precision and recall to the user queries.

REFERENCES

[1] Díaz-Galiano, M.C., M.T. Martín-Valdivia, and L.A. Ureña-López, *Query expansion with a medical ontology to improve a multimodal information retrieval system.* Computers in Biology and Medicine, 2009. **39**(4): p. 396-403.

[2] Gschwandtner, T., et al., *Easing semantically enriched information retrieval--An interactive semi-automatic annotation system for medical documents.* International Journal of Human-Computer Studies, 2010. **68**(6): p. 370-385.

[3] Moskovitch, R. and Y. Shahar, *Vaidurya: A multiple-ontology, concept-based, context-sensitive clinical-guideline search engine.* Journal of Biomedical Informatics, 2009. **42**(1): p. 11-21.

[4] Sager, N., C. Friedman, and M.S. Lyman, *Medical Language Processing: Computer Management of Narrative Data*. 1987: Addison-Wesley Longman Publishing Co., Inc. 320.

[5] BIREME. *DeCS/VMX*. 2010 15 fev 2009]; Available from: http://decs.bvs.br/vmx.htm.

[6] Gindl, S., K. Kaiser, and S. Miksch, *Syntactical negation detection in clinical practice guidelines.* Studies in health technology and informatics, 2008. **136**: p. 187.

[7] Bhogal, J., A. Macfarlane, and P. Smith, *A review of ontology based query expansion.* Inf. Process. Manage., 2007. **43**(4): p. 866-886.

[8] BERNERS-LEE, T., J. Hendler, and O. Lassila, *The semantic web.* Scientific American, 2001. **284**(5): p. 34-43.

[9] Studer, R., V.R. Benjamins, and D. Fensel (1998) *Knowledge engineering: Principles and methods*. Data & Knowledge Engineering **25**, 161-197 DOI: 10.1016/S0169-023X(97)00056-6

[10] Munir, K., et al., *Semantic Information Retrieval from Distributed Heterogeneous Data Sources.* FIT Islamabad, special track on bioinformatics for academia and industry, 2006.

[11] Abdou, S. and J. Savoy, *Searching in Medline: Query expansion and manual indexing evaluation.* Inf. Process. Manage., 2008. **44**(2): p. 781-789.

[12] Kiryakov, A., et al., *Semantic Annotation, Indexing, and Retrieval.*, in *The SemanticWeb - ISWC 2003*. 2003. p. 484-499.

[13] Lourenço, A., et al., *BioDR: Semantic indexing networks for biomedical document retrieval.* Expert Systems with Applications, 2010. **37**(4): p. 3444-3453.

[14] Agosti, M., G. Bonfiglio-Dosio, and N. Ferro, *A historical and contemporary study on annotations to derive key features for systems design.* International Journal on Digital Libraries, 2007. **8**(1): p. 1-19.

[15] Mutalik, P.G., A. Deshpande, and P.M. Nadkarni, *Use of general-purpose negation detection to augment concept indexing of medical documents: a quantitative study using the UMLS.* Journal of the American Medical Informatics Association : JAMIA, 2001. **8**(6): p. 598-609.

[16] Chapman, W., *A Simple Algorithm for Identifying Negated Findings and Diseases in Discharge Summaries.* Journal of Biomedical Informatics, 2001. **34**(5): p. 301-310.

A WordNet-based Semantic Similarity Measure Enhanced by Internet-based Knowledge

Gang Liu[1, 2], Ruili Wang[1, 2*]
[1] School of Engineering and Advanced Technology,
Massey University, Palmerston North, New Zealand
[2] State Key Laboratory for Novel Software Technology,
Nanjing University, Nanjing, P.R. China
[*] R.Wang@Massey.ac.nz

Jeremy Buckley
Simplar Limited
Wellington
New Zealand

Helen M. Zhou
School of Electrical Engineering
Manukau Institute of Technology
Auckland, New Zealand

***Abstract*—Approaches for measuring semantic similarity between words have been widely employed in various areas such as Artificial Intelligence, Linguistics, Cognitive Science and Knowledge Engineering. A new semantic similarity measure is proposed in this paper, which exploits the knowledge retrieved from a semantic network (i.e., WordNet) and the Internet. In particular, the structure information from WordNet and the statistic information obtained from the Internet are combined to quantify the semantic similarity between words. The new similarity measure is evaluated by comparing the rating results with two sets of human benchmark data. Experimental results indicate that, the proposed similarity measure outperforms previous WordNet-based semantic similarity measures.**

Keywords-semantic similarity; WordNet; Normalised Google Distance

I. INTRODUCTION

Measuring semantic similarity between words has been playing an important role in many research areas such as Artificial Intelligence, Linguistics, Cognitive Science and Knowledge Engineering [25]. The semantic similarity between words can be utilised to disambiguate word senses [15], detect malapropism [12] and improve the accuracy of ontology mapping [8]. Accurate estimation of semantic similarity can benefit all these applications in the aforementioned research areas and enhance their performances.

Measuring the semantic similarity or distance between words is a process of quantifying the relatedness between the words utilising the knowledge obtained from certain information sources. These information sources can be: (i) lexical recourses such as dictionaries, thesauri and semantic networks [16]; (ii) collections of documents such as corpus [25], and (iii) the Internet [3,5].

Semantic networks are considered as better choices for estimating semantic similarity than other lexical resources [2]. WordNet [4] is one of the most popularly used semantic networks for estimating semantic similarities [1,2,6,7,10,12,16, 19,20,24,26].

Based on the way of utilising WordNet, the WordNet-based semantic similarity measures can be classified into three categories: (i) *node-based methods*, which estimate the semantic similarity by computing the amount of information contained by related words in WordNet (e.g., [19]). (ii) *edge-based methods*, which assess the semantic similarity by calculating the length of edges on the shortest path between the words in WordNet (e.g., [26]). (iii) *hybrid methods*, which combine the information content theory and the structure information from WordNet to estimate the semantic similarity between words (e.g., [7]).

This paper proposes a hybrid method utilising the Internet as a corpus to overcome shortcomings of previous hybrid methods. In particular, our method utilises Normalised Google Distances (NGD) [3] to derive the semantic similarity between words combining with the structure information from WordNet. According to our experimental results, introducing the Internet knowledge to WordNet-based semantic similarity measures can increase the accuracy of estimating semantic relatedness between words.

The remainder of this paper is organised as follows. Section II provides a brief introduction on WordNet and reviews previous WordNet-based semantic similarity measures. Section III introduces our new WordNet-based semantic similarity measure, and the motivations of using the Internet as a corpus and NGD as a tool for estimating semantic relatedness between concepts. Section IV presents evaluation experiments and their results. Section V concludes the paper.

II. WORDNET AND WORDNET-BASED SEMANTIC SIMILARITY MEASURES

WordNet is a semantic network, which is organised in such a way that synsets and wordsenses are the nodes of the network, and relations among the synsets and wordsenses are the edges of the network. In WordNet, each meaning of a word is represented by a unique *wordsense* of the word, and a *synset* (stands for "synonym set") is consisting of a group of wordsenses sharing the same meaning. More than two thirds of the nodes in WordNet are synsets. *hyponymOf* is the key relationship for noun synsets in WordNet, which has been widely used to estimate the semantic relatedness among nouns.

WordNet has been commonly used to measure semantic similarity among words since it has the inherent advantages of being structured in the way of simulating human recognition behaviours [4]. The reminder of this section will give a brief review on previous WordNet-based semantic similarity measures organised in the aforementioned three categories.

A. Node-based methods

Node-based methods use the amount of information contained by related nodes (i.e., related concepts) in WordNet to estimate semantic similarity between the concepts of interest, i.e., c_1 and c_2. Thus, this kind of methods is also called as information-based methods.

Most of node-based methods employ the information content to quantify the amount of information that a concept contained. According the definition in the information theory [23], the *Information Content (IC)* of a concept c can be quantified by $IC(c) = -\log(P(c))$, where $P(c)$ is the probability of c appearing in a corpus.

Resnik [19] believed that the similarity of c_1 and c_2 is determined by the closest common superordinate concepts (i.e., hypernyms) of c_1 and c_2 in WordNet. Thus, Resnik proposed to use the IC of the lowest hypernyms of c_1 and c_2 to calculate the semantic relatedness between c_1 and c_2.

Richardson and Smeaton [20] amended Resnik's method [19] by using a different equation to calculate the IC of the lowest hypernyms of c_1 and c_2.

Banerjee and Pedersen [1] developed a score schema to estimate the relatedness through cross comparing the words being used in the definition of c_1 and c_2 and their hypernyms.

The drawbacks of node-based methods include: (i) it is a time-consuming work to analysis the corpora for estimating the IC values; (ii) unbalanced contents of the employed corpora may significantly decrease the accuracy of the IC values.

B. Edge-based methods

Edge-based methods utilise the shortest path between concepts (i.e., c_1 and c_2) in WordNet to estimate the semantic relatedness between c_1 and c_2. Lengths of all edges on the shortest path are accumulated to quantify the semantic similarity. It is the way of calculating the length of edges that differentiates methods in this category.

Sussna [24] proposed to use the depths of the taxonomy tree of WordNet that c_1 and c_2 are in, and the type-specific fanout factors to compute the semantic relatedness between c_1 and c_2. The *type-specific fanout factor* is defined by the type of the edges connecting c_1 and c_2, and the numbers of edges setting off from c_1 and c_2, respectively.

Leacock and Chodorow [10] suggested that the semantic relatedness between c_1 and c_2 can be estimated using the edge number of the shortest path between c_1 and c_2, and the depth of the involved taxonomy tree in WordNet.

Yang and Powers [26] proposed to utilise the properties of edges and the edge number of the shortest path to calculate the semantic similarity between c_1 and c_2. The properties of edges considered are: (i) the relation type of the edge, and (ii) the depth of the edge.

Hirst and St-Onge [6] utilised edge directions to estimate edge lengths of the shortest path. An edge direction is determined by the relation type of the edge. The directions of edges in a path between concepts are used to determine if the path is allowable, and to define the strength of turns in the allowable path. Hirst and St-Onge used the length of the shortest allowable path between c_1 and c_2, and the number of the direction turns in the path to estimate the semantic relatedness between c_1 and c_2.

The accuracy of the edge-based methods is significantly affected by the lack of considering the varieties of semantic distances between adjacent words, which is caused by the uneven word densities in WordNet. For instance, Sussna [24], and Leacock and Chodorow [10] did not consider the variety of all edge lengths; Hirst and St-Onge [6], and Yang and Powers [26] ignored the diversity of semantic distance among edges that are in a same type.

C. Hybrid methods

Hybird methods combine the information from different resources to estimate the semantic similarity between concepts, e.g., combining the IC of concepts with the structure information retrieved from WordNet to conduct the estimation.

Jiang and Conrath [7] accumulated the scaled length of all the edges in the shortest path between concepts to estimate the semantic similarity of the concepts. The edge length between concept c (a node in the shortest path) and concept p (the parent node of c in the shortest path) is calculated by $length(c, p) = \log(P(p)) - \log(P(c))$. Structure information from WordNet such as (i) local density of the nodes in the shortest path, (ii) depth of the nodes, and (iii) the relation type of edges are utilised to scale the edge length.

According to previous research [2,16], hybrid methods outperform the other two kinds of methods. However, hybrid methods share a common weakness with all node-based methods: the data sparseness of the employed corpus. This weakness can be caused by two reasons: (i) the size of the corpus is relatively undersized, and (ii) the coverage of the corpus is not well balanced. One possible way to complement the deficiency is to use a better corpus.

III. THE PROPOSED SEMANTIC SIMILARITY MEASURE

Our semantic similarity measure is a hybrid method combining the knowledge acquired from the Internet and the structure information from WordNet to quantify the semantic relatedness between concepts. In our method, the Internet is used as a corpus for estimating the semantic distance between adjacent concepts along the shortest path between the concepts of interest, i.e., c_1 and c_2, in WordNet.

A. The Internet as a corpus

We believe that the Internet is an ideal corpus for calculating IC of concepts. The reasons include: (i) the Internet is the current largest electronic text source; (ii) the Internet is a relatively balanced knowledge source; (iii) the knowledge in the Internet can be updated regularly and up-to-date.

Well-performed Internet search engines such as Google enable us to use the Internet as a huge corpus. With help from these search engines, the frequency of a word appearing in the

Internet can be easily estimated using the number of web pages that contain the word.

We use hypernyms as context words to tag word senses. A corpus with tagged word senses is one of the key requirements for semantic similarity measures to give accurate estimations on the similarities between concepts when corpora are involved. In word sense disambiguation area, context words are believed to share linguistic information with the word of interest and can be used as a word senses indicator [15]. Experimental results of *Clever Search* developed by Kruse et al. [9] show that hypernyms are effective to be used as context words to indicate word senses for querying the Internet.

B. Normalised Google Distance

We use Normalised Google Distance (NGD) [3] between the end nodes of each edge to estimate the edge lengths of the shortest path. The *NGD* utilises the numbers of Google search results to approximate the Kolmogorov Complexity [11] of words for calculating the semantic relatedness between the words. Given two words x and y, the NGD between x and y can be calculated by the following equation:

$$NGD(x, y) = \frac{\max\{\log f(x), \log f(y)\} - \log f(x, y)}{\log N - \min\{\log f(x), \log f(y)\}}$$

where $f(x)$, $f(y)$, and $f(x, y)$ are the numbers of web pages found by Google that contain x, y, and both x and y, respectively. N is a normalising factor, whose value is a reasonable large value that is larger than both $f(x)$ and $f(y)$.

C. Calculation of Semantic similarity

We employ the following structure information from WordNet to scale the edge lengths of the shortest path: (i) the relation type of the edges; (ii) the number of adjacent words that the end nodes have; (iii) the depth of the end nodes in their own taxonomy tree, and (iv) the maximum depth of the involved hierarchy structures.

Since Jiang and Conrath's method [7] outperforms other existed semantic similarity measures [2,16], we proposed an equation based on their equation to calculate the edge lengths of the shortest path:

$$wt(c, p) = \left(\beta + (1-\beta)\frac{\overline{E}}{E(p)}\right)\left(\frac{D}{d(p)}\right)^{\alpha} NGD(c, p)\ T(c, p)$$

where c is a node on the shortest path; p is the parent node of c; $\overline{E}$ is the average local density of all nodes; $E(p)$ is the local density of p (i.e., the number of its children nodes); D is the maximum depth of the hierarchy structure that c and p are in; $d(p)$ is the depth of p; α, β and $T(c, p)$ are the depth factor, density factor, and edge type factor, respectively; $NGD(c, p)$ is the Normalised Google Distance between c and p.

The semantic relatedness between c_1 and c_2 can then be calculated by the following equation:

$$rel(c_1, c_2) = \sum_{n \in \{S(c_1, c_2) - Sol(c_1, c_2)\}} wt(n, parentOf(n))$$

where $S(c_1, c_2)$ is the set of nodes in the shortest path between c_1 and c_2 ; $Sol(c_1, c_2)$ is the set of nodes in the shortest path which have no parent nodes in the path; n is a node included by $S(c_1, c_2)$ but excluded by $Sol(c_1, c_2)$.

IV. Evaluation

This section introduces the evaluation of the proposed semantic similarity measure. Similarity ratings computed by our method are compared with human judgement data in this evaluation. Two sets of prevalent human benchmark data are employed: (i) Rubenstein and Goodenough data set (RG-Set) [22] and (ii) Miller and Charles data set (MC-Set) [13]. The Pearson Product-moment Correlation Coefficient [21] is employed to calculate the consistency between similarity ratings.

Our method is compared with Jiang and Conrath's method [7] as well. Jiang and Conrath's method has been considered as one of the best semantic similarity measures [2,16]. In other words, we can believe our method is better than other methods if our method can outperform Jiang and Conrath's method.

Table I reports the similarity ratings on MC-Set computed by Jiang and Conrath's method and our method, respectively.

TABLE I. Human and computed rating on MC-Set

MC-Set	Human Ratings	Jiang and Conrath's Method	Our Method
car - automobile	3.92	3.409	3.473
gem - jewel	3.84	3.403	3.496
journey - voyage	3.84	3.062	3.303
boy - lad	3.76	2.899	3.256
coast - shore	3.7	3.402	3.172
asylum - madhouse	3.61	3.234	3.18
magician - wizard	3.5	3.433	3.445
midday - noon	3.42	3.402	3.472
furnace - stove	3.11	2.055	1.337
food - fruit	3.08	2.851	1.542
bird - cock	3.05	3.365	3.187
bird - crane	2.97	3.365	3.028
tool - implement	2.95	3.153	3.05
brother - monk	2.82	0.943	2.17
crane - implement	1.68	2.921	2.643
lad - brother	1.66	1.409	1.927
journey - car	1.16	0	0
monk - oracle	1.1	0.207	1.459
food - rooster	0.89	0.185	0.6
coast - hill	0.87	2.937	2.686
forest - graveyard	0.84	0.195	0.197
monk - slave	0.55	0.968	2.089
coast - forest	0.42	2.311	1.166
lad - wizard	0.42	1.363	1.466
chord - smile	0.13	2.193	1.117
glass - magician	0.11	0.032	1.019
noon - string	0.08	0	0
rooster - voyage	0.08	0	0
cemetery - woodland	0.95	0.178	0.185
shore - woodland	0.63	2.262	1.472

Table II shows that when MC-Set is used, the correlation coefficients between human rating and the ratings computed by Jiang and Conrath's method and our method, respectively. The correlation of human judgement replication listed in the table was reported by Miller and Charles [13].

TABLE II. THE CORRELATION COEFFICIENTS ON MC-SET

Methods	Correlation
Jiang and Conrath's and human rating	0.7509
Our method and human rating	0.8087
Human judgement (replication) [13]	0.8848

Because of the limitation of space, we will not report our similarity ratings on RG-Set in this paper. Table III reports when RG-Set is used, the correlation coefficients between the human ratings and the ratings computed by each of Jiang and Conrath's method and our method.

TABLE III. THE CORRELATION COEFFICIENTS ON RG-SET

Methods	Correlation
Jiang and Conrath's and human rating	0.7129
Our method and human rating	0.7634

The above experimental results indicate that our method outperforms Jiang and Conrath's method. Since Jiang and Conrath's method is one of the best existed semantic similarity measures [2,16], we believe that our method can provide better estimations on semantic similarities than previous methods.

V. CONCLUSIONS

In this paper, we proposed a new hybrid method for semantic similarity measure by combining the structure information from WordNet and the statistic information obtained from the Internet.

In our method, the length of the shortest path between two concepts in WordNet is used to measure the semantic similarity of the two concepts. The Internet is used as a corpus for estimating the length of each edge on the shortest path. Previous methods usually employ stand alone corpora to estimate the edge lengths, and their performances are limited by the data sparseness or unbalanced knowledge of the employed corpus. Our experimental results show that the Internet can overcome the drawbacks that stand alone corpora brought to semantic similarity measures. We use context words to tag word senses and Normalised Google Distance (NGD) to calculate the length of edges on the shortest path.

The proposed semantic similarity measure has been evaluated by comparing the rating results with human benchmark data. We also compare our results with the results returned by Jiang and Conrath's method, which, according to [2,16], provides one of the best results on measuring semantic relatedness. The experimental results indicate that employing the Internet as a corpus and using NGD to compute the edge lengths enable our method to outperform previous semantic similarity measures.

REFRENCES

[1] S. Banerjee and T. Pedersen, "Extended gloss overlaps as a measure of semantic relatedness," In Proc. of the 18th Intl Joint Conf. on Artif. Intell., pp. 805-810, 2003.

[2] A. Budanitsky and G. Hirst, "Evaluating WordNet-based measures of semantic distance," Comput. Linguist., 32(1), pp.13-47, 2006.

[3] R. Cilibrasi and P. Vitányi "The Google Similarity Distance," IEEE Trans. Knowl. Data Eng., 19(3), pp.370-383, 2007.

[4] C. Fellbaum, WordNet: An Electronic Lexical Database, MIT Press, 1998.

[5] E. Gabrilovich and S. Markovitch, "Computing semantic relatedness using Wikipedia-based explicit semantic analysis," In Proc. of the 20th Intl Joint Conf. on Artif. Intell., pp. 1606-1611, 2007.

[6] G. Hirst and D. St-Onge, "Lexical chains as representations of context for the detection and correction of malapropisms," In WordNet: An Electronic Lexical Database, The MIT Press, 1998, pp. 305–332.

[7] J.J. Jiang and D.W. Conrath, "Semantic similarity based on corpus statistics and lexical taxonomy," In Proc. of the Intl Conf. on Research in Comput. Linguist., pp. 19–33, 1997.

[8] S. Kaza and H. Chen, "Evaluating ontology mapping techniques: An experiment in public safety information sharing," Decis. Support Syst., doi:10.1016/j.dss.2007.12.007, 2008.

[9] P.M., Kruse, A. Naujoks, D. Roesner, and M. Kunze, "Clever Search: A WordNet Based Wrapper for Internet Search Engines," In Proc. of 2nd GermaNet Workshop 2005, arXiv:cs/0501086v1, 2005.

[10] C. Leacock and M. Chodorow, "Combining local context and WordNet similarity for word sense identification," In WordNet: An Electronic Lexical Database, The MIT Press, 1998, pp. 265–283.

[11] M. Li and and P. Vitanyi, An Introduction to Kolmogorov Complexity and Its Applications, Springer-Verlag, New York, 1993.

[12] D. Lin, "An information-theoretic definition of similarity," In Proc. of the Intl Conf. on Mach. Learn., pp. 296–304, 1998.

[13] G.A. Miller and W.G. Charles, "Contextual correlates of semantic similiarity," Lang. Cognitive Proc., 6(1), pp. 1–28, 1991.

[14] J. Morris and G. Hirst, "Lexical cohesion computed by thesaural relations as an indicator of the structure of text," Comput. Linguist., 17(1):21–48, 1991.

[15] R. Navigli, "Word Sense Disambiguation: A Survey," ACM Comput. Surv., 41(2), pp. 1-69, 2009.

[16] G. Pirrò, "A semantic similarity metric combining features and intrinsic information content." Data Knowl. Eng., pp. 1289-1308, 2009.

[17] W.V. Quine, Ontological Relativity and Other Essays, New York, Columbia University Press, 1969.

[18] R. Rada, H. Mili, E. Bicknell, and M. Bletner, "Development and Application of a Metric on Semantic Nets," IEEE Trans. Syst. Sci. Cybern., 19(1), pp. 17-30, 1989.

[19] P. Resnik, "Using information content to evaluate semantic similarity," In Proc. of the 14th Intl Joint Conf. on Artif. Intell., pp. 448–453, 1995.

[20] R. Richardson and A.F. Smeaton, "Using WordNet in a Knowledge-Based Approach to Information Retrieval," Working Paper CA-0395, School of Computer Applications, Dublin City University, Ireland, 1995.

[21] J. L. Rodgers and W. A. Nicewander, "Thirteen ways to look at the correlation coefficient," The American Statistician, 42(1), pp.59–66, 1988.

[22] H. Rubenstein and J. B. Goodenough, "Contextual correlates of synonymy," Comm. of the ACM, 8(10), pp. 627–633, 1965.

[23] C.E. Shannon, "A mathematical theory of communication," Bell Syst. Techn. J., 27(1), pp. 379-423, 1948.

[24] M. Sussna, "Word sense disambiguation for free-text indexing using a massive semantic network," In Proc. of the 2nd Intl Conf. on on Inform. and Knowl. Manage., pp. 67–74, 1993.

[25] P.D. Turney, "Similarity of semantic relations," Comput. Linguist., 32(3), pp.379-416, 2006.

[26] D. Yang and D.M.W. Powers, "Measuring semantic similarity in the taxonomy of WordNet," In Proc. of the 28th Australasian Comp. Sci. Conf., pp.315–322, 2005.

Semantic Enabled Sensor Network Design

Jing Sun[1], Hai H. Wang[2] and Hui Gu[1]
[1]Department of Computer Science, The University of Auckland, New Zealand
j.sun@cs.auckland.ac.nz, hgu005@aucklanduni.ac.nz
[2]School of Engineering and Science, Aston University, United Kingdom
h.wang10@aston.ac.uk

Abstract

Wireless Sensor Network (WSN) systems have become more and more popular in our modern life. They have been widely used in many areas, such as smart homes/buildings, context-aware devices, military applications, etc. Despite the increasing usage, there is a lack of formal description and automated verification for WSN system design. In this paper, we present an approach to support the rigorous verification of WSN modeling using the Semantic Web technology. We use Web Ontology Language (OWL) and Semantic Web Rule Language (SWRL) to define a meta-ontology for the modeling of WSN systems. Furthermore, we apply ontology reasoners to perform automated verification on customized WSN models and their instances. We demonstrate and evaluate our approach through a Light Control System (LCS) as the case study.

1. Introduction

Wireless Sensor Network (WSN) systems have become more and more popular in our daily life [6]. It can be found in many areas, such as industrial plants, residential buildings, smart portable devices, as well as military fields, etc. For example, in a chemical plant, sensors are used to monitor the leaking of toxic materials. In a smart building, sensors are used to detect the motions inside secured facilities. However, the cost of developing a WSN is relatively expensive. Once the sensors have been deployed, it is very hard to change and reconfigure them. Therefore, a good initial design prior to the implementation is essential, which can make the system more reliable and fault tolerant.

With regard to initial designs, there are many approaches that can be used to model WSN systems. For instance, the Unified Modeling Language (UML) was used to specify the WSN design [9]. Despite the semi-formality of UML, the weak ability in modeling WSN instances and the lack of support for automated verification hinder the approach. Recently, Dong et. al. [1] proposed a process algebra based notation, i.e., Active Sensor Process (ASP) for the modeling of WSN systems. However, there has been no automated verification support to facilitate the promised reasoning. On the other hand, there are mature formal techniques from the Semantic Web community that can be effectively applied to this domain. Recently, Web Ontology Language (OWL) has been used to model the knowledge base of WSN and semantic technologies has been widely accepted as important components of complex, cross-jurisdictional, heterogeneous, dynamic sensor network systems [3, 8, 4]. For example, OntoSensor [8] is one of such approaches, where a knowledge base for WSN was developed using OWL. As a forerunner, OntoSensor provides a comprehensive model for WSN system specifications. It covers most of the concepts and properties that can be used in a WSN design. However, one weakness of the approach lies in that it has not fully utilized the reasoning functionalities of the ontology models. Hence, OntoSensor lacks of the capability of analyzing and verifying the correctness of its WSN specifications. As a matter of fact, ontology reasoning could greatly assist the users in creating high quality WSN models as well as further exploring the hidden knowledge, checking the correctness and minimizing redundancies.

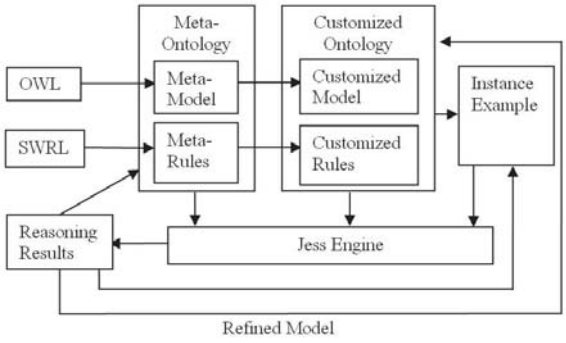


Figure 1. Overall approach to WSN design.

In this research, we aim at fully utilizing the reason-

ing functionalities of the ontology approach in verifying the correctness of WSN design and managing WSN models. We use rule-based notations, i.e., Semantic Web Rule Language (SWRL) [5] to capture complicated constraints among different concepts of a WSN system. Moreover, automated verification can be carried out to check whether these constraints are correctly enforced by the design, in terms of the WSN model and its specific configurations (instances). The users can apply generalized rules of WSN systems that are encoded in the meta-ontology, as well as they can define new rules for a customized WSN system by extending the meta-model. Figure 1 presents an overview of our ontology-based approach. Firstly, a WSN meta-model is defined using the OWL/SWRL languages. The meta-model is designed specially for the WSN context, which includes most of the conceptual terminologies that can be used in a WSN system design. In the meta-model, we also defined additional constraints among different concepts using SWRL rules, which capture complicated relationships among the entities. The meta-model plus the rules form the WSN meta-ontology that can be used to create customized WSN system specifications. When users declare their own customized WSN specifications by extending the meta-model, all the meta-level rules are automatically enforced upon the customized model. The customized WSN ontology also consists of model specific terminologies and the rules. For example, we can customize the meta-ontology into defining a specific light control sensor network system of a smart building. The terms used in the light control system extend the meta-ontology, e.g., specific motion sensors and light devices. The rules used in the customized model further constrain the requirements of the specific system, such as the conditions for a light to be turned on, the light illumination in a room in relation to the sunlight intensity readings of the out-door sensor and so on. After defining the customized WSN ontology, we can create instance configurations from the ontology, e.g., specific numbers of offices and lights in a light control system configuration. At this stage, we can perform reasoning tasks on the ontology and its instances to check the correctness of design. The Jess [2] is a rule engine that can be used for such a purpose.

The rest of the paper is organized as follows. Section 2 presents the modeling and reasoning of the WSN meta-ontology. Section 3 presents a Light Control System (LCS) as a case study to demonstrate the usage of the WSN meta-ontology and the reasoning on customized WSN designs. Section 4 concludes the paper and discusses the future work.

2. Ontological Meta-model for WSN Systems

A WSN structure can be divided into three levels, i.e., node level, group level and network level. Our model concentrates on the group level, which means the ontology is built based on how sensors are managed into different groups to provide specific functions and how they interact with each other. We can view a WSN system as a service orientated network, where each group provides services to other groups as well as works collaboratively together. A group consists of two different types of sensors, i.e., leader sensors and normal sensors. A leader sensor acts as the leader of the group that controls other sensors.

$$
\begin{array}{ll}
Group \sqsubseteq \top & Sensor \sqsubseteq \top \\
Service \sqsubseteq \top & \\
LeaderSensor \sqsubseteq Sensor & NormalSensor \sqsubseteq Sensor \\
Group \sqsubseteq \forall\, hasId.int & Group \sqsubseteq = 1\, hasId \\
Group \sqsubseteq \forall\, hasLeaderSensor.LeaderSensor & \\
Group \sqsubseteq \exists\, hasLeaderSensor.LeaderSensor & \\
Group \sqsubseteq \forall\, hasNormalSensor.NormalSensor & \\
Group \sqsubseteq \exists\, hasNormalSensor.NormalSensor & \\
Group \sqsubseteq \forall\, hasService.Service & \\
Group \sqsubseteq \exists\, hasService.Service &
\end{array}
$$

As defined above, a group has an *id* that is used to uniquely identify itself. Each group consists of `Leader Sensors`, `Normal Sensors` and `Services`. We do not restrict the number of services that a group can provide, as a group may provide more than one services to others. For example, a sensor can be used to detect both temperate and altitude at the same time, thus the group that the sensor belongs to can provide two kinds of services simultaneously. Note that the '$\forall$' and '$\exists$' combinations in the above definition represent closure axioms on the entities through their corresponding properties. They are useful in performing automatic classification on the ontology with the Open World Assumption (OWA) of DL. For example, the above states that a group must have a non-empty set of lead sensors through the `hasLeadSensor` property, and the only things that a group can related to through the `hasLeadSensor` property are lead sensors. Similar closure axioms can be defined on the normal sensor and service.

Sensors are used to receive and send data, which are formed in groups to accomplish specific tasks in a WSN system. Our WSN ontology model focuses on the group level of WSN design. For example, the distance of a signal can travel is directly proportional to the power consumption of a sensor node. Therefore, a signal that travels 10 meters in distance consumes more energy than that of the same signal propagates through a neighboring sensor with two 5 meters distance. Thus the cooperation among the sensors in a group is a typical behavior in WSN systems. Sensor grouping can be divided into two main categories, i.e., location groups and logical groups. A location group simply indicates that all the sensors in the group are organized by their physical positions, which can be further divided it into three different types, i.e., a planar mesh group, a geographic group and a spanning tree group. We use the planar mesh groups as examples to demonstrate the modeling. The geo-

graphic and spanning tree groups can be modeled in a similar way.

The planar mesh group is a commonly used structure in WSN systems, where there is no centralized communication and the sensors in the group only sends signals to its nearest neighboring sensors. The structure forms a planar mesh network. This is known as an ad-hoc network where each sensor is willing to propagate the information. One advantage of such a network is its fault tolerance, where broken nodes should not effect the functionalities of the system as a whole, since the data information can always find an alternative path to eventually reach its destination. A planar mesh network is typically deployed into harsh environments. For example, a planar mesh WSN group can be applied in a forest surrounding to collect the soil and weather information. As the sensors need to withstand all sorts of weather conditions, such as rain, wind, frost, snow, etc., fault tolerance is essential. The planar mesh location group can be defined in OWL as follows.

$$
\begin{aligned}
&\mathit{PlanarMeshGroup} \sqsubseteq \mathit{LocationGroup}\\
&\mathit{PlanarMeshGroup} = 1\,\mathit{hasStartingSensor}\\
&\mathit{PlanarMeshGroup} \sqsubseteq \forall\,\mathit{hasStartingSensor}.\\
&\qquad\qquad \mathit{PlanarMeshLeaderSensor}\\
&\mathit{PlanarMeshLeaderSensor} \sqsubseteq \mathit{LeadSensor}\\
&\mathit{PlanarMeshLeaderSensor} \sqsubseteq \forall\,\mathit{hasNeighbour}.\\
&\qquad\qquad \mathit{PlanarMeshNormalSensor}\\
&\mathit{PlanarMeshLeaderSensor} \sqsubseteq \exists\,\mathit{hasNeighbour}.\\
&\qquad\qquad \mathit{PlanarMeshNormalSensor}\\
&\mathit{PlanarMeshLeaderSensor} \sqsubseteq \geq 1\,\mathit{hasNeighbour}.\\
&\qquad\qquad \mathit{PlanarMeshNormalSensor}\\
&\mathit{PlanarMeshNormalSensor} \sqsubseteq \mathit{NormalSensor}
\end{aligned}
$$

The leader sensor still acts as the coordinator of the group. It has at least one normal sensor as its neighbor and manages other normal sensors in the group. A normal sensor in a planar mesh group can have more than one neighboring sensors (leader or normal). The size of the network can be quite large depending on the number of sensors in the group. The group only needs to know the location of the leader sensor, as all other sensors belongs to the group can eventually be reached from it.

A logical group represents a group of sensors that are organized by certain business logic, e.g., temperature, pressure, humidity, altitude, etc. Therefore, strictly speaking, a location group can be considered as one type of logical group, where the physical position is its logic. Logical groups can be defined similarly as the location group. In addition, a logical group can be used in conjunction with other logical groups or location groups. For example, a group can extend both a geographic group and a humidity group in its hierarchy, which means that the group is used for humidity measuring purpose and its sensors are managed through the geographic structure. For interested readers, the complete WSN meta-ontology definition can be found online at: `http://www.cs.auckland.ac.nz/~jingsun/WSN/SensorNetwork_3_4.owl`

One of the distinguishing features of logic based ontology languages is that it has good reasoning support to perform desired verification. Designed as a web ontology language, the full-automatic, high efficiency and large scalability are the most important requirements for OWL and SWRL reasoning services. Ontology reasoning can be used to verify the WSN design at both the conceptual level and the instance level. It can assist users to reduce the amount of work needed in building a good WSN specification. For example, we can define a WSN model partially and use ontology reasoners to infer the rest of the knowledge about the model. Moreover, ontology reasoning can be used to check constraints (relationships) among different instances. Such additional constraints can be captured by the SWRL rules in the ontology and enforced upon the instances by reasoning. In general, the use of OWL reasoners for WSN systems includes but is not limited to the following.

Composition – is a fundamental advantage of using ontology reasoners. OWL provides the ability to specify sets of *sufficient and necessary* conditions to recognize members of a class and infer that one class must be a subclass of another. Classes for which there is at least one set of sufficient conditions are known as "defined" classes. (The symbol $\equiv$ is used in the DL syntax to denote defined classes.) By using the necessary and sufficient conditions defined in OWL, new concepts can be defined systematically from existing concepts. This can result in a dramatic reduction in the number of facts that have to be maintained explicitly. In OWL, complex expressions can be built up using restrictions and logical operators without having to name each intermediate concept before use. This embedding of 'anonymous concepts' allows more expressive representations to be built easily and naturally. For example, we might define *SensoriaPressureGeographicGroup* as pressure sensor groups which are organized geographically and led by a sensor produced by *Sensoria*.

$$
\begin{aligned}
&\mathit{SensoriaPressureGeographicGroup} \equiv\\
&\quad \mathit{PressureGroup} \sqcap \mathit{GeographicGroup} \sqcap\\
&\qquad \forall\,\mathit{hasCenterSensor}.(\mathit{Sensor} \sqcap\\
&\qquad\quad \exists\,\mathit{hasManufacturer}.\mathit{Sensoria})
\end{aligned}
$$

Poly-hierarchy – represents the structure of a knowledge base. As we mentioned before, a sensor or sensor group can be organized into different classes based on its properties and usages. Managing and maintaining such complex poly-hierarchy of concepts is hard. Changes often need to be made in several different places, and keeping everything in step is error prone and laborious. For example, the property values of a sensor in WSN might change over time and its network system has to be reconfigured dynamically. Using a classifier, we can develop normalized knowledge models. In a normalized ontology, even the most complex poly-hierarchies are built out of simple non-overlapping trees. The OWL class definitions are used to define concepts bringing the trees. The reasoners can maintain the relationships amongst defined concepts auto-

matically. Changes are always made in exactly one place. The new poly-hierarchy relationship can be computed on the fly. For example, if we set up a new type of pressure geographical sensor group using *WINSNG*2, which is one kind of sensor produced by *Sensoria*. This can be modeled as the following.

$$WINSNG2 \sqcap Sensor \sqcap \exists\, hasManufacturer.Sensoria$$
$$TestSensorGroup \equiv PressureGroup \sqcap GeographicGroup \sqcap \forall\, hasCenterSensor.WINSNG2$$

Note that there is no need to assert *TestSensorGroup* is a subclass of *SensoriaPressureGeographicGroup*. OWL reasoners will automatically infer this relationship, as the sufficient and necessary condition of *SensoriaPressureGeographicGroup* has been satisfied. If later we update the leading sensors of *TestSensorGroup* to *MPS-D3*, which is a product of *SEBA*, we only need to modify the class *TestSensorGroup*. The reasoners will update the class hierarchy automatically, i.e., removing the subclass relationship between *TestSensorGroup* and *SensoriaPressureGeographicGroup*.

Instantiation – is the process of matching instances against classes. By using instantiation, we can define a general individual, and the reasoning can classify which specific concept it belongs to. The following example illustrates how it works. For example, if we would like to further restrain that a lead sensor can only control normal sensors, not other lead sensors. We specify the following SWRL rule.

$$Sensor(?y) \wedge controlSensor(?x, ?y) \wedge LeaderSensor(?x) \rightarrow NormalSensor(?y)$$

`Sensor(?y)` and `LeaderSensor(?x)` define the types of individuals `x` and `y`. Note that we define `y` as a generalized sensor instead of a specific type of sensor. The `controlSensor(?x, ?y)` indicates the relationship between `x` and `y`. `NormalSensor(?y)` is the result of this rule. The whole rule can be understood as '*if y is a sensor and it is controlled by a leader sensor x, then y must be a normal sensor*'.

Property inference – is used to derive the existence of a certain property in the ontology. With inference, we can build a partial WSN model, and use reasoning to derive the rest of the knowledge. For example, the following SWRL rule captures that '*if a group x has a leader sensor y, y controls z, then z must belongs to group x*'.

$$Group(?x) \wedge LeaderSensor(?y) \wedge NormalSensor(?z) \wedge hasLeaderSensor(?x, ?y) \wedge controlSensor(?y, ?z) \rightarrow hasNormalSensor(?x, ?z)$$

From the above, we specified three entities, i.e., a `Group x`, a `LeaderSensor y` and a `NormalSensor z`. If there is a relationship `hasleaderSensor` between `x` and `y` and another relationship `controlSensor` between `y` and `z`, we can infer that there must be a relationship `hasNormalSensor` exists between `x` and `z`. Similarly, instances can be checked in Protégé through the Jess engine. The above rule illustrates the inference property of ontology reasoning. The user only needs to define the relationships from a group to a leader sensor and from a leader sensor to a normal sensor. The rule can infer the relationship between the group and the normal sensor automatically.

Consistency Checking – is the process of detecting violation of constraints (inconsistency) in a model. After we define the WSN ontology, we can create specific WSN instances (configurations) and use reasoning to check whether the instances are consistent with respect to the model. For example, the following rule captures that a normal sensor can only be controlled by at most one lead sensor.

$$LeaderSensor(?x) \wedge NormalSensor(?y) \wedge controlSensor(?x, ?y) \rightarrow controlledBySensor(?y, ?x)$$

The rule states if a `LeaderSensor x` control a `NormalSensor y`, then `y` must be only controlled by `x`, not any other lead sensors. If we have an instance that states `y` also controlled by another leader sensor z, then there is an inconsistency (conflict) in the model. Similarly, this can be checked in Protégé through Jess.

The above examples illustrate different types of reasoning that can be performed by the WSN meta-ontology model. The most important strength and purpose of logic-based ontology approach is that they allow classes to be defined by complex class expressions built recursively from previously defined classes and properties using constructors provided by the ontology language. In the next section, we demonstrate the modeling and verification of a customized WSN that extends the meta-ontology.

3. Case Study - A Light Control System (LCS)

In this section, we extend the above defined WSN meta-ontology to design a Light Control System (LCS) [7] as a case study of WSN modeling and apply reasoning to enhance the design. LCS is a context-aware system that uses sensors to detect the motions of people and decide whether it should turn on/off the lights inside the facilities. It also has a set of outdoor sensors to detect the intensity of the day light in order to automatically adjust the illuminations of the lights inside the facilities. Detailed LCS requirements can be found in [7]. Due to the space limit, we can not list all of them, as our goal is to use it as an examples to demonstrate the verification.

In the LCS, room, hallway and wall can be defined as subclasses of the Geographic Group, where the diameters of those groups represent the physical space. Our design is based on a building structure of a university facility, where the types of rooms included are computer labs, hardware labs, meeting rooms, offices and peripheral rooms. For each

group, we defined a controller that acts as the leader sensor. There are three types of controllers specified, i.e., room controller, hallway controller and wall controller. Let's consider the room controller as an example.

$$RoomController \sqsubseteq LeaderSensor$$
$$RoomController \sqsubseteq \forall controlElectricalDeviceGroup.LightGroup$$
$$RoomController \sqsubseteq \geq 1\ controlElectricalDeviceGroup$$
$$RoomController \sqsubseteq \forall controlSensor.MotionSensor$$
$$RoomController \sqsubseteq \geq 1\ controlSensor$$
$$LightSensor \sqsubseteq NormalSensor$$
$$LightSensor \sqsubseteq \forall hasMeasurand.LightDetector$$
$$LightSensor \sqsubseteq = 1\ hasMeasurand$$

The above syntax states that a room controller is one type of leader sensor that can control at least one light group and at least one motion detector. After we created the controllers, we need to deploy some sensors into the system. In the LCS, we mainly have two types of sensors, i.e., motion sensor and light sensor. The motion sensor is used to detect if there is any movement in a certain area. The light sensor is installed on the facility wall to detect the intensity of the outdoor day light. Light sensor and motion sensor both extend from a normal sensor. For example, the eighth line constrains that a light sensor can only be used for detecting light intensity; while the ninth line constrains that light sensor can only have one measurand. Other entities of the LCS, such as offices, meeting rooms, hallways, etc., can be defined in a similar way. For interested readers, the complete LCS model can be found online at: `http://www.cs.auckland.ac.nz/~jingsun/WSN/LightSystem_3_4.owl`

In this section, we used SWRL rules to specify some of the LCS functional requirements that were listed in [7] and demonstrate how the ontology reasoning can help us to build a high quality specification. Note that because the LCS model extends the WSN meta-ontology defined in Section 3, the meta-level reasoning we demonstrated in Section 3.3 can be automatically applied on the LCS model. However, here we would like to focus more on examples of the LCS specific reasoning as follows.

"*The illumination value of the lights in a room plus the day light intensity detected by the light sensors outside the room should be equal to a certain value, for instance 100.*"

$$Office(?x) \wedge Wall(?y) \wedge besideWall(?x, ?y) \wedge RoomController(?w) \wedge WallController(?z) \wedge hasLeaderSensor(?x, ?w) \wedge hasLeaderSensor(?y, ?z) \wedge LightGroup(?c) \wedge LightSensor(?a) \wedge Light(?d) \wedge controlElectricalDeviceGroup(?w, ?c) \wedge controlSensor(?z, ?a) \wedge LightDetector(?b) \wedge hasMeasurand(?a, ?b) \wedge detectLightIntensity(?b, ?e) \wedge deviceGroupHasSomeDevice(?c, ?d) \wedge swrlb : subtract(?f, 100, ?e) \rightarrow lightHasIntensity(?d, ?f)$$

In the above, we demonstrate a SWRL representation of the illumination requirement in LCS. It simply specifies that if an outdoor sensor detects the daylight intensity value, say 20, then the lights in the office should have an illumination value of 100-20 = 80 (if we assume that the safe illumination value is 100). Note that the `swrlb:subtract(?f, 100, ?e)` is a built-in mathematic rule in SWRL that expresses 'f = 100 - e' [5]. With this constraint enforced, it can provide two kinds of reasoning, i.e., inferring new knowledge and checking inconsistency. Firstly, if we knew the outdoor daylight intensity reading, we can automatically derive the light illumination value in the room. Secondly, if a user wrongly specified a value of the light illumination inside a room with respect to its outdoor daylight intensity reading, the reasoning can easily detect such inconsistent instances.

"*When a room/hallway is occupied by a person, the lights inside the facility must be kept on.*"

$$MotionDetector(?x) \wedge MotionSensor(?y) \wedge hasMeasurand(?y, ?x) \wedge controlSensor(?z, ?y) \wedge controlElectricalDeviceGroup(?z, ?u) \wedge deviceGroupHasSomeDevice(?u, ?v) \wedge detectMotion(?x, true) \rightarrow isDeviceTurnedOn(?v, true)$$

The above rule states that a room controller `z` controls both the motion sensor `y` and light group `u`. If the motion sensor detects some movement from the motion detector `x`, then the lights `v` that belong to the light group `u` should be turned on. Again, this rule enforces the availability of lights according to the motion detectors in the facility and also checks (maintains) the correct status of the lights in the system. Other LCS requirements in [7], e.g., '*if a room/hallway is unoccupied for a certain time the lights in the facility need be switched off automatically*', etc., can be modeled and verified using the Jess rule engine in a similar manner. In the following section, we demonstrate another type of reasoning that can be used to monitor the behavior patterns of a WSN system.

Besides the reasoning examples demonstrated above, SWRL rules can also be used to determine dynamic actions in a WSN system. For example, we define the following two SWRL rules in the LCS.

Rule-1 :
$$MotionDetector(?x) \wedge MotionSensor(?y) \wedge hasMeasurand(?y, ?x) \wedge controlSensor(?z, ?y) \wedge controlElectricalDeviceGroup(?z, ?u) \wedge deviceGroupHasSomeDevice(?u, ?v) \wedge detectMotion(?x, true) \rightarrow isDeviceTurnedOn(?v, true)$$

Rule-2 :
$$MotionDetector(?x) \wedge MotionSensor(?y) \wedge hasMeasurand(?y, ?x) \wedge controlSensor(?z, ?y) \wedge$$

$controlElectricalDeviceGroup(?z, ?u) \wedge$
$deviceGroupHasSomeDevice(?u, ?v) \wedge$
$detectMotion(?x, false) \wedge detectNothing(?x, 10)$
$\rightarrow isDeviceTurnedOn(?v, false)$

Rule-1 states if a room or hallway is occupied by someone, the lights in that room or hallway should be turned on. And rule-2 is the reverse case of rule-1, which states if a hallway/room is not occupied for more than 10 seconds, the lights belong to that hallway/room should be turned off. Now, if we have two hallways connecting to each other, namely hallway a and b, as show in Figure 2.

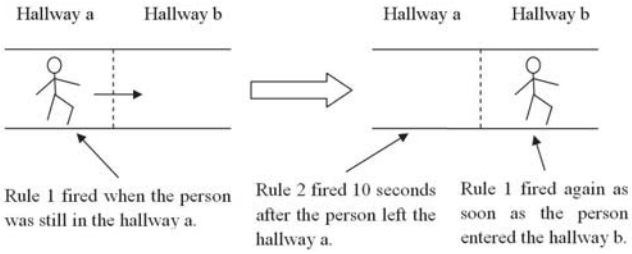


Figure 2. Movement from hallways a into b.

If a person goes into hallway a, it will trigger Rule-1 to be fired. We can record as ⟨Rule-1, a⟩. Suppose this person moves from hallway a into hallway b, the sensor installed in hallway b detect this person and Rule-1 is fired again. We record as ⟨Rule-1, b⟩. If there is no one else in hallway a at that moment, after 10 seconds, Rule-2 will be fired. We can record ⟨Rule-2, a⟩. Finally, we can write the rule invocation sequence for this person (instance) as: ⟨Rule-1, a⟩ → ⟨Rule-1, b⟩ → ⟨Rule-2, a⟩. Now, suppose if this sequence has occurred for the same instance and no other direct connected sections are involved during the same period, we can conclude that there must have been a person moved from hallway a to hallway b in that duration period. Hence, we can define a set of control sequences in the WSN system (in the forms of rule actions), where each sequence represents a behavior pattern that could be engaged by the individuals (instances) in the system. By monitoring these sequences (patterns) occurred in the system, we can come to aware of the basic activities happened during a certain period of time. Moreover, these behavior pattern recognition results can be used to make further context-aware decisions in the WSN system, e.g., for security or patient monitoring purposes.

4. Conclusion

In this paper, we proposed a formal ontological approach for WSN modeling and verification. We defined an WSN meta-ontology in the OWL/SWRL language, which specifies the basic terminologies and relationships that captures the fundamental requirements of a WSN design. These rules can provide subsumption, inference and consistency reasoning on the WSN instances. When users design a WSN model using our approach, they can create customized WSN specifications by extending the top level meta-ontology. In addition, they could initialize parts of the specification and use the ontology reasoner to automatically infer the rest of the knowledge in the design. Furthermore, the rules and their invocation sequences in our approach can be effectively used as action patterns in monitoring the dynamic behaviors of a WSN system implementation. Finally, we demonstrated the design and verification of the Light Control System (LCS) as a case study.

In the future, we plan to develop a Graphical User Interface (GUI) that aims at assisting a user-friendly creation and verification of WSN designs. The purpose of the GUI is to allow the designers to build their WSN models in transparent of the underlying OWL/SWRL syntaxes.

References

[1] J. S. Dong, J. Sun, J. Sun, K. Taguchi, and X. Zhang. Specifying and Verifying Sensor Networks: an Experiment of Formal Methods. In *Proceedings of the 10th International Conference on Formal Engineering Methods*, pages 318–337. Springer-Verlag, October 2008.

[2] E. Friedman-Hill. *Jess in Action: Java Rule-Based Systems*. Manning Publications Co., Greenwich, CT, USA, 2003.

[3] M. Gomez, A. Preece, and G. de Mel. Towards Semantic Match Making in Sensor-Mission Assignment: Analysis of the Missions and Means Framework. Technical report, ITA Technical Paper, 2007.

[4] J. Goodwin and D. Russomanno. An Ontology-Based Sensor Network Prototype Environment. In *Proceedings of the Fifth International Conference on Information Processing in Sensor Networks*, pages 1–2, April 2006.

[5] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. `http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/`, May 2004.

[6] F. L. Lewis. *Chapter 2 in D. J. Cook and S. K. Das, editors, Smart Environments: Technologies, Protocols, and Applications*, pages 11–46. John Wiley and Sons, New York, 2005.

[7] S. Queins, G. Zimmermann, M. Becker, M. Kronenburg, C. Peper, R. Merz, and J. Schaefer. The Light Control Case Study: Problem Description. *Journal of Universal Computer Science*, 6(7):586–596, 2000. `http://www.jucs.org/jucs_6_7/light_control_problem_description`.

[8] D. Russomanno, C. Kothari, and O. Thomas. Sensor Ontologies: from Shallow to Deep Models. *Southeastern Symposium on System Theory*, 0:107–112, 2005.

[9] G. Sunyé, A. L. Guennec, and J.-M. Jézéquel. Design Patterns Application in UML. In *ECOOP '00: Proceedings of the 14th European Conference on Object-Oriented Programming*, pages 44–62, London, UK, 2000. Springer-Verlag.

Using Semantic Annotations for Supporting Requirements Evolution

Bruno Nandolpho Machado, Lucas de Oliveira Arantes, Ricardo de Almeida Falbo

Ontology and Conceptual Modeling Research Group (NEMO),
Computer Science Department, Federal University of Espírito Santo, Vitória, Brazil
{brunonandolpho, lucasdeoliveira}@gmail.com, falbo@inf.ufes.br

***Abstract* — Requirements change for a variety of reasons and at different stages of the project development. Changes to items in a requirements document must be propagated to other items that depend on the changed items in order to maintain their consistency. This paper explores the use of semantic annotations in requirements document templates to support requirements evolution.**

Keywords – Requirements evolution; requirements documentation; semantic documentation.

I. INTRODUCTION

The Requirements Engineering (RE) process plays a key role to ensure that software products will fully support and evolve with the business processes. It is the process by which requirements are gathered, analyzed, documented, and managed throughout the software lifecycle [1]. During the RE process, elicited requirements need to be precisely specified and documented. Moreover, requirements change for a variety of reasons. Such changes may be required at various stages of the software lifecycle, and must be propagated through other items that depend on the changed items. In order to maintain their mutual consistency, we need to manage traceability links among items and to propagate changes along such links [2].

Requirements are usually recorded in one or more documents, which are used to communicate requirements to different stakeholders. There are many different ways to structure requirements documents, depending on, among others, the type of the system being developed, the target audience, the level of detail to be considered, and organizational practices. To ensure that the essential information is included in each document, organizations should define their own standards for requirements documents. If an organization works with different types of requirements documents, it should define an appropriate template for each requirements document type [3, 4].

Despite its shortcomings, structured natural languages, augmented with graphical models, remains the most practical way for most software organizations to document their requirements [4]. Moreover, despite the current advances in electronic documentation along with the boom of collaborative text edition tools (such as wiki engines), desktop text editors are still the most frequently solution used by software organizations when it comes to electronic documentation [5,6]. Whether on the use of wiki engines or the use of desktop text editors, documents produced by these tools are still the main vehicle for knowledge dissemination [5,7,8]. This is the case of software engineering in general, and RE in particular.

Requirements documents hold a considerable amount of information that are to be mainly interpreted by human readers, such as requirements statements, use case descriptions, and so on. Managing requirements evolution requires reading different versions of different documents, in a task that is often dull and error prone. In addition, gathering relevant information contained in different documents spread through the organization's repositories demands a considerable effort and, because of that, this activity is often skipped [5].

Requirements traceability can be more easily achieved if the semantic content of the requirements documents could be exposed in order to allow visibility of the data and the relationships embedded in the document, and if the semantic content of each document version is extracted and registered into a version control system. In order to make these scenarios possible, it is essential to allow semantic metadata annotation into documents, turning requirements documents into semantic requirements documents.

For dealing with theses problems, Arantes and Falbo [9] developed an Infrastructure for Semantic Document Management (ISDM) [9] that presents the following features: semantic annotation of document templates; traceability support; searching based on extracted semantic content; and change notification subscription.

This paper discusses how this infrastructure is used to support requirements evolution, and is organized as follows: Section II talks briefly about requirements evolution, and semantic documentation; Section III shortly presents the ISDM and its main components; Section IV addresses the use of ISDM in the requirements management context, and presents some preliminary results from using ISDM in practical situations. Section V compares our work with some related ones. Finally, Section VI presents our conclusions.

II. REQUIREMENTS EVOLUTION AND SEMANTIC DOCUMENTATION

Two important activities of the RE process are requirements documentation and evolution. The results of requirements development should be documented for latter

agreement between customers and developers about the software to be built, and to serve as basis for further development and evolution.

As said before, requirements can be recorded in one or more documents, each one devoted to different classes of stakeholders. Pfleeger and Atlee [10], for instance, suggest the use of two different types of documents: the requirements definition, which is written in terms of the customer's vocabulary, and the requirements specification, which is written in terms of the system's interface.

There are many different ways to structure these documents. Ideally software development organizations should define templates for different types of requirements documents, imposing a standard structure on them [2,4].

With regard to the requirements definition document, it should specify the agreed requirements statements in a language that supports communication with stakeholders. The most obvious option is to document these statements using free prose in natural language. However, this approach is prone to several defect types, such as ambiguity. To overcome these problems, we may adopt a disciplined approach for documentation, using structured natural language. In such approach, stylistic rules on how statements should be written, and predefined statement templates, among others, are used to discipline writing the statements [2].

The agreed statements posed in a requirements definition document are subject to change. In order to maintain consistency, these changes must be propagated through other items that depend on the changed item. Since evolution is inevitable, we must prepare for change, and traceability is one of the most important ingredients for this. Consistency maintenance requires managing traceability links among items. However, traceability management is not an easy task. To take full advantage of its benefits, we need to reduce the complexity and cost of establishing and maintaining the traceability graphs [2].

A way of addressing this issue is applying a semantic documentation approach to requirements documentation. Allowing users to add metadata annotations to documents can improve the understanding and accessibility of the data contained on it. Features such as document annotation, data extraction from metadata, and data indexing and searching, are somehow the basis for semantic documentation [7,8,9].

III. An Infrastructure for Semantic Document Management

In essence, the Infrastructure for Semantic Document Management (ISDM) provides: (i) a way to semantically annotate document templates; (ii) a mechanism for controlling versions of the semantic content extracted from semantic document versions, and therefore providing a way for tracking the evolution of the data embedded inside a semantic document; and (iii) data visibility to end-users, allowing searches and data-change notification subscription, to aid developers to get an up-to-date information about something he/she is interested in [9].

As shown in Figure 1, the ISDM architecture is composed by two main elements [9]: the Semantic Document Repository (SDR), which is responsible for storing semantic documents; and the Main Module, which is composed by three sub-modules: (i) the Semantic Annotation Module is responsible for allowing users to semantically enrich a document template; (ii) the Data Extraction and Versioning Module is responsible for extracting the semantic content from an annotated document whenever a new version of that document is checked into the SDR. After extraction, the semantic content of the version is stored in another repository, called Data Repository, that is also part of this module; (iii) the Search and Traceability Interface Module is responsible for providing an API (Application Programming Interface) that allows users and other systems to perform ontology-based searches and data traceability towards the Data Repository.

In a nutshell, document engineers annotate document templates using the Semantic Annotation Module. Later on, developers and analysts may instantiate that template, generating semantic documents. These semantic documents are checked into the SDR. When a new version of a semantic document is available at the SDR, the Data Extraction and Versioning Module unwraps the semantic content of that version and stores it into the Data Repository, making the information of that version available. At this point, it is possible to further enhance the integration of information scattered throughout various semantic documents contained in the SDR. This is done by merging the graphs corresponding to the last versions of the documents contained in the repository. Finally, users and other systems may interact with the Search and Traceability Interface Module in order to perform queries about the evolution of a particular semantic content stored in the Data Repository.

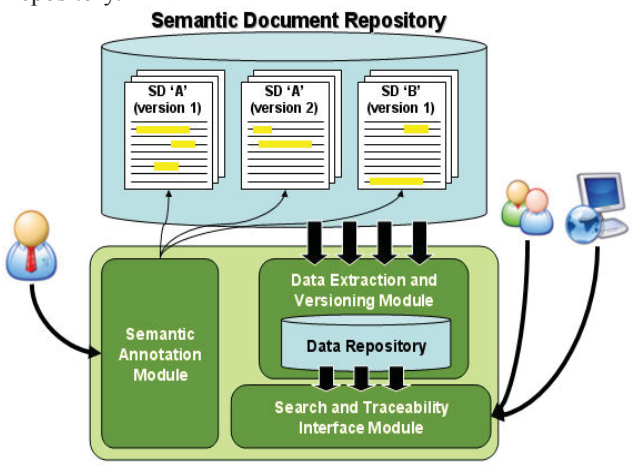


Figure 1. ISDM Architecture.

For annotating document templates, domain ontology-based annotations are used. ISDM works with templates and

documents written in Open Document Format (ODF) [11], and Open Office [12] was chosen as the document editor for composing annotations in document templates. In order to allow template annotation, specialized instructions for annotating text fragments and tables were developed. These kinds of annotations encapsulate annotations directed for document instances, allowing document engineers to add a set of instructions directly in these document elements (tables and text fragments). These instructions are processed when a document instance is analyzed by the Data Extraction and Versioning Module that will ultimately generate instances and relations accordingly. For more details, see [9].

IV. SUPPORTING REQUIREMENT EVOLUTION WITH SEMANTIC ANNOTATIONS

During the software life cycle, changes in requirements are quite common. New requirement dependencies are discovered, requirements statements are rewritten and priorities change. If a change is propose in a requirement, we need to identify how this change affects others requirements. In this scenario, the traceability matrix is an important artifact that is used to analyze the impact of changing requirements. Defining a network of requirement dependencies is the starting point for creating the requirements traceability matrix. Moreover, the use of a requirements management tool can support developing traceability matrices.

To deal with this, we are currently using the ISDM for supporting some tasks of the RE process at NEMO (Ontology & Conceptual Modeling Research Group). In NEMO, we use two types of documents for documenting requirements: a Requirements Document (RD) and a Requirements Specification (RS). The first is directed to clients and users, and captures user requirements. It is written in natural language, following rules defined for writing requirements statements. The second details the user requirements into systems requirements, and serves as basis for further development. It is mainly composed by models (use case diagrams, class diagrams, state diagrams, among others), although there are also some textual parts, especially the ones related to use case descriptions. Due to space limitations, in this paper we only discuss how we are using semantic annotations in RDs.

A. Annotating the Requirements Document Template

The NEMO's RDs are composed by some preliminaries, and four sections. For elaborating a RD, developers should follow the RD template shown in Figure 2.

First of all, there are two important information placed in text fragments: the name of the project, and the names of the responsible for the document. For capturing them, the ISDM provides annotations for annotating text fragments. Following we present the text fragment annotations added to the RD template. These annotations are done based on the requirements ontology proposed in [13].

Requirements Document

Project : <<project name>>
Responsible: <<names of the responsible analysts, separated by commas>>

1. Introduction

This document presents the user requirements of the <<system name>>. It is organized as follows: Section 2 describes the system purpose; Section 3 presents a description of the problem domain; Section 4 presents the user requirements elicited from clients and users.

2. System Purpose

<<one paragraph describing the system purpose>>.

3. Domain Description

<<a free text briefly giving an overview of the domain, describing the problem to be solved and business processes to be supported>>

4. User Requirements

Functional Requirements

Id	Statement	Priority	Depends on
FRXX	<<sentence following defined pattern>>	<<possible values: High, Medium, Low>>	<<ids of the requirements on which the requirement depends>>

Business Rules

Id	Statement	Priority	Depends on
BRXX	<<idem FR>>	<<idem FR>>	<<idem FR>>

Non Functional Requirements

Id	Statement	Priority	Category	Depends on
NFRXX	<<idem FR>>	<<idem FR>>	<<type of the NFR>>	<<idem FR>>

Figure 2. Requirements Document Template

[[completeText]];instance({content},http://localhost/ontologies/SE/onto.owl# Project,$project);

[[break with ',']];
instance({slice},http://localhost/ontologies/SE/onto.owl#Person, $person);

property($person,http://localhost/ontologies/SE/onto.owl#involvedIn,$project);

The first annotation is added in the place marked with the <<project name>> tag; the second is added in the place marked with the <<names of the responsible analysts, separated by commas>> tag. The third annotation is done based on the requirements ontology, and says that the people informed as responsible analysts are involved in the project. Other tags shown in text fragments, such as the one related to the system purpose, are annotated in a similar way.

ISDM also provides annotations for annotating tables, which were used in the RD template for annotating the Domain Description table, the Functional Requirements (FR) table, the Business Rules (BR) table and the Non Functional Requirements (NFR) table. In the case of FR table, as shown in Figure 2, the first column refers to the FR id; the second to the FR statement, the third to its priority, and the last one to a list of the ids of the requirements (FRs, BRs and NFRs) on which it depends, separated by comma. Annotations in tables allow that each column has a different set of instructions.

The following annotations were added to the FR table in the semantic template:

[[ignorerow0]];

[[at0]];instance({content},http://localhost/ontologies/SE/onto.owl#FunctionalRequirement,#req);
property(#req,http://localhost/ontologies/SE/onto.owl#artifactProducedIn,#project);

[[at1]];property(#req,http://localhost/ontologies/SE/onto.owl#description,{content});

[[at2]];property(#req,http://localhost/ontologies/SE/onto.owl#priority,{content});

[[at 3 break with ','];instance({slice},http://localhost/onto.owl#Requirement,#reqline);
property(#req,http://localhost/ontologies/SE/onto.owl#relatedWith,#reqline);

The first annotation only says to ignore the first row in the table, since it does not contain data (it is a header). The annotations that begin with *[[at]]* are used for annotating the content from a column. The second annotation indicates that the content of the first column is a FR and that this FR is produced in the project informed before. The third annotation points out that the content of the second column is the statement of the FR captured in the first column. Finally, the last annotation says that the current FR depends on the requirements informed in the fourth column.

The other tables and their annotations are quite similar to the FR table. Thus, making use of the semantically anotated template, it is possible to the Data Extraction and Versioning Module to extract the semantic content present in the requirements document, as explained next..

B. Extracting and Visualizing Information from the RDs

Once the RD template is instantiated and a new version of the RD is produced, it should be committed in the corresponding Semantic Document Repository (SDR) (see Figure 1), in order to the Data Extraction and Versioning Module (DEVM) process the semantic document. DEVM generates instances and relations accordingly to the contents of the document in the Data Repository, in an OWL format.

Whenever a new version of a semantic document is commited in the repository, a new version of the document content is generated in the Data Repository, and it is possible to use the services from the platform.

Figure 3 presents part of the tables concerning functional requirements and business rules, produced in the context of the Cargo Delivery Control Project. The versions of the RDs of this project are stored in the the repository /home/nemo/reposg7. Taking the third verson of this RD into account, it is possible to generate the traceability matrix shown in Figure 4. The full matrix is not presented in this paper due to space limitation. It is worthwhile to point out that as the data is stored in OWL format, we can make some inferences using the JENA engine.

Functional Requirements

Id	Statement	Priority	Depends on
FR01	The system shall control types of cargos.	High	
FR02	The system shall control transportation rates.	High	FR01, FR12, BR03, BR05
FR03	The system shall allow the customer to make a request to transport a cargo from a quotation previously made.	High	FR02, BR02, NFR02, NFR06
…	…	…	…

Business Rules

Id	Statement	Priority	Depends on
BR01	Clients who have hired a service can not be excluded.	High	
BR02	The system must generate a unique identifier for each quotation.	High	
BR03	Each cargo type has a procedure to be followed to be transported.	High	

Figure 3. Part of a Requirements Document.

As the dependency relationship (the last column of the requirements tables) is transitive, we can infer, for instance, that if FR03 depends on FR02 and FR02 depends on FR01, then FR03 depends on FR01. Direct dependencies are shown in the figure with the flag "D"; inferred dependencies, on the other hand, are marked in the table with the letter "I". Such information is very important, since it is useful for analyzing the impact of a change. Inferring requirements dependencies in large projects can be difficult, laborious and error prone to perform manually.

Req	FR01	FR02	FR03	...	BR01	BR02	BR03
FR01				...			
FR02	D			...			D
FR03	I	D		...		D	I
…				...			

Figure 4. Part of the Traceability Matrix.

Another service provided by ISDM is to trace the differences between versions of a document. Using this service, a developer may look for the changes made in a specific document. Figure 5 shows the results from a query for providing the differences between versions 2 and 3 of the RD partially presented in Figure 3. This document was modified, for instance, by changing the statement of FR03 and the requirements on which it depends. Such changes are preceded by "(CHANGED)" in the figure. Moreover, there were three FRs added and one removed, which are represented in the figure by lines beginning with the words "(ADDED)" and "(REMOVED)", respectively. Using this service, we can follow up requirements evolution, as they are listed when they are added, removed or modified in a new version of the RD.

On the other hand, during the execution of the project, a developer may want to know the history of a specific requirement. To this end, the developer can use the service for visualizing the evolution of a requirement, illustrated in Figure 6. As we can see in this figure, *FR04* was added in the Revision 1. The requirements on which it depends were changed in the Revision 2, when two related requirements

were added (*BR03* and *NFR01*). Finally, in revision 3, *FR04* was removed from the RD. With this service, a developer can trace the evolution of a specific requirement, detailing how it was related to other requirements and the values of its properties along the project.

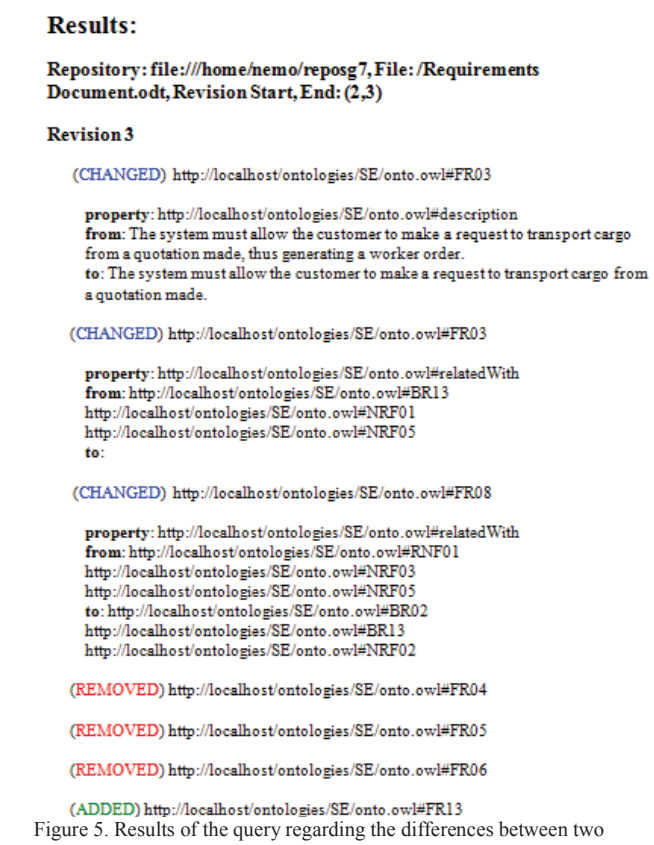


Figure 5. Results of the query regarding the differences between two versions of a RD.

Figure 6- Search Form evolutionary tracing of an individual and results t

C. *A Preliminary Evaluation*

ISDM has been used to trace the requirements of projects performed at NEMO. Up to now, 7 projects used it. In 4 of these projects, the RDs evolved in two versions. In the other 3 projects, there were three versions. The various versions of these documents were added to the Semantic Document Repository (SDR), and each project had its own version control repository. The three tables shown in Figure 7 present data regarding the evolution of the RDs of these projects, including how many requirements were added, changed or removed in each version of each project.

Revision 1

Project	Added	Changed	Removed	Sum
Project 1	29	-	-	29
Project 2	10	-	-	10
Project 3	25	-	-	25
Project 4	29	-	-	29
Project 5	13	-	-	13
Project 6	10	-	-	10
Project 7	24	-	-	24

Revision 2

Project	Added	Changed	Removed	Sum
Project 1	11	16	12	24
Project 2	16	10	-	26
Project 3	3	22	3	25
Project 4	2	27	1	30
Project 5	11	12	-	24
Project 6	14	10	-	24
Project 7	9	17	5	28

Revision 3

Project	Added	Changed	Removed	Sum
Project 4	9	9	4	35
Project 6	13	15	2	35
Project 7	2	11	10	20

Figure 7. Changes of requirements in numbers.

The numbers shown in Figure 7 give an idea of how the use of ISDM for managing requirements has proved to be useful in NEMO. Since many requirements are added, removed and changed, especially at the beginning of the projects, controlling their changes and impacts in other artifacts are essential. The use of ISDM has speeded the impact analysis and, as a consequence, the time spent in performing changes. Before the use of ISDM at NEMO, those analyses were done by humans, using only the functionalities provided by OpenOffice for comparing documents. By showing the changes done and generating traceability matrices, ISDM provided a very useful basis for impact analysis.

V. RELATED WORK

Since evolution is inevitable, there are several works that aim at minimizing evolution efforts. Special attention has being devoted to requirements traceability management.

Rochimah et al. [14] evaluated seven traceability approaches focusing on their contributions to simplify software evolution tasks. Three of them generate traceability

links in an automated way, while other three are semi-automated, in the sense that they combine a manual and automated way in obtaining the traceability links. Regarding the degree of automation, our approach can be classified as semi-automated, since templates are annotated manually and the links between requirements are informed by the analysts when the templates are filled in. On the other hand, the traceability matrices are automatically generated, and the changes done can be queried. Concerning the degree of formality, Rochimah et al. consider that the seven evaluated approaches are semi-formal. This is also the case of our approach, which uses an ontology as basis for the template annotations. Finally, regarding the change type, as the other approaches, ours allows to identify additions, deletions, and modifications. A distinguishing feature of our approach is that analysts work in the same way they have always done, i.e. filling templates in a text editor, and committing them in the project's repository. In this way, our approach is in line with the "support in-place traceability" best practice, defined by Huang et al. [15] as traceability being provided to the artifacts residing within their native environments.

There are several requirements engineering tools, such as TRACE [16], that provide various forms of traceability and support to change. However, at the best of our knowledge, none of them uses a semantic documentation approach, and thus users have to interact with the tool, instead of writing requirements documents in a text editor. In our work, the idea is just to allow the analysts to continue using a desktop text editor and, by means of semantic annotations made in templates, extract and track requirements in a transparent way to the user.

VI. CONCLUSIONS

In this paper we discussed the use of ontology-based annotations in semantic Requirements Document (RD) templates for supporting requirements managing and evolution. Our strategy was to instantiate and to extend the Infrastructure for Semantic Document Management (ISDM) [9], developing functionalities that address important issues for requirements management, such as automatic generation of traceability matrices. The annotations are done in RD templates, and they are based on the conceptualization defined in the Software Requirements Ontology proposed in [13]. It is important to highlight that, since we annotate templates, the effort spent with annotations is very small compared with the benefits obtained.

The resulting tool was used to support requirements management in 7 projects. From this preliminary use, we have already glimpsed some improvements to be done. First, we need to develop a more user-friendly interface for performing searches and for displaying traceability matrices. Second, ISDM provides other features that can be explored in the context of Requirements Management. This is the case of the change notification subscription functionality. This feature allows notifying users when a given individual in a document changes. This could be useful to notify stakeholders when a requirement that they are interested in changes. Third, although our approach has shown to be useful for organizations that use a desktop text editor for documenting requirements, we know that, ideally, software organizations should use requirements management tools for that. Thus, we are working on integrating ISDM with the requirements management tool of ODE (Ontology-based Development Environment) [17], a Software Engineering Environment developed at NEMO.

ACKNOWLEDGMENTS

This research is funded by the Brazilian Research Funding Agencies FAPES (Process Number 45444080/09) and CNPq (Process Numbers 481906/2009-6 and 483383/2010-4).

REFERENCES

[1] A. Aurum, C. Wohlin, Engineering and Managing Software Requirements, Springer-Verlag, 2005.
[2] A. Lamsweerde, Requirements Engineering: From System Goals to UML Models to Software Specifications, Wiley, 2009.
[3] G. Kotonya, I. Sommerville, Requirements engineering: processes and techniques, John Wiley, 1998.
[4] K.E. Wiegers, Software Requirements: Practical techniques for gathering and managing requirements throughout the product development cycle. 2nd Edition, Microsoft Press, 2003.
[5] T. C. Lethbridge, J. Singer, A. Forward, "How Software Engineers Use Documentation: The State of the Practice", IEEE Software, vol. 20, no. 6, pp. 35-39, Nov./Dec. 2003.
[6] Forward, A., Lethbridge, T.C., "The relevance of software documentation, tools and technologies: a survey". Document Engineering. DocEng'02, 2002.
[7] Bruggemann, B. M.; Holz K.P.; Molkenthin F., "Semantic Documentation in Engineering", Proceedings of the Eighth International Conference on Computing in Civil and Building Engineering , California, USA, August 2000.
[8] H. Eriksson, "The semantic-document approach to combining documents and ontologies", International Journal of Human-Computer Studies, Volume 65 , Issue 7, 2007.
[9] L.O. Arantes, R.A. Falbo, "An Infrastructure for Managing Semantic Documents", Proc. 14th IEEE International Enterprise Distributed Object Computing Conference Workshops (EDOCW), 2010.
[10] S.L. Pfleeger, J. Atlee, Software Engineering: Theory and Practice, 4th edition, Prentice Hall, 2009.
[11] OASIS Open Document Format for Office Applications. *www.oasis-**open**.org/committees/**office**/*
[12] OpenOffice.org – The Free and Open Productivity Suite. Visited in April, 30th 2010. http://www.openoffice.org/
[13] Falbo, R. A., Nardi, J., C., Evolving a Software Requirements Ontology. In: Proceedings of the XXXIV Latin-american Conference on Informatics - CLEI´2008, Santa Fé, Argentina, 2008. p. 300-309.
[14] S. Rochimah, W. M. N. Wan Kadir, A. H. Abdullah, "An Evaluation of Traceability Approaches to Support Software Evolution", The Second International Conference on Software Engineering Advances (ICSEA 2007), French Riviera, France, 2007.
[15] J. Cleland-Huang, R. Settimi, E. Romanova, B. Berenbach, S. Clark, "Best Practices for Automated Traceability", Computer, Vol. 40, Issue 6, pp. 27 – 35, June 2007.
[16] H.P.-J.Thunem, "TRACE: A Generic Tool for Dependable Requirements Engineering". The European Safety and Reliability 2009 Conference, Volume 1, pp. 137–142, Prague, Czech Republic, 2009.
[17] R. A. Falbo, F. B. Ruy, R. Dal Moro, "Using Ontologies to Add Semantics to a Software Engineering Environment", Proceedings of the17th International Conference on Software Engineering and Knowledge Engineering, Taipei, China , 151-156 , 2005.

Design Software Architecture Models using Ontology

Jing Sun[1], Hai H. Wang[2] and Tianming Hu[3]
[1]Department of Computer Science, The University of Auckland, New Zealand
j.sun@cs.auckland.ac.nz
[2]School of Engineering and Science, Aston University, United Kingdom
h.wang10@aston.ac.uk
[3]Department of Computer Science, Dongguan University of Technology, China
tmhu@ieee.org

Abstract

Software architecture plays an essential role in the high level description of a system design, where the structure and communication are emphasized. Despite its importance in the software engineering process, the lack of formal description and automated verification hinders the development of good software architecture models. In this paper, we present an approach to support the rigorous design and verification of software architecture models using the semantic web technology. We view software architecture models as ontology representations, where their structures and communication constraints are captured by the Web Ontology Language (OWL) and the Semantic Web Rule Language (SWRL). Specific configurations on the design are represented as concrete instances of the ontology, to which their structures and dynamic behaviors must conform. Furthermore, ontology reasoning tools can be applied to perform various automated verification on the design to ensure correctness, such as consistency checking, style recognition, and behavioral inference.

1. Introduction

Software architecture plays an essential role in the high level design of a software system. Analogy to civil engineering, it represents the fundamental structural and behavioral descriptions of the system under design. Despite its importance in the software engineering process, the lack of formal description and automated verification hinders the development of good architecture models. Traditionally, software architectures are specified using diagrammatic and textual notations [8, 9]. However, architecture models defined in this manner are likely to be inconsistent and error prone due to the informality of the description and the lack of means of rigorous verification to ensure the correctness. As a result, formal modeling techniques have been applied to software architecture descriptions [1, 4], which is aimed at achieving precise specification and rigorous verification of the intended structures and behaviors in the design. The main advantage of such verifications is the ability to determine whether a modeled structure can correctly satisfy a set of given properties in the requirements of a system in order to assure quality of design. However, one drawback of many existing approaches in the automated verification of software architecture models is their limited scalability, as large architecture models are computationally expensive to process. For example, a recent work by Kim and Garlan [7] proposed the modeling and verification of architecture styles using the Alloy language and its analyzer. In their approach, a few architecture styles based on ACME descriptions were translated and verified using the Alloy analyzer. Although it offers a useful insight to the ability of applying formal modeling in automating the verification of architecture descriptions, the performance issue is a practical limitation of the research. The problem arose from that large-sized architecture models dramatically expand the search spaces of the verification in the Alloy SAT solver. To overcome this problem, Wong et al. [11] recently proposed a model splitting approach for the parallel verification of Alloy based architecture models using their underlying styles. The approach improved the performance of the verification. However, the overheads of the model decomposition as well as the dependency issues among the sub-models during the parallel verification phase still remain as challenges.

On the other hand, there are mature large-scale reasoning means from the Semantic Web [2] technology that can be adopted to provide more promising solutions towards the problem, both in terms of knowledge representation and automated verification. The semantic web vision recommended by the World Wide Web Consortium (W3C) has emerged as the next generation of the web from the late

nineties. It extends the current web by assigning web content with a well-defined meaning, aimed at enabling intelligent machine processing of web resources. Description Logic (DL) based ontology languages, such as the Web Ontology Language (OWL) [6] and Semantic Web Rule Language (SWRL) [5], have been proposed to meet the needs of representing the complicated relationships among different entities. One of the advantages of the ontology languages, such as OWL and SWRL, is that DL reasoning engines, such as Pellet [10] and Jess [3], can be effectively used to perform large-scale automated reasoning on ontologies and their instances, e.g., subsumption reasoning, consistency checking, classification and knowledge inference. The scale of such verifications is usually in terms of thousands of ontological individuals (instances). To a certain extent, the semantic web approach is essentially an application of formal methods into the web community, where web resources are formally specified using description logic notations and rigorously verified using ontological reasoning engines.

In this paper, we explore the synergy between software architecture modeling and the semantic web ontological reasoning. We view software architecture models as ontology representations, where their structures and communications must hold. The overall approach of our ontological based software architecture modeling and verification is presented in Figure 1. As shown from the diagram, we

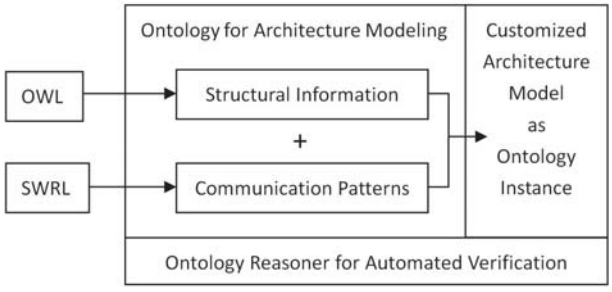


Figure 1. Overall Approach of Ontology-based Architecture Modeling and Verification.

first define ontology models that are specific to software architecture design using the OWL/SWRL languages based on different architecture styles. OWL is used to specify the structure and relationship constraints among the architecture terminologies, e.g., the participating components, connectors and their connections. SWRL is used to capture the dynamic interactions within the architecture models as additional constraints, e.g., the requester and provider communication protocol of a Client and Server style. The OWL ontology plus the SWRL rules form the meta-ontology that can be used to create customized architecture specifications. When users define their own customized architecture models by extending the meta-ontology (styles), all the meta-level rules are automatically enforced upon the customized models, i.e., the particular structures and communications are carried over onto the new design. A customized ontology can also consist of system specific terminologies and rules. For example, we can customize the meta-ontology into defining the architecture model of a specific internet application that extends the Client and Server style and yet has its own additional communication behaviors. After creating the customized ontology, architecture configurations can be defined as instances of the underlying ontology design, e.g., specific numbers of web servers and named clients involved in the system layout, etc.

Furthermore, there are two levels of verification that can be performed in our approach to ensure the correctness of an architectural design, namely, the ontology and the instance levels. The former ensures the correctness of the architecture model itself, e.g., no conflicting connection and communication constraints among the components. The latter ensures the conformance of a particular configuration with respect to its design, e.g., a certain configuration should always have its pre-defined structures and perform its own communication protocols. Moreover, ontology reasoners can be used to fully automate the verification process. For example, the Pellet reasoner can be used to check the consistencies of the ontology and instance models, and the Jess rule engine can be used to automatically derive the communication sequences on configurations, etc. In addition, ontology classifiers can be used to automatically recognize the style patterns that are used in a design at the ontology level.

The remainder of the paper is organized as follows. Section 2 presents ontology definitions for software architecture modeling which includes commonly used architecture styles. Section 3 illustrates the use of the architecture ontology in modeling and verifying a customized Three-Tier network architecture as a case study. Finally, section 4 concludes the paper and discusses the future work.

2. Ontology for Software Architecture Design

Software architecture modeling focuses on the high level description of a system in term of the elements in the system and the interactions among the elements. Components and connectors are two fundamental units in an architecture description [1]. Components describe the identifiable computation entities of a system, where connectors specify the patterns of communication among these entities. Each component consists of a set of ports as its external visible interfaces. A connector consists of a set of roles that are used to describe the patterns of a particular kind of communication. In order to establish a connection between different components of the system, the roles of a connector must be attached to the ports of the participating components.

The use of connectors separates the concern of computation from communication in a system description.It further promotes encapsulation and reusability in architecture modeling. In OWL, we define the following classes and properties to represent the different entities and their relationships in an architectural design.

$$\begin{array}{ll} \mathit{Component} \sqsubseteq \top & \mathit{Connector} \sqsubseteq \top \\ \mathit{Port} \sqsubseteq \top & \mathit{Role} \sqsubseteq \top \\ \mathit{Process} \sqsubseteq \top & \mathit{Id} \sqsubseteq \top \end{array}$$
$$\mathit{Component} \sqcap \mathit{Connector} \sqcap \mathit{Port} \sqcap \mathit{Role} \sqcap \mathit{Process} \sqcap \mathit{Id} = \bot$$

The above defines `Component`, `Connector`, `Port`, `Role` and `Process` as mutually disjoint classes. They are top level entities in the architecture description, where the `Process` represents an atomic behavior that a port or a role can perform. The `Id` defines the message identification number that is shared by different processes involved in a communication. For example, the processes in a client and server communication, e.g., sending a request, invoking the server, server returning the result, receiving the result back by the client, may share the same message id for referring to the same client request interacting through the client and the server.

$$\begin{array}{ll} \geq 1\, \mathit{hasPort} \sqsubseteq \mathit{Component} & \top \sqsubseteq \forall\, \mathit{hasPort}.\mathit{Port} \\ \geq 1\, \mathit{hasRole} \sqsubseteq \mathit{Connector} & \top \sqsubseteq \forall\, \mathit{hasRole}.\mathit{Role} \\ \geq 1\, \mathit{hasAttachment} \sqsubseteq \mathit{Role} & \\ \top \sqsubseteq \forall\, \mathit{hasAttachment}.\mathit{Port} & \top \sqsubseteq \leq 1\, \mathit{hasAttachment} \end{array}$$
$$\mathit{isAttachedTo} = (^{-}\mathit{hasAttachment})$$
$$\geq 1\, \mathit{hasAction} \sqsubseteq (\mathit{Port} \cup \mathit{Role})$$
$$\top \sqsubseteq \forall\, \mathit{hasAction}.\mathit{Process}$$
$$\geq 1\, \mathit{hasCommunication} \sqsubseteq \mathit{Process}$$
$$\top \sqsubseteq \forall\, \mathit{hasCommunication}.\mathit{Process}$$

After defining the basic entities, we use OWL property definitions to further capture the relationships among them. For example, the `hasPort` is a relation from a `Component` as its domain to a Port as its range, which simply denotes that a component has a set of ports as its interfaces. Similarly, the `hasRole` represents that a connector has a set of roles as its participating parties in the connection. The attachment of a role to a specific port is defined as the property `hasAttachment`. Note that `hasAttachment` is defined as a functional property from a role to a port, which means that a port can be attached to different roles, but not vise versa. The relation `isAttachedTo` is an inverse relation to the `hasAttachment` function, which maps a port back to its attached role. We also define two more relationships to capture the dynamic interactions within a component or a connector and their communications. The property `hasAction` describes the set of atomic processes that a port or a role is able to perform. The property `hasCommunication` captures the interaction sequences of a particular communication (protocol). In addition, the connectivity constraint need to be enforced upon the model, i.e., in order to establish a valid attachment between a role and a port, the port must be able to perform the behaviors (actions) that the role has. Otherwise, the port can not 'play' the role in the connection due to behavior mismatch. This can be easily specified using a SWRL rule as follows.

$$\mathit{hasAttachment}(?r, ?p) \wedge \mathit{hasAction}(?r, ?x) \rightarrow \mathit{hasAction}(?p, ?x)$$

The above states that if a role `?r` is attached with a port `?p` and the role has a process `?x` in its action, then the port must also have the same process `?x` in its action set. Note that a port may have more behaviors than that of its attached role, which is allowed during an attachment. This also gives the flexibility for a port to be able to participate in different roles of connections. The ontology reasoning engine asserts the above rule when checking the correctness of the attachments in the architecture design.

After defining an ontology for the component and connector model, we will further demonstrate the construction of commonly used software architecture styles by extending this base ontology. In the following subsection, we use the Client and Server style as an example to illustrate the modeling. Other styles can be defined in a similar manner.

2.1. Ontology for the Client and Server style

The Client and Server structure is one of the most widely adopted styles in software architecture description, especially in distributed and network based computer systems. It offers a loosely coupled multi-connection mechanism in a consumer and provider fashion. A server is a component that provides a set of services that are exposed to various clients for consumption. A client is a component that acts as a consumer to those services. We define the following OWL classes for the Client and Server style by extending the component and connector ontology.

$$\begin{array}{ll} \mathit{Client} \sqsubseteq \mathit{Component} & \mathit{Server} \sqsubseteq \mathit{Component} \\ \mathit{Request} \sqsubseteq \mathit{Port} & \mathit{Provide} \sqsubseteq \mathit{Port} \\ \mathit{CnS} \sqsubseteq \mathit{Connector} & \\ \mathit{Provider} \sqsubseteq \mathit{Role} & \mathit{Consumer} \sqsubseteq \mathit{Role} \\ \mathit{SendRequest} \sqsubseteq \mathit{Process} & \mathit{ReceiveResult} \sqsubseteq \mathit{Process} \\ \mathit{InvokeServer} \sqsubseteq \mathit{Process} & \mathit{ServerReturn} \sqsubseteq \mathit{Process} \end{array}$$

The above extends the component and connector model and defines the basic entities involved in a Client and Server structure, i.e., client, server, connector, ports, roles and participating processes. Note that the `Request` port belongs to the `Client` component and the `Provide` port belongs to the `Server` component. `CnS` is a connector that has a `Provider` role and a `Consumer` role. We refined the basic behaviors of the style into four communication processes, i.e., `SendRequest`, `ReceiveResult`, `InvokeServer` and `ServerReturn`, where the first two belongs to the consumer and the last two belongs to the provider. These relations are specified using OWL properties as follows.

$Client \sqsubseteq \exists hasPort.Request \quad Server \sqsubseteq \exists hasPort.Provide$
$CnS \sqsubseteq \exists hasRole.Provider \quad CnS \sqsubseteq \exists hasRole.Consumer$
$CnS \sqsubseteq \forall hasRole.(Provider \sqcup Consumer)$
$Request \sqsubseteq \exists isAttachedTo.Consumer$
$Request \sqsubseteq \forall isAttachedTo.Consumer$

In addition, we defined a necessary and sufficient condition for the `CnS` connector, where there is a closure axiom on the `hasRole` property. This enforces the classification on any connector that only has a provider and a consumer as its roles to be a connector of the Client and Server style. Note that the necessary and sufficient conditions contribute to the automatic recognition of architecture styles through ontology classification, which will be demonstrated later.

With the structure information of the Client and Server style defined, we then use SWRL rules to capture its dynamic communication behaviors (protocol) as follows. We divided the entire communication of the Client and Server style into two parts, i.e., the communication of the connector and the internal behaviors within the server. The former consists of two action sequences - (1) receive a request from the consumer role and invoke the server through the provider role, (2) receive a server return value from the provider role and send the result back through the consumer role. This is captured by the first SWRL rule below using the `hasCommunication` property. The latter represents the internal actions of a server that contributes to the overall communication of the service consumption, i.e., whenever there is an invocation of request on the server, there should always be a corresponding response returned from the server. This is captured using the second SWRL rule shown below.

$$CnS(?cn) \wedge Consumer(?cr) \wedge Provider(?pr) \wedge$$
$$p1 : Id(?id) \wedge SendRequest(?r) \wedge InvokeServer(?i) \wedge$$
$$ServerReturn(?ret) \wedge ReceiveResult(?res) \wedge$$
$$p1 : hasRole(?cn, ?cr) \wedge p1 : hasRole(?cn, ?pr) \wedge$$
$$p1 : hasAction(?cr, ?r) \wedge p1 : hasAction(?cr, ?res) \wedge$$
$$p1 : hasAction(?pr, ?i) \wedge p1 : hasAction(?pr, ?ret) \wedge$$
$$p1 : hasId(?r, ?id) \wedge p1 : hasId(?i, ?id) \wedge$$
$$p1 : hasId(?ret, ?id) \wedge p1 : hasId(?res, ?id)$$
$$\rightarrow p1 : hasCommunication(?r, ?i) \wedge$$
$$p1 : hasCommunication(?ret, ?res)$$

$$Server(?s) \wedge Provide(?p) \wedge InvokeServer(?i) \wedge$$
$$ServerReturn(?r) \wedge p1 : hasPort(?s, ?p) \wedge$$
$$p1 : hasAction(?p, ?i) \wedge p1 : hasAction(?p, ?r) \wedge$$
$$p1 : Id(?id) \wedge p1 : hasId(?i, ?id) \wedge p1 : hasId(?r, ?id)$$
$$\rightarrow p1 : hasCommunication(?i, ?r)$$

After completing the ontology definition for the Client and Server style, we can invoke the Pellet reasoner to check the consistency on the ontology. In addition, we can create particular configurations of the style and infer new knowledge on the models. We will demonstrate these verification aspects with an example in the next section.

3. A Three-Tier Network Architecture

In this section, we demonstrate the modeling and verification of a Three-Tier network example by using the architecture style ontology that we defined earlier. In modern web applications, the systems are constructed using a multi-layered structure, where a middle-ware layer is introduced to encapsulate the business logics. The middle layer is named as the 'Application Server', which can be considered as a mediator between the clients and the data storage facilities. The application server acts as both a server to the user requests and a client to the database server for data manipulations. We model the Three-Tier architecture using the following OWL classes by extending the style ontology that we defined in the previous sections.

$ThinClient \sqsubseteq Component$
$ApplicationServer \sqsubseteq Component$
$DatabaseServer \sqsubseteq Component$
$CnA \sqsubseteq Connector \quad AnD \sqsubseteq Connector$
$AppRequest \sqsubseteq Port \quad AppProvide \sqsubseteq Port$
$AppConsumer \sqsubseteq Role \quad DBProvider \sqsubseteq Role$
$SendApp \sqsubseteq SendRequest \quad AppResult \sqsubseteq ReceiveResult$

We define the `ThinClient`, `ApplicationServer`, `DatabaseServer` as basic components involved in the structure. In addition, `CnA` is a connector that connects the thin client with the application server, while `AnD` is the connector that relates the application server to its database server. The above also gives the definitions of the ports used in the model, i.e., `AppRequest`, `AppProvide`, `DBRequest`, and `DBProvide`. The roles used in the structure include `AppConsumer`, `AppProvider`, `DBConsumer` and `DBProvider`. Finally, we define the processes that involved in the communication of the structure, e.g., `SendApp` extends the `SendRequest` process of the Client and Server style ontology, which denotes the client sending a request to the application server. Other processes, such as `AppResult` extends the `ReceiveResult` process, `AppReturn` extends the `ServerReturn` process, and so on, can be defined in a similar manner. With the entities of the Three-Tier ontology specified, we can add property definitions to relate the classes, e.g., the client only has application request ports, the application server has both a provide port to its client and a request port to the database server, and so on. Due the space limit, we can not list all of them here. Furthermore, the communication behaviors of the Three-Tier architecture need to be redefined in terms of the specific processes used in the new structure. For example, the interactions of the application server are defined using the following SWRL rule.

$$ApplicationServer(?as) \wedge AppProvide(?ap) \wedge$$
$$DBRequest(?dr) \wedge AppAccess(?aa) \wedge AppReturn(?ar) \wedge$$
$$SendDB(?sd) \wedge DBResult(?dres) \wedge p1 : Id(?id) \wedge$$

$$p1 : hasId(?aa, ?id) \wedge p1 : hasId(?sd, ?id) \wedge$$
$$p1 : hasId(?dres, ?id) \wedge p1 : hasId(?ar, ?id) \wedge$$
$$p1 : hasPort(?as, ?ap) \wedge p1 : hasPort(?as, ?dr) \wedge$$
$$p1 : hasAction(?ap, ?aa) \wedge p1 : hasAction(?ap, ?ar) \wedge$$
$$p1 : hasAction(?dr, ?sd) \wedge p1 : hasAction(?dr, ?dres)$$
$$\rightarrow p1 : hasCommunication(?aa, ?sd) \wedge$$
$$p1 : hasCommunication(?dres, ?ar)$$

The above states that `ApplicationServer` has two ports i.e., `AppProvide` and `DBRequest`. The `AppProvide` port acts as the interface for providing services to the client and has `AppAccess` and `AppReturn` as its actions, while the `DBRequest` port acts as the interface for requesting data access service from the database server and has `SendDB` and `DBResult` as its actions. The behavior of the application server mainly consists of two types of communication sequences, i.e., (1) $\langle AppAccess, SendDB \rangle$ - invokes service from a client request via `AppAccess` and sends the converted data request to database server via `DBRequest`; (2) $\langle DBResult, AppReturn \rangle$ - receives data result from the database server via `DBResult` and returns the converted result to the client via `AppReturn`. This concludes the dynamic behaviors of the application server. Similarly, the communication behaviors of other entities in the model, such as, the `ThinClient` and `DatabaseServer` components, the `CnA` and `AnD` connectors can be redefined by simply cloning the server and connector communication rules from the Client and Server style ontology.

3.1. Styles recognition via classification

Up to now, we have completed the ontology definition of the Three-Tier network architecture model. Careful readers might have already noticed that the entities of the Three-Tier ontology mainly extend the component and connector model, apart from its specific process definitions. This was 'deliberately' chosen to demonstrate the automatic style recognition via ontology classification. In Section 3.2, we mentioned that the necessary and sufficient conditions with closure axioms can contribute to the automatic recognition of different architecture styles through ontology classification. Figure 2 represents the inferred hierarchy of the classes in the Three-Tier example after running the 'classify taxonomy' function of the Pellet ontology reasoner. As shown in the diagram, the left-hand side panel contains the asserted (original) class hierarchy of the Three-Tier ontology by directly extending the component and connector model, the right-hand side panel contains the inferred class hierarchy after the classification. We can see that except for the `ApplicationServer`, all other classes that we defined in the Three-Tier ontology have been re-classified into the corresponding categories under the Client and Server ontology. For example, the `ThinClient` is moved to a subclass of the `Client`, and the `DatabaseServer` is now under the `Server` class. The `CnA` and `AnD` connectors have both been moved to subclasses of the `CnS` connector. Similarly, the ports and roles that we defined earlier have also been re-classified under the specific ports and roles of the Client and Server style ontology. Note that in this example, because we defined a server to be only having the 'Provide' ports and a client to be only having the 'Request' ports, the `ApplicationServer` in Figure 2 was not classified as either a client or a server, since it has both an `AppProvide` port to the thin client and a `DBRequest` port to the database server.

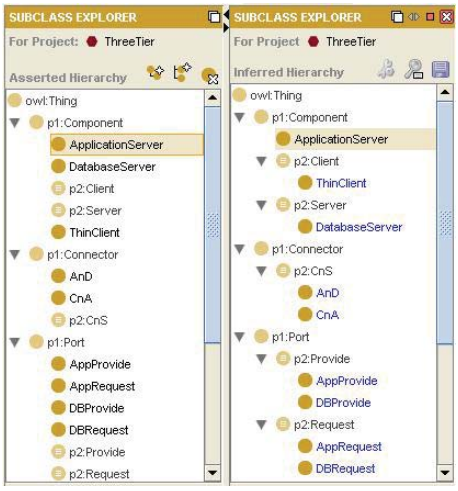


Figure 2. Automated style recognition of the Three-Tier architecture via classification.

Effectively speaking, the above indicates that the ontology classifier have automatically recognized the Three-Tier ontology to be composed of two basic Client and Server structures. We believe that this kind of automated style recognition can be very useful during software architecture modeling. The users only need to create their customized architecture models by extending the top level component and connector ontology, and let the ontology classifier to automatically recognize the styles that involved in the design. It not only helps the users to realize the architectural design patterns, but also assists the designers in decomposing a complex system into clear structured and verifiable sub-models.

3.2. Communication generation via inference

With the Three-Tier ontology defined, we can create specific configurations of this structure through the instance declarations. For example, we defined a network configuration with twenty thin clients, two application servers and one database server. Each application server

handles about ten clients and they both connected to the same database server. Note that such a configuration involves 294 instances in total, which consists of the instances for all the components, connectors, ports, roles, processes, and client identifiers. Once the instances are created, we attach the desired roles to their corresponding ports to form the communication topology. When ports and roles are connected, we can then invoke the Jess rule engine to automatically infer the communication sequences on the instance configuration. For example, a typical inferred communication sequence for $Client_1$ could be – '$\langle SendApp_1, AppAccess_1\rangle$, $\langle AppAccess_1, SendDB_1\rangle$, $\langle SendDB_1, DBAccess_1\rangle$, $\langle DBAccess_1, DBReturn_1\rangle$, $\langle DBReturn_1, DBResult_1\rangle$, $\langle DBResult_1, AppReturn_1\rangle$, $\langle AppReturn_1, AppResult_1\rangle$'. This effectively illustrates exactly what happened in the model in terms of communication interactions among the entities, i.e., starting from the client submits a request to the server until the client receives the corresponding result. The Jess rule engine generates 300 inferred axioms on the configuration model, which includes twenty separately identified communication sequences with respect to the number of clients involved in the configuration (each contains the above seven communication pairs with different client identifier labeled) to simulate the interactions among the instances of the clients and servers.

4. Conclusion

In this paper, we proposed a formal approach to software architecture modeling and verification using the semantic web technology. We represented architecture models as ontology descriptions and applied DL reasoners to perform automated verification on the design. The OWL definitions specify the structure information of the model, where the SWRL rules capture the dynamic communication in the styles. Users can easily extend the style ontology in defining their customized architecture models. In the aspect of automated verification, we demonstrated two levels of reasoning, i.e., style recognition via ontology classification and communication sequences generation using rule inference. The former works on the ontology level and can automatically recognize the style patterns used in the design, where the latter applies to the instance level and enables the users to automatically derive the interactions within the configuration of a design. As a complex architecture model usually consists of a combination of different architecture styles, these two levels of verification can be effectively used together to enhance the quality of a design. For example, when creating a customized architecture ontology model, the user only needs to define the model by extending the top level component and connector ontology and then invoke the classifier to automatically recognize the architecture styles used in the design. Once the customized architecture model is classified into recognizable styles, the users can then create specific configurations (e.g., partially) of the design (ontology) and invoke the rule engine to automatically infer new knowledge and derive the communication sequences on the model. To demonstrate the effectiveness of our approach, we illustrated the design and verification of a Three-Tier network architecture model as the case study.

In the future, we plan to develop a visual tool support that aims at assisting the design and verification of architecture models based on the proposed ontology approach. The tool should allow the designers to build their architecture models graphically without knowing the underlying syntax and knowledge of OWL/SWRL. It should integrate all the ontology definitions and verification steps described in the paper and automated them in one coherent visual interface.

References

[1] R. Allen and D. Garlan. A Formal Basis for Architectural Connection. *ACM Trans. Softw. Eng. Methodol.*, 6(3):213–249, 1997.

[2] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 2001.

[3] E. Friedman-Hill. *Jess in Action: Java Rule-Based Systems*. Manning Publications Co., Greenwich, CT, USA, 2003.

[4] D. Garlan and B. Schmerl. Architecture-driven Modelling and Analysis. In *11th Australian Workshop on Safety Critical Systems and Software*, pages 3–17. Australian Computer Society, 2006.

[5] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. `http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/`, 2004.

[6] I. Horrocks, P. F. Patel-Schneider, and F. van Harmelen. From $\mathcal{SHIQ}$ and RDF to OWL: The Making of a Web Ontology Language. *J. of Web Semantics*, 1(1):7–26, 2003.

[7] J. S. Kim and D. Garlan. Analyzing Architectural Styles with Alloy. In *ISSTA 2006 Workshop on Role of Software Architecture for Testing and Analysis*, pages 70–80, NY, USA, 2006. ACM.

[8] D. Luckham and J. Vera. An Event-Based Architecture Definition Language. *IEEE Trans. Softw. Eng.*, 21:717–734, September 1995.

[9] N. Medvidovic and R. N. Taylor. A Framework for Classifying and Comparing Architecture Description Languages. In *6th European Software Engineering Conference*, pages 60–76, NY, USA, 1997. Springer-Verlag.

[10] E. Sirin and B. Parsia. Pellet: An OWL DL Reasoner. In *International Workshop on Description Logics*, 2004.

[11] S. Wong, J. Sun, I. Warren, and J. Sun. A Scalable Approach to Multi-Style Architectural Modeling and Verification. In *13th IEEE International Conference on Engineering of Complex Computer Systems*, pages 25–34. IEEE Press, 2008.

Debug Concern Navigator

Masaru Shiozuka
Kyushu Institute of Technology
Fukuoka, Japan
siozuka@gmail.com

Naoyasu Ubayashi
Kyushu University
Fukuoka, Japan
ubayashi@acm.org

Yasutaka Kamei
Kyushu University
Fukuoka, Japan
kamei@ait.kyushu-u.ac.jp

***Abstract*—Programmers tend to spend a lot of time debugging code. They check the erroneous phenomena, navigate the code, search the past bug fixes, and modify the code. If a sequence of these debug activities can be automated, programmers can use their time for more creative tasks. To address this problem, this paper proposes *dcNavi* (Debug Concern Navigator), a concern-oriented recommendation system for debugging. The *dcNavi* provides appropriate hints to programmers according to their debug concerns such as *"how to handle this exception"* and *"how to use this API"*. We propose the notion of DCG (Debug Concern Graph), an extension of the *Concern Graphs*. A DCG appends a set of debugging information such as past bug fixes and test results to a concern graph. Debug hints are provided in the form of code snippets showing the sample code modification before and after a bug fix.**

***Keywords*-debug, concern graph, recommendation**

I. Introduction

In many software development projects, programmers tend to spend a lot of time debugging code [5]. They check the erroneous phenomena, navigate the code that may include bugs, and search external useful resources such as API specifications and documents. It is effective if IDEs (Integrated Development Environments) provide programmers with advices summarized by these external resources. However, it is not clear which kind of information navigation should be provided to the programmers because these external resources provide only indirect hints for debugging and do not always resolve the bugs.

It is favorable if IDEs can provide the code snippets before and after similar bug fixes by searching past repositories because programmers can understand how to fix a bug by referring concrete examples. Especially, it is useful for novice programmers. To achieve this goal, we have to develop the following mechanisms because the code in the repositories should be associated to the debugging activities in order to provide effective hints: 1) a mechanism for collecting and archiving the debugging activities from IDEs; 2) a mechanism for associating the debugging activities to the program structures; and 3) a mechanism for retrieving the code snippets advising bug fixes from the repositories.

To deal with this challenge, this paper proposes *dcNavi* (Debug Concern Navigator), a concern-oriented recommendation system for debugging. The *dcNavi* provides appropriate hints to programmers according to their debug concerns by using a repository containing not only program information but also past test results and their bug fixing history. We propose the notion of DCG (Debug Concern Graph), an extension of the *Concern Graphs* [10], [11] that helps programmers identify and reason about concerns in programming and maintenance tasks by abstracting the implementation details of a concern and makes explicit the relationships among concerns. A DCG appends a set of debugging information such as bug fixes and test results to a concern graph.

The remainder of this paper is structured as follows. In Section 2, we point out the issues on debugging. In Section 3, the concept of DCGs is illustrated. In Section 4, an overview of *dcNavi* is shown. In Section 5, the effectiveness of *dcNavi* is evaluated. In Section 6, related work is introduced. Concluding remarks are provided in Section 7.

II. Motivation

In this section, we discuss the problems on debugging by showing a typical debugging scenario.

A. A typical debugging scenario

We consider a situation in which a student is debugging a Java program handling file operations such as *read* and *write*. The student with insufficient programming skills tends to make a mistake because these file operations interact with external resources. The code below shows the student's program that reads a text file. In this program, the *readFile* method reads a text line from a file specified by the parameter. In the case of *read* failure, null is returned.

```
[List 1]
01: public class Property {
02:   public String readFile (String pathname)
03:     throws IOException {
04:       String val = null;
05:       File file = new File(pathname);
06:       FileReader fr = new FileReader(file);
07:       BufferedReader br = new BufferedReader(fr);
08:       val = br.readLine();
09:       return val;
10:   }}
```

The code below shows a unit test program for the *readFile* method. While the *testReadFile* method is successfully ter-

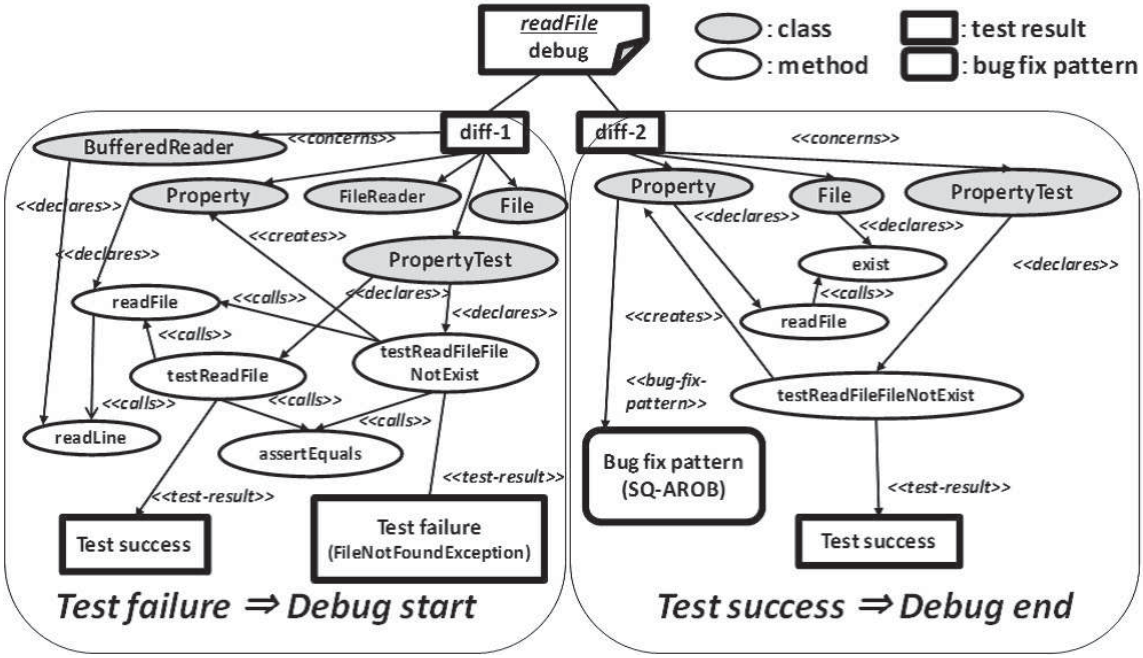


Figure 1. DCG: Debug Concern Graph

minated, *FileNotFoundException* is thrown in the case of the *testReadFileFileNotExist* method execution (line 13).

```
[List 2]
01: public class PropertyTest extends TestCase {
02:   public void testReadFile()
03:     throws IOException {
04:       Property property = new Property();
05:       assertEquals("true",
06:         property.readFile("property.txt"));
07:   }
08:   public void testReadFileFileNotExist()
09:     throws IOException {
10:       Property property = new Property();
11:       // property2.txt does not exist !!
12:       assertEquals(null,
13:         property.readFile("property2.txt"));
14:   }}
```

The student begins to debug the program and is faced with the following three problems: 1) although the programmer guesses that he or she has to append the code for dealing with the case in which a file does not exist, the programmer does not know how to write the code avoiding *FileNotFoundException* because it is an unfamiliar exception for the programmer; 2) the programmer cannot be convinced of the correctness of the API usage (*BufferedReader*) because it is an unfamiliar library; and 3) the programmer does not know whether other test cases are needed.

Although it might not be difficult for professional programmers to predict the cause of a bug by referring to their experience, many beginners are faced with the various difficulties when they debug a program. For example, it is not easy for a beginner to use program exceptions or test failures as a hint for fixing a bug because they do not have enough experience and knowledge for programming languages, library usages, and source code reviews. Even if a programmer has a long career, he or she will be faced with the similar difficulties when the programmer works for unfamiliar new projects, languages, and frameworks.

B. Our approach

Many programmers tend to make the same kind of bugs concerning API usages, exception handling, and domain knowledge [4], [8], [9], [12]. It is effective to provide hints from past bug fix data.

This paper proposes *dcNavi*, a recommendation system for giving an answer to the programmer's questions such as *"how to handle this exception"* and *"how to use this API"*. We call these questions *debug concerns*. The *dcNavi* provides hints for bug fixes by reusing debug experience and knowledge archived in the graph structures, DCGs, containing not only source code but also static analysis information, test results, and revision numbers before/after bug fixes. By using graph structures as knowledge bases, it becomes easy to navigate and search related information that may be hints for debugging. In *dcNavi*, a hint is provided in the form of code snippets showing the code modification before and after a bug fix because the correct code samples are effective for program understanding.

III. Debug Concern Graph

In this section, we explain the notion of DCG in detail.

A. Graph structure

Figure 1 illustrates two DCGs showing before and after debugging the *readFile* method from Section 2. The root

node *readFile debug* indicates the name of a debug activity consisting of sub-activities related to a test failure (*diff-1*) and a test success (*diff-2*). We consider *test failure time* and *test success time* as *debug start point* and *debug end point*, respectively.

The DCG consists of element nodes and edges. The nodes include not only program elements such as classes and methods, but also a variety of debugging activities annotated with applied bug fixes and test results. Each node is connected to other nodes by edges annotated with stereotypes such as *calls* (method call), *creates* (object instantiation [some edges are omitted in Figure 1 for simplicity]), *declares* (method declaration), and three kinds of relations: 1) *concerns* (relation between a debug activity and a class concerned by a programmer); 2) *test-result* (relation between a test class and its execution result); and 3) *bug-fix-pattern* (relation between a modified class and an applied bug fix pattern).

The bug fix patterns are the code modification catalogues for fixing a variety of bugs and can be considered hints for debugging. We adopted the bug fix patterns proposed in [9]. The patterns are well documented and consist of twenty-seven catalogues such as MC-DAP (Method Call with Different Actual Parameter Values), SQ-AROB(Addition or Removal of Method Calls in a Short Construct Body), and IF-CC (Change of If Condition Expression). The *dcNavi* detects the applied bug fix pattern by comparing the code before debugging with the code after the bug fix. In Figure 1, the SQ-AROB pattern is applied.

B. Graph evolution

A DCG is created and expanded automatically as a programmer debugs a program. First, a sub-activity-node is created when a test is executed. The structure of the *diff-1* node is fixed and the *diff-2* node is created at the time of test failure. If the test fails, a programmer begins to browse classes, methods, and fields that might include a bug. These browsed program elements are automatically added to a DCG. The result of the test execution is also added to the DCG. After the code modification is completed, the test is executed again. If the test succeeds, the bug fix pattern node is added to the DCG.

Our approach is integrated with TDD (Test-Driven Development) [2]. The TDD events such as test execution, fail, success can be good triggers for enriching concern graphs with debugging activities that are automatically associated to program elements such as classes and methods.

C. Concern query

It is a difficult problem to translate a programmer's debug concern to a query for obtaining the code snippets showing the program modification before and after the bug fix because searching only keywords (e.g., *FileNotFoundException*) included in the debug concern matches with many code fragments that are not relevant to the bug fix. To deal with this problem, *dcNavi* searches a sub graph that matches the bug fix patterns and includes the specified keywords. The bug fix patterns show not only how to fix bugs but also the reasons of the bugs. The bug fix patterns play an important role in our recommendation system because many bug fixes can be categorized into the patterns, the advice can be generated from actual code snippets annotated with the bug reasons, and the recommendation mechanism is implemented by simple pattern matching.

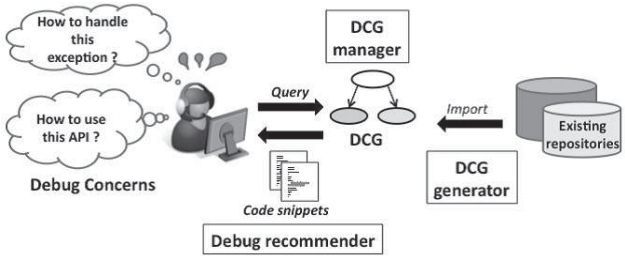


Figure 2. dcNavi

IV. DCNAVI

The *dcNavi* consists of three facilities including *DCG manager* for supporting graph construction and evolution, *DCG generator* for importing existing repositories, and *debug recommender*. Figure 2 illustrates the outline.

The *dcNavi* supports debugging in Java and is implemented as an Eclipse plug-in by using Mylyn and JUnit. Mylyn [7], [3] monitors programmer's activities, creates a *task context* that focuses his or her workspace, and automatically links all relevant artifacts to the task context. JUnit is a unit testing framework for Java.

A. DCG manager

DCG manager creates and expands DCGs as follows: 1) program elements related to debug concerns are automatically captured and added to DCGs by using Mylyn's task-focused interface (*concern* edges in DCGs are also appended automatically); 2) structural associations among program elements such as *declares*, *creates*, and *calls* are added to DCGs by analysing source code; 3) test results are added to DCGs by monitoring JUnit test executions; and 4) the applied bug fix patterns are added to DCGs by checking the *diff* before and after bug fixes determined by test failure and success events.

Programmers do not have to be aware of creating DCGs because the *dcNavi* automatically constructs DCGs based on Mylyn. Using Mylyn, the information only related to the debug activities is added to the DCGs. The static program analysis information, test results, and applied bug fix patterns are merged with Mylyn and Eclipse. Programmers can browse the debug information from the IDE because the *dcNavi* is tightly integrated to Eclipse.

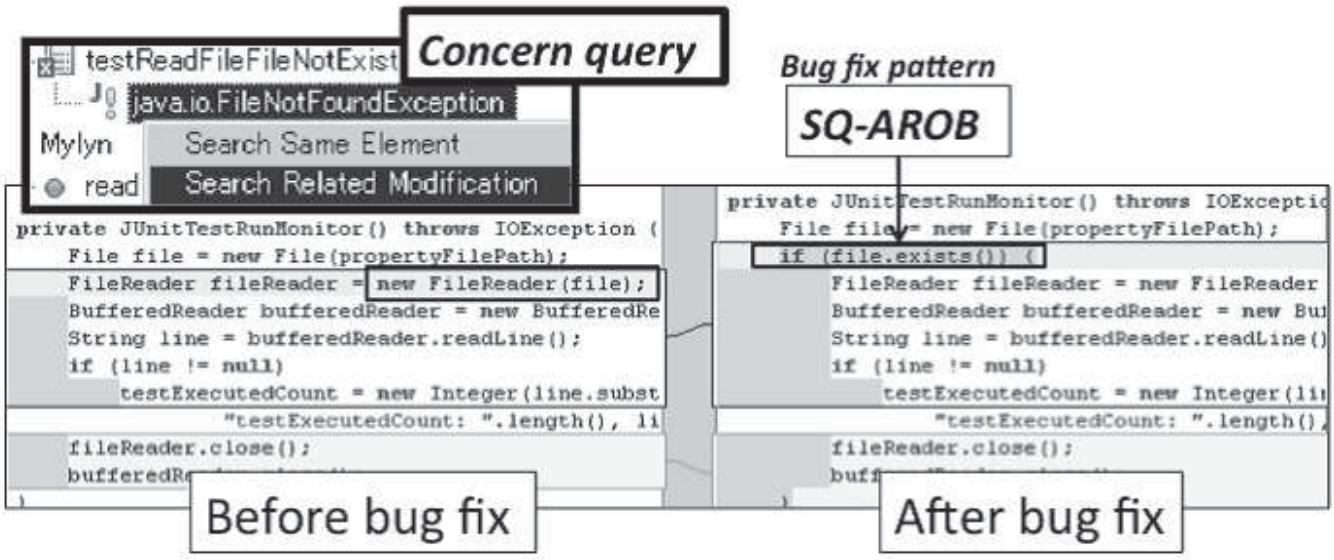


Figure 3. Recommendation result

B. DCG generator

Although our approach is effective for debugging, it needs a repository in which past debugging information is archived. This is not realistic because there are many repositories that do not take into account debugging activities.

To deal with this problem, we developed the DCG generator to convert a set of existing *Subversion* (SVN) repositories such as open source projects into DCGs. The DCG generator uses commit-log keywords such as *bug*, *fix*, and *patch* to define test success or failure, because a sub-activity node (e.g., *diff-1*) in a DCG has to be created per each debugging process staring from a test failure and terminating by a test success. Although a DCG construction is based on TDD in *dcNavi*, there are no data concerning TDD activities in most existing repositories. We have to predict and compensate this data. Moreover, we have to add program elements only concerning to the debug activities because a concern graph is automatically expanded by using the facility of Mylyn. The DCG generator links only program elements related to the bug fixes appearing in the commit-logs to the corresponding sub-activity node by analysing the program structures. That is, *concerns* edges are automatically added to DCGs.

C. Debug recommender

Figure 3 illustrates an example of a concern query *"Search Related Modification" of FileNotFoundException* and its recommendation result. The *dcNavi* explores a DCG from the *testReadFileFileNotExist* node (see Figure 1) and traverse connected nodes such as *Property* and *readFile* with comparing past bug-fixed graphs. The recommendation in Figure 3 shows that a file existence check should be inserted. In this case, the SQ-AROB pattern indicates that *"Addition of Method Calls [file.exists()]"* is needed for this bug fix. This recommendation shows that programmers tend to forget to write this checking code.

The recommendation algorithm is as follows.

Step1) Search a method to be modified: The *dcNavi* searches the test target method because our approach is based on TDD. This method needs to be modified. In this case, we have to find *readFile*, a method that throws the *FileNotFoundException* exception by tracing the *call* edges in the DCG from the *testReadFileFileNotExist* node.

Step2) Obtain recommendation candidates: The *dcNavi* obtains all the tests failing to handle the *FileNotFoundException* exception from the DCG. Next, the *dcNavi* finds the test target methods whose *after-modification* include the bug fix patterns that can be reached from the starting node such as *FileNotFoundException*. These test target methods are recommendation candidates. In the case of Figure 3, *JUnitTestRunMonitor*, a candidate of debug recommendation, includes the SQ-AROB pattern. Currently, the *dcNavi* does not search the IF-CC pattern because it matches with many bug fixes and is not appropriate for obtaining essential bug causes.

Step3) Recommend code snippets: The *dcNavi* recommends the code snippets ranked by the graph similarity metric:

```
Similarity(G1, G2) =
 #common_node / (#G1_node + #G2_node) / 2).
```

This metric indicates the degree of the similarity between two graphs *G1* and *G2*. #common_node is the number of nodes commonly appearing in *G1* and *G2*. This metric becomes high if the number of the nodes commonly shared between a current graph and a past bug-fixed graph is large.

Project	Period (MM/DD/YYYY)	NOR (NOB)	LOC	NOM	ANR	P (%)	R (%)	F(%)
projecthosting-connector	12/16/2009 - 05/14/2010	18(2)	2743	0	N/A	N/A	N/A	N/A
industrial-mylyn	05/20/2010 - 07/09/2010	50(2)	10953	0	N/A	N/A	N/A	N/A
Mylyn-Mntis Connector	02/22/2007 - 07/14/2010	537(107)	18536	20	3.20	23.44	32.61	27.27
Origo	12/22/2006 - 10/08/2010	3761(867)	5769	7	0.00	N/A	0.00	N/A
qcMylyn	08/06/2009 - 09/26/2010	223(80)	13117	54	5.70	34.42	50.24	40.85
Redmine-Mylyn Connector	05/26/2008 - 05/19/2010	423(134)	14729	53	3.13	22.29	33.94	26.91
Remember The Milk	01/26/2008 - 11/24/2009	70(13)	7259	9	2.89	3.85	20.00	6.45
Scrum Vision	05/24/2008 - 07/12/2010	431(14)	43263	10	3.50	0.00	0.00	N/A
subclipse	06/20/2003 - 10/14/2010	4745(923)	167100	84	3.94	14.20	40.52	21.03

NOR (Number of revisions) NOB (Number of bug fix revisions) LOC (Line of code)
NOM (Number of methods including bugs) ANR (Average number of recommendations per method)
P (Precision) R (Recall) F (F-measure)

Table I
EVALUATION OF NINE ECLIPSE PLUG-IN PROJECTS

In the case of Figure 3, *readFile* (before modification) and *JUnitTestRunMonitor* (before modification) correspond to *G1* and *G2*, respectively. The code snippet of the *JUnitTestRunMonitor* (after modification) is useful for fixing the bug of the *readFile* method. As shown in Figure 3, it is easy for programmers to fix bugs by referring to concrete code snippets. Currently, the *dcNavi* supports several concern queries such as *"Search Related API Usages"*, *"Search Test Cases"*, and *"Search Review Points"*.

V. EVALUATION

In this section, we evaluate the effectiveness of our approach in terms of the recommendation quality.

A. Test data and Criteria

We generated DCGs by using nine open source repositories created in the Eclipse plug-in projects. We selected a set of repositories related to Mylyn because we predicted that there were similar bug trends in the same domain. First, we collected 4/5 revisions from each project and all revisions of other eight projects as the training data. Next, using this data, we checked how many recommendations were provided to the bugs contained in the remaining 1/5 revisions of each project. After that, we compared correct bug fix set with recommendations provided by *dcNavi*.

We consider that a recommendation is correct when the *dcNavi* recommends a bug fix pattern that is used in the real bug fix. More concretely, we define the criteria of correct recommendation as follows: 1) the bug fix pattern applied to the real debugging is the same as that of recommendation; and 2) all the method calls related to the bug fix pattern must be included in the method to be modified. We think that a past bug fix dealing with the same method calls can be a hint for debugging and the applied pattern indicates the reason of the bug. In this evaluation, we used the value 0.20 as the minimum graph similarity metric. Precision becomes high if we use a large value as the similarity metric. On the other hand, recall becomes low in this case. The average F-measure in all nine projects becomes most favorable when the value of the similarity metric is 0.20.

B. Evaluation results

Table I shows the results of the recommendations. The rough precision (the fraction of the correct recommendations among all recommendations) ranges from 15% to 35%. The rough recall (the fraction of the correct recommendations among all correct bug fixes) ranges from 20% to 50%. In case of *qcMylyn* (HP Quality Center Mylyn Connector), 174 (= 223 * 4/5) revisions and 10,035 revisions (other eight projects) are used as the training data. There are 54 methods including bugs and the average number of the provided recommendations per method (ANR) is 5.70. Precision and recall are 34.42% and 50.24%, respectively.

C. Discussion

Our criteria of the correctness is slightly rigorous because all the method calls related to the bug fix pattern must be included in the method to be modified (see condition 2). However, it is effective to reuse the code with the same structure in which only the called methods are different (e.g., code templates). Although *dcNavi* recommends this kind of code, the code is not treated as a correct answer in our evaluation because the condition 2 is not satisfied. On the other hand, there may be too many recommendations if the condition 2 is deleted. We have to relax the condition 2 in order to regard this type of code as a correct answer.

Someone might think that the percentage of correct recommendations provided by *dcNavi* is not high. One reason is the rigorous criteria used in this evaluation as mentioned above. However, it is not easy to define the *correctness* in essence because the usefulness of recommendations varies according to the programmer's debug situation and skills.

To deal with this problem, we plan to introduce the recommendation options such as name matching and the value of minimum graph similarity metric. It would be better if programmers can specify the category of recommendations: API, exception handling, project-specific library, and so on. We think that the debug advice will be right to the point if the scope of the recommendation is explicitly specified. By introducing the recommendation options, we expect that we

can define the *correctness* suitable to each option. This is our future work.

VI. Related Work

Debugging is one of the crucial issues in software engineering and many researchers have proposed a variety of support tools such as debuggers and static analysis tools. Although the information provided by these tools is effective, they cannot directly advise programmers on the bug fixes. Recently, several recommendation systems for debugging have been proposed to deal with this problem.

The *Whyline* [6] is a debugging tool that allows programmers to ask *"Why did" and "Why didn't"* questions about the bugs. Programmers choose from a set of questions generated by static and dynamic analyses, and the tool provides answers of the bug causes. While *Whyline* uses static and dynamic analysis information of the target program, *dcNavi* uses not only program analysis information but also past bug fix information. We admit that fine-gained dynamic analysis information is effective for detecting the fault location and we plan to add the information to DCGs. If *Whyline* and *dcNavi* are integrated, we can provide more appropriate debug hints to the programmers.

The *DebugAdvisor* [1] is a search tool that allows programmers to express the context of the bugs and search through diverse data such as natural language text and core dumps. The *DebugAdvisor* allows programmers to search using a fat query, which could be kilobytes of structured and unstructured data. The *dcNavi*, which can be considered one of the search tools, explores structured DCGs focusing debugging. Although the goal of our approach is different from that of *DebugAdvisor*, we admit that large-scale searching is essential for the practical debug support.

The *FixWizard* [8] supports the tasks that identify the code peers existing in the program and recommend the similar fixes to its peers. Although *FixWizard* is similar to our approach, *dcNavi* focuses on the notion of *Concern* that can be related to a variety of debugging knowledge.

In *dcNavi*, the algorithm for searching the recommended methods is simple and general. It is effective that we can use the algorithm specific to the debug concern. The *CAR-Miner* [12] provides an approach for mining exception-handling rules as sequence association rules. These rules are specific to exception-handling.

The *BugMem* [4] provides a bug finding algorithm using bug fix memories consisting of project-specific bugs and fix knowledge. These bug fix memories use a learning process to extract project-specific bugs.

VII. Conclusions

This paper proposed a new concept, *debug recommendation based on Concern Graphs*, for providing the appropriate hints to programmers according to their debug concerns. Adopting our approach, programming, testing, and debugging can be systematically integrated based on the *Concern Graphs*. Although our approach is effective as demonstrated in Section 5, we have to improve the quality of the recommendation. As the first step, we plan to integrate *dcNavi* with runtime verification and dynamic analysis.

References

[1] Ashok, B., Joy, J., Liang, H., Rajamani, S. K., Srinivasa, G., and Vangala, V.: DebugAdvisor: A Recommender System for Debugging, In *Proceedings of the 7th joint meeting of the European Software Engineering Conference (ESEC) and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE)*, pp.373-382, 2009.

[2] Beck, K.: *Test-Driven Development: By Example*, Addison Wesley, 2002.

[3] Kersten, M. and Murphy, G. C.: Mylar: a degree-of-interest model for IDEs, In *Proceedings of the 4th International Conference on Aspect-oriented Software Development (AOSD 2005)*, pp.159-168, 2005.

[4] Kim, S., Pan, K., and Whitehead, E. E. J.: Memories of Bug Fixes, In *Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE 2006)*, pp.35-45, 2006.

[5] Ko, A. J., Aung. H., and Myers, B. A.: Eliciting Design Requirements for Maintenance-Oriented IDEs: A Detailed Study of Corrective and Perfective Maintenance Tasks, In *Proceedings of the 27th International Conference on Software Engineering (ICSE 2005)*, pp.126-135, 2005.

[6] Ko, A. J. and Myers, B. A.: Debugging Reinvented: Asking and Answering Why and Why Not Questions about Program Behavior, In *Proceedings of the 30th International Conference on Software Engineering (ICSE 2008)*, pp.301-310, 2008.

[7] Mylyn, http://www.eclipse.org/mylyn/.

[8] Nquyen, T. T., Nquyen, H. A., Pham, N. H., Al-kofahi, J., and Nquyen, T. N.: Recurring Bug Fixes in Object-Oriented Programs, In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering (ICSE 2010)*, pp.315-324, 2010.

[9] Pan, K., Kim, S. and Whitehead, Jr. E. J.: Toward an understanding of bug fix patterns, *Empirical Software Engineering*, vol. 14, no. 3, pp.286-315, 2009.

[10] Robillard, M. P. and Murphy, G. C.: Concern Graphs: Finding and Describing Concerns Using Structural Program Dependencies, In *Proceedings of the 24th International Conference on Software Engineering (ICSE 2002)*, pp.406-416, 2002.

[11] Robillard, M. P.: Automatic Generation of Suggestions for Program Investigation, In *Proceedings of the 5th Joint Meeting of the European Software Engineering Conference (ESEC) and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE)*, pp.11-20, 2005.

[12] Thummalapenta, S. and Xie, T.: Mining Exception-handling Rules as Sequence Association Rules, In *Proceedings of the 31st International Conference on Software Engineering (ICSE 2009)*, pp.496-506, 2009.

PAFL: Fault Localization via Noise Reduction on Coverage Vector

Lei Zhao
Computer School of
Wuhan University
Wuhan, China, 430072
zhaolei.whu@gmail.com

Zhenyu Zhang
State Key Laboratory of Computer Science,
Institute of Software, Chinese Academy of Sciences
Beijing, China, 100190
zhangzy@ios.ac.cn

Lina Wang
Computer School of Wuhan University, Key Laboratory of
Aerospace Information Security and Trust Computing
Wuhan, China, 430072
lnwang@whu.edu.cn

Xiaodan Yin
Computer School of
Wuhan University
Wuhan, China, 430072
yinxiaodan.whu@gmail.com

***Abstract*—Coverage-based fault localization techniques assess the extent of how much a program entity relates to faults by contrasting the execution spectra of passed executions and failed executions. However, previous studies show that different test cases may generate similar or identical coverage information in program execution, which makes the execution spectra of program entities indistinguishable to one another, thus involves noise and decreases the effectiveness of existing techniques. In this paper, we use the concept of coverage vector to model program coverage in execution, compare coverage vectors to capture the similarity among test cases, reduce noise by removing similar coverage vector to refine the execution spectra, and based on them assess the suspicious basic blocks being related to fault. We thus narrow down the search region and facilitate fault localization. The empirical evaluation using Siemens programs and realistic UNIX utilities shows that our technique effectively addresses the problem caused by similar test cases and outperforms existing representative techniques.**

Keywords-fault localization; execution path; noise reduction

I. INTRODUCTION

Coverage-based fault localization (CBFL) techniques have been proposed to support software debugging [8][11] [13]. By contrasting the coverage statistics of program entities (such as statements, blocks and predicates) between passed executions and failed executions, CBFL techniques can locate the program entities which exercising are strongly correlated to the program execution failures observed. Previous studies showed that CBFL techniques are effective in locating faults [8][13].

Since test cases may not always be generated to satisfy some coverage criteria, and there is no guarantee that the test suite reduction task is always conducted, it is common that different executions may cover similar and even identical execution paths [4]. Similar coverage information makes the execution spectra of program entities indistinguishable in passed and failed executions and thus decreases the effectiveness of previous fault localization techniques or even makes them lose effect, especially when coincidental correctness occurs [5]. For example, suppose a faulty statement is exercised in the program execution of all passed and all failed executions, it is hard to pinpoint it by contrasting its execution spectra in passed and failed executions. Such a case may have a high chance to happen in real life programs (e.g., a faulty statement may exist in the main method of a program and must be exercised by all the executions). Previous study also shows that execution similarity and coincidental correctness occurs frequently in realistic programs [12].

In this paper, we propose to use the concept of coverage vector to count the distinct execution paths, to capture the happening of execution similarity. We first calculate the failing rate of each coverage vector as the ratio of the number of failed executions covering it to the number of all executions covering it. For each basic block, we then calculate two numbers, the number of coverage vectors exercising that basic block and covered by failed executions, and the number of coverage vectors either exercising that basic block or covered by failed executions. We then use the ratio of the former to the latter as the suspiciousness score of that basic block. We next sort all the basic blocks in the descending order of thus calculated suspiciousness scores. In case of a tie, which means two or more blocks sharing identical suspiciousness scores, we try to use the average failing rate of coverage vectors exercising a block to further determine the order of the blocks.

We use seven Siemens programs and three UNIX utilities to evaluate our technique, and compare it with five representative techniques, namely, Tarantula [7], Jaccard [1], SBI [13], SAFL [4], and ICST10 [10]. The empirical results show that our technique is promising on the studies subject programs. Further analysis show that our technique is promising to alleviate the impact of execution similarity.

The contributions of this paper are twofold. (i) We propose to use the concept of coverage vector to count the distinct execution paths, and make use of it to estimate the occurring of similarities from the coverage information. (ii) We propose a new fault localization technique, PAFL, which is empirically evaluated to be promising in locating fault, especially in case of high execution similarity.

II. MOTIVATING EXAMPLE

In this section, we use an example to demonstrate previous techniques and motivate our approach.

The code excerpt in Figure 1 finds the middle value in three given numbers. A fault exists in statement s_2, which accesses the variable x instead of z. We choose six test cases to demonstrate in this example. The mark "•" in each cell indicates that a statement is exercised in the program

Blocks	Statements	Test cases						Previous techniques										Distinct paths				Our approach	
		t_1	t_2	t_3	t_4	t_5	t_6	Tarantula		Jaccard		SBI		SAFL		ICST10		p_1		p_2			
		F	*P*	*P*	*P*	***F***	*P*	score	rank	score	rank	score	rank	score	rank	score	rank	t_1	t_2,t_3,t_4	t_5	t_6	score	rank
	`Mid() {`																						
	s_1 `int x, y, z, m;`																						
	s_1 `read ("Enter 3 num:", x, y, z);`																						
b_1	s_2 `m=x; /* a mutant of m=z */`	•	•	•	•	•	•	0.50	4	0.33	4	0.50	4	0.81	6	0.50	4	•	•	•	•	0.50	2
	s_3 `if (y<z)`																						
b_2	s_4 `if (x<y)`					•	•	0.67	2	0.33	4	0.67	2	1	5	0.67	2			•	•	0.33	6
b_3	s_5 `m=y;`																						
b_4	s_6 `else if (x<z)`					•	•	0.67	2	0.33	4	0.67	2	1	5	0.67	2			•	•	0.33	6
b_5	s_7 `m=x;`																						
b_6	s_8 `else if (x>y)`	•	•	•	•			0.40	6	0.20	6	0.40	6	1	5	0.40	6	•	•			0.33	6
b_7	s_9 `m=y;`																						
b_8	s_{10} `else if (x>z)`	•	•	•	•			0.40	6	0.20	6	0.40	6	1	5	0.40	6	•	•			0.33	6
b_9	s_{11} `m=x;`																						
b_{10}	s_{12} `printf ("The middle is:", m);`	•	•	•	•	•	•	0.50	4	0.33	4	0.50	4	1	5	0.50	4	•	•	•	•	0.50	2
	`}`																						
	Code examining effort to locate fault:							66%		66%		66%		100%		66%						33%	

Figure 1. Motivating example – the program Mid

execution with respect to a test case. The execution results (*P* for pass and *F* for fail) are shown in table header.

We apply previous techniques Tarantula [7], Jaccard [1], SBI [13], SAFL [4], and ICST10 [10] to locate fault (statement s_2) in this example. For example, Tarantula assigns suspiciousness score 0.50 to statement s_2, and finally needs to examine 66% of all code to locate the fault. The results of Jaccard, SBI, SAFL, and ICST10 can be similarly explained. Unfortunately, none of them can locate the fault with somehow affordable code examining efforts (e.g., less than 50%). This is because that the faulty statement (s_2) happen to be exercised in all passed and failed executions, so that they are indistinguishable in execution spectra. On the other hand, Statements s_4 and s_6 are given higher suspiciousness scores than statements s_8 and s_{10} because the former happen to be exercised in relatively more failed executions than the latter. Note that statements s_4 and s_6 are exercised in one failed execution (t_5) and one passed execution (t_6), while statements s_8 and s_{10} are exercised in one failed execution (t_1) and three passed executions (t_2, t_3, and t_4). Among the two passed test cases (t_5 and t_6), which exercise the former (statements s_4 and s_6), 50% of them are failed ones. While only 25% of the test cases that exercise the latter (statements s_8 and s_{10}) are failed ones. As a result, statements s_4 and s_6 are given higher suspiciousness scores because of such imbalance.

Execution similarity may frequently occur in realistic programs [12]. We could not expect to always benefit from the imbalance observed in previous paragraph. To measure the execution similarity, we design to use the coverage information. In addition, we also measure the execution similarity to estimate the occurring of coincidental correctness in passed executions and manage to alleviate the impact of coincidental correctness to execution spectra of statements.

From Figure 1, we observe that there are in total two distinct paths covered by the six test cases [2], which are denoted as $p_1 = \langle b_1, b_2, b_4, b_{10}\rangle$ and $p_2 = \langle b_1, b_3, b_7, b_{10}\rangle$. After that, we adopt SBI's formula $\frac{\text{failed(p)}}{\text{failed(p)+passed(p)}}$ to estimate the failing rate of a path, which means the probability of an execution (covering that path) revealing a failure. Note that, here we use a path, rather than a statement, as the program entity in the formula. Since p_1 is covered by one failed test case and three passed test cases, the failing rate of p_1 is 0.25. Similarly, the failing rate of p_2 is 0.5. A path with a positive failing rate means that the corresponding execution can reveal failures. A path with failing rate zero means that the corresponding execution reveals no failure. We then adopt Jaccard's formula $\frac{\text{failed(b)}}{\text{totalfailed+passed(b)}}$ to calculate the suspiciousness score for each block. Here, *failed*(*b*) means the number of distinct paths, which have positive failing rates and exercise block *b*; *passed*(*b*) means the number of distinct paths, which have failing rates of zero and exercise block *b*; *totalfailed* means the number of distinct paths, which have positive failing rates. Thus, the suspiciousness score of block b_1 is calculated as $\frac{2}{2+2} = 0.50$. As a result, we finally take 33% code examining effort to locate the fault.

The above example has interestingly demonstrated that previous techniques may not be effective in a common case, while our approach has the potential to address it. In the next section, we will elaborate on our model.

III. OUR APPROACH

In this section, we introduce the problem settings, give definitions, and elaborate on our model PAFL.

A. Definitions

We use the definition of "coverage vector" to formally describe the concept of "distinct path" used in Section II.

[Definition I] *An **original coverage vector** $ocv_i = \langle b_1, b_2, ..., b_n\rangle$ ($b_j \in \{0, 1\}$ for $j = 1, 2, ..., n$) of program execution $P(t_i)$ is a tuple. We use $ocv_i(b_j)$* to retrieve the *j*-th element in the tuple, *where $ocv_i(b_j) = 1$ means the basic block b_j is exercised in the execution, $ocv_i(b_j) = 0$ means b_i is not exercised in the execution. For the coverage vector ocv_i with respect to execution $P(t_i)$, we also say $P(t_i)$ covers ocv_i.*

In Figure 1, there are six original coverage vectors. The coverage vector with respect to test case t_1 is $ocv_1 = \langle 1, 0, 0,$

0, 0, 1, 0, 1, 0, 1⟩. Apparently, an original coverage vector can be covered by many different executions (even by both some passed executions and some failed executions). So let us move to Definition II.

[Definition II] *The **distinct coverage vector set** $CV = \{cv_1, cv_2, ..., cv_p\}$ is the distinct set (with no repeating elements) of all original coverage vectors ocv_i with respect to the program execution $P(t_i)$ of each test case t_i. Each element $cv_i \in CV$ is called a **coverage vector**. Similarly, we use $cv_i(b_j)$ to retrieve the j-th element in the tuple of cv_i.*

By such definition, we know that we have $cv_i \neq cv_j$ for any two coverage vectors cv_i and cv_j ($1 \leq i < j \leq p$). In Figure 1, there are two coverage vectors, namely, cv_1 = ⟨1, 0, 0, 0, 0, 1, 0, 1, 0, 1⟩ and cv_2 = ⟨1, 1, 0, 1, 0, 0, 0, 0, 0, 1⟩.

B. Failing Rate of Coverage Vector

Since a coverage vector may be covered by both passed executions and failed executions, we are also interested in the ratio of failed executions that covers the coverage vector (to all executions that covers the coverage vector). We use the term *failing rate of a coverage vector* to denote such a ratio, which is calculated using equation (1).

$$\theta(cv_i) = \frac{failed(cv_i)}{failed(cv_i) + passed(cv_i)} \quad (1)$$

In equation (1), *failed* (cv_i) and *passed* (cv_i) respectively refer to the number of failed and passed executions that cover cv_i. They are calculated using equation (2) and (3).

$$failed(cv_i) = \left|\{v_j | P(v_j) \text{ covers } cv_i\}\right| \quad (2)$$

$$passed(cv_i) = \left|\{u_j | P(u_j) \text{ covers } cv_i\}\right| \quad (3)$$

Here, we adopt the formula of SBI in equation (1) because it gives a best estimation to the probability of an exercised program entity causing a failure [14].

For a coverage vector cv_i with $\theta(cv_i)$ greater than zero, it indicates that cv_i is covered by at least one failed execution, and it is also denoted as a *failed coverage vector*. For a coverage vector cv_i with $\theta(cv_i)$ equals to zero, it indicates that cv_i is covered by no failed execution, and it is also denoted as a *passed coverage vector*.

According to equation (1), the failing rates of cv_1 and cv_2 are $\theta(cv_1) = 0.25$ and $\theta(cv_2) = 0.50$, respectively. Both of them are failed coverage vectors.

C. Suspiciousness Scores of Blocks

After we have identified all the coverage vectors, we also need to calculate suspiciousness scores for basic blocks. Inspired by previous study [6], we employ the Jaccard similarity coefficient to evaluate the suspiciousness scores for basic blocks, by contrasting the execution spectra of basic blocks on the coverage vector level. In this paper, we use the term $susp(b_i)$ to denote such suspiciousness score of basic block b_i, which is calculated using equation (4).

$$susp(b_i) = \frac{\left|\{cv_j | \theta(cv_j) > 0 \wedge cv_j(b_i) = 1\}\right|}{\left|\{cv_j | \theta(cv_j) > 0 \vee cv_j(b_i) = 1\}\right|} \quad (4)$$

The numerator represents the number of failed coverage vectors that cover b_i, the denominator represents the number of coverage vectors that are either failed coverage vectors or cover b_i. Here, we adopt the similarity coefficient Jaccard because it has mature mathematical basis. Further, it has been used in previous techniques and empirically shown effective in locating faults in programs [1].

Equation (4) estimates the extent of how much a basic block is related to faults. The greater the value, the more the basic block will be related to fault. According to equation (4), we can recall the motivating example in Section II and revisit the suspiciousness scores calculated in Section II. The suspiciousness scores for b_1 is calculated as $\frac{2}{2+2} = 0.50$.

D. Tie breaking

After all the blocks are sorted according to their suspiciousness of relating to fault and form a list, programming may search along the generated list for the fault. Particularly, when some basic blocks have identical suspiciousness scores, we use equation (5) to break tie.

$$conf(b_i) = \frac{\sum_{cv_j(b_i)=1}[\theta(cv_j)]}{\left|\{cv_j | \theta(cv_j) > 0, cv_j(b_i) = 1\}\right|} \quad (5)$$

Equation (5) calculates the average failing rate of the coverage vectors that exercising basic block b_i. The rational is that for two basic blocks having identical probability of causing failure, we deem the one whose appearance in a path has higher chance to reveal a failure as more related to faults.

For example, in Figure 1, basic blocks b_1 and b_{10} form a tie. We calculate that $conf(b_1) = conf(b_{10}) = \frac{0.25+0.5}{2} = 0.375$ so that the tie still cannot be break and thus b_1 and b_{10} are evaluated as a whole. Finally, we need to examine 33% of all code to locate the fault.

IV. EVALUATION

A. Experiments Setup

In this paper, we use the 7 Siemens programs and 3 UNIX utilities to evaluate our technique. Each of them has several faulty versions (downloaded from the SIR repository [3]). They have been used in previous studies [9][13][14]. Table 1 shows the statistics of the subject programs used in the experiments.

In our experiment, we select techniques Tarantula [7], Jaccard [1], SBI [13], SAFL [4], and ICST10 [10] to compare with. Tarantula is an old technique and has a lot of variants [7][13]. Jaccard is evaluated very effective in

Table 1. Statistics of subjects

Subjects	# of faulty versions	# of test cases	Description
print_tokens	7	4130	lexical analyzer
print_tokens2	10	4115	lexical analyzer
replace	32	5542	pattern replacement
schedule	9	2650	priority scheduler
schedule2	10	2650	priority scheduler
tcas	40	1578	altitude separation
tot_info	23	1054	information measure
flex	56	567	lexical parser
grep	21	809	text processor
gzip	18	213	compressor
in total	226		

previous studies [10][14]. SBI is the statement-level version of CBI [9], while the latter is a classic predicate-level technique. SAFL and ICST10 investigate execution similarity to reduce the noise from coincidental correctness and relates to them.

B. *Effectiveness on Subject Programs*

To know the overall effectiveness of the studied techniques, we take the average of the 10 programs to show in Figure 2. In Figure 2, the x-coordinates mean the percentage of code examined in each faulty version; the y-coordinates show the percentage of faulty versions, in which, faults can be located within the code examining effort specified by the x-coordinates.

From Figure 2, we observe that at most checkpoints (except the 50% checkpoint), PAFL is more effective than, or at least comparable to, the other techniques. For example, on average, by examining up to 5% of all the code in faulty versions, PAFL can locate faults in 34% of all faulty versions, Jaccard can locate 33%, Tarantula can locate 23%, SBI can locate 22%, SAFL can locate 6%, and ICST10 can locate 26%. It shows that PAFL has an overall better effectiveness than the other techniques studied.

Table 2 shows the mean effectiveness of these techniques on each program. Limited by the space, we cannot show results of the minimum, maximum, and standard deviation measurements. This table shows that for these programs, PAFL is often, but not always, the best among the four techniques.

V. CONCLUSION

In this paper, we demonstrate that frequently occurred execution similarity may affect the effectiveness of existing fault localization techniques and propose PAFL to alleviate the impact of coincidental correctness. We use the concept of coverage vector to count distinct execution paths, and calculate failing rate for each coverage vector, thus refine the executions spectra to reduce noise. The empirical study shows that our technique outperforms five previous techniques on Siemens and UNIX programs. The experiment also shows that our technique is particularly effectiveness to alleviate the impact of execution similarity and coincidental correctness. The future work is to investigate the impact of test case selection to PAFL and adapt PAFL to locate faults in multi-fault programs.

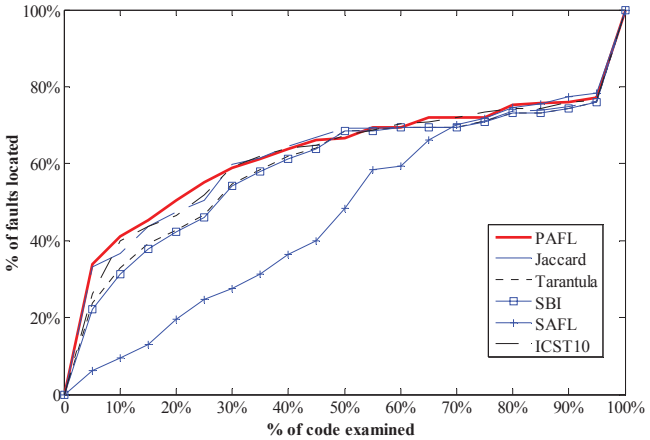


Figure 2. Overall effectiveness

Table 2. Mean effectiveness on individual programs

Subjects	PAFL	Jaccard	Tarantula	SBI	SAFL	ICST10
print_tokens	**72%**	74%	77%	77%	84%	74%
print_tokens2	**22%**	24%	25%	25%	55%	24%
replace	24%	**21%**	24%	24%	37%	**21%**
schedule	**23%**	24%	25%	25%	53%	24%
schedule2	85%	85%	85%	85%	**82%**	84%
tcas	54%	56%	58%	58%	66%	**51%**
tot_info	**37%**	43%	47%	47%	64%	43%
flex	**27%**	**27%**	30%	32%	45%	30%
grep	**21%**	**21%**	23%	26%	34%	23%
gzip	**12%**	**12%**	14%	15%	18%	14%

ACKNOWLEDGMENTS

This work is supported by the National Natural Science Foundation of China (nos. 60970114, 61003027, 61073006) and the Scholarship Award for Excellent Doctoral Student granted by Chinese Ministry of Education.

REFERENCES

[1] R. Abreu, P. Zoeteweij, and A.J.C.van Gemund. On the accuracy of spectrum-based fault localization. I*n Proc. of Testing: Academic and Industrial Conference, Practice and Research Techniques*.

[2] B Baudry, F Fleurey, and Y. Traon. Improving Test Suites for Efficient Fault Localization. In *Proc. of ICSE'06*.

[3] H. Do, S. G. Elbaum, and G. Rothermel. Supporting controlled experimentation with testing techniques: an infrastructure and its potential impact. *Empirical Software Engineering*, 2005.

[4] D. Hao, L. Zhang, Y. Pan, H. Mei, and J. Sun. On similarity-awareness in testing-basedfault localization. *JASE*, 2008.

[5] R. M. Hierons. Avoiding coincidental correctness in boundary value analysis. *TOSEM.*, 2006.

[6] P. Jaccard. Étude comparative de la distribution florale dans une portion des Alpes et des Jura. Bulletin del la Socit Vaudoise des Sciences Naturelles 37.

[7] J. A. Jones and M. J. Harrold. Visualization of test informationto assistfault localization. In *Proc. of ICSE'02*.

[8] J. A. Jones and M. J. Harrold. Empirical evaluation of the tarantula automaticfault-localization technique. In *Proc. of ASE'05*.

[9] B. Liblit, A. Aiken, A. Zheng, and M. I. Jordan. Bug isolation via remote program sampling. In *Proc. of PLDI'03*.

[10] W. Masri, and R. Assi. Cleansing Test Suites from Coincidental Correctness to Enhance Fault-Localization. In *Proc. of ICST'10*.

[11] R. Santelices, J. A. Jones, Y. Yu, and M. J. Harrold. Lightweightfault localization using multiple coverage types. In *Proc. of ICSE'09*.

[12] X. Wang, S. C. Cheung, W. K. Chan, and Z. Zhang. Taming coincidental correctness: refine code coverage with context pattern to improve fault localization. In *Proc. of ICSE'09*.

[13] Y. Yu, J. A. Jones, and M. J. Harrold. An empirical study of the effects of test-suite reduction on fault localization. In *Proc. of ICSE'08*.

[14] Z. Zhang, W. K. Chan, T. H. Tse, B. Jiang, and X. Wang. Capturing propagation of infected program states. In *Proc. of FSE/ESEC'09*.

Using Coverage and Reachability Testing to Improve Concurrent Program Testing Quality

Simone R. S. Souza[1], Paulo S. L. Souza[1], Mario C. C. Machado[1],
Mário S. Camillo[1], Adenilso Simão[1] and Ed Zaluska[2]
[1] Instituto de Ciências Matemáticas e de Computação – Universidade de São Paulo
P.O. 668 – São Carlos – Brasil – 13560-970
{srocio, pssouza,mmachado, adenilso}@icmc.usp.br, mariocamillo@gmail.com
[2] Eletronics and Computer Science – University of Southampton
ejz@ecs.soton.ac.uk

Abstract

The testing of concurrent software is a challenging task. A number of different research approaches have investigated adaptation of the techniques and the criteria defined for sequential programs. A major problem with the testing of concurrent software that persists is the high application cost due to the large number of the synchronizations that are required and that must be executed during testing. In this paper we propose a complementary approach, using reachability testing, to guide the selection of the tests of all synchronization events according to a specific coverage criterion. The key concept is to take advantage of both coverage criteria, which are used to select test cases and also to guide the execution of new synchronizations, and reachability testing, which is used to select suitable synchronization events to be executed. An experimental study has been conducted and the results indicate that it is always advantageous to use this combined approach for the testing of concurrent software.

1. Introduction

Concurrent applications are inevitably more complex than sequential ones and, in addition, all concurrent software contains features such as nondeterminism, synchronization and inter-process communication which significantly increase the difficulty of validation and testing.

For sequential programs, many testing problems were simplified with the introduction of testing criteria and the implementation of supporting tools. A testing criterion is a predicate to be satisfied by a set of test cases which can be used as a template for the generation of test data [10].

Extending previous work on sequential program structural testing criteria, we have proposed structural testing criteria for the validation of concurrent programs, applicable to both message-passing software [14] and shared-memory software [12]. These testing criteria are designed to exploit information about the control, data and communication flows of concurrent programs, considering both sequential and parallel aspects.

The use of these criteria significantly improves the quality of the test cases, providing a coverage measure that can be used in two important testing procedures. In the first one, the criteria can be used to guide the generation of test cases, where the criteria are used as guideline for test data selection. The second testing procedure is related to the evaluation of a test set; in this case, the criteria can be used to determine when the testing activity can be terminated based on sufficient coverage of the required elements. The main contribution of the proposed testing criteria is to provide an efficient coverage measure for evaluating the progress of the testing activity and the quality of test cases.

This approach uses static analysis of the program under test to extract relevant information for testing, which is straightforward to apply and generate relevant information for coverage testing. The problem is the large number of infeasible elements generated that must be analyzed. An element is infeasible if there is no set of values for the parameters (the input and global variables) that cover that element. Complete determination of infeasible elements is an extremely difficult problem and it is not possible to determine them automatically.

Lei and Carver [7] proposed a method (based on reachability testing) to obtain all of the executable synchronizations of a concurrent program (from a given execution of the program) in a way that reduces the number of redundant synchronizations. As this method uses dynamic information, only feasible synchronizations are generated, which

is a considerable advantage. However, a difficulty with this method is the high number of possible combinations of synchronization that are generated. For complex programs, this number is very high, which limits the practical application of this approach. In [1], Carver and Lei proposed a distributed reachability testing algorithm, allowing different test sequences be executed concurrently. This algorithm reduces the time to execute the synchronizations, but the authors do not comment about the effort necessary to analyze the results from these executions.

Lei and Carver's [7] method is essentially complementary to our approach. They do not address how to select the test case which will be used for the initial run, while we use the static analysis of the program to select the optimum test cases in advance.

In this paper we propose a complementary approach, using reachability testing to target coverage testing for synchronization events. The idea is to take appropriate advantage of both approaches: information about synchronizations provided by the coverage criterion are used to decide which race variants will be executed, selecting only synchronizations that have not already been covered by existing test cases. It is therefore possible to execute each synchronization at least once and to use reachability testing to select only those synchronizations that are feasible.

This paper is structured as follows. In Section 2 we describe related work on the testing of concurrent software, presenting more details on the coverage testing and reachability testing approaches. In Section 3 we present the test strategy proposed in this paper. In Section 4, an experimental study to evaluate our test strategy is presented and the results obtained are discussed. Finally, in Section 5 we present our conclusions together with future work.

2. Concurrent Program Testing

Traditional testing techniques are often not well-suited to the testing of concurrent or parallel software, in particular when nondeterminism and concurrency features are significant. Many researchers have developed specific testing techniques addressing such issues and in addition there have been initiatives to define suitable testing criteria [16, 18, 17, 8, 11, 15]. The detection of race conditions and mechanisms for replay testing have also been investigated [4, 7, 2, 3].

Yang [18] describes a number of challenges for the testing of parallel software: 1) developing static analysis; 2) detecting unintentional races and deadlock in nondeterministic programs; 3) forcing a path to be executed when nondeterminism might exist; 4) reproducing a test execution using the same input data; 5) generating the control flow graph for nondeterministic programs; 6) providing a testing framework as a theoretical base for applying sequential testing criteria to parallel programs; 7) investigating the applicability of sequential testing criteria to parallel program testing; and 8) defining test coverage criteria based on control and data flow.

Lei and Carver [7] proposed the *reachability testing* for generating all feasible synchronization sequences (and only them). This method guarantees that every partially-ordered synchronization will be exercised exactly once without repeating any sequences that have already been exercised. The method involves the execution of the program in a semi-deterministic way; the execution is deterministic up to a given point, from which it runs nondeterministically. The resulting synchronization sequence (sync-sequence), which is feasible, is analyzed and a new feasible sequence (if possible) is computed. The authors employ a reachability schema to calculate the synchronization sequence automatically. The reachability testing uses dynamic information to execute all feasible synchronization sequences, generating all *race variants* from one particular execution.

The reachability testing process is illustrated in Figure 1 (extracted from [7]). The figure shows a space-time diagram in which vertical lines represent four threads of a concurrent program. The interaction between processes is represented by arrows from a send event to a receive event. Diagram Q_0 shows the one execution of the program, generating the synchronizations: (s_1^{T1}, r_1^{T2}), (s_2^{T4}, r_2^{T2}), (s_3^{T2}, r_3^{T3}), (s_4^{T4}, r_4^{T3}). V_1, V_2 and V_3 are *race variants* of Q_0 and feasible executions generated during reachability testing execution. A problem here is the high number of possible combination of synchronization that are generated and (for complex software) this number can be very high, restricting any practical application of the strategy. This approach has the important advantage that it will generate only feasible synchronization sequences, which is an important consideration when reducing the cost of the testing activity.

2.1. Structural Testing for Concurrent Programs

In this section we describe our test model and criteria for validation of message-passing software [14]. The test model captures control, data and communication information. The model considers that a fixed and known number of processes n is created at the initialization of the concurrent application. These processes may each execute different programs. However, each one executes its own code in its own memory space. The concurrent program is defined by a set of n parallel processes $Prog = \{p^0, p^1, \ldots p^{n-1}\}$. Each process p has its own Control Flow Graph CFG^p, that is built using the same concepts as traditional software [10]. In other words, a CFG of a process p is composed of a set of nodes N^p and a set of edges E^p. Each node n in the process p is represented by the notation n^p and corresponds to a set of commands that are sequentially executed or can be

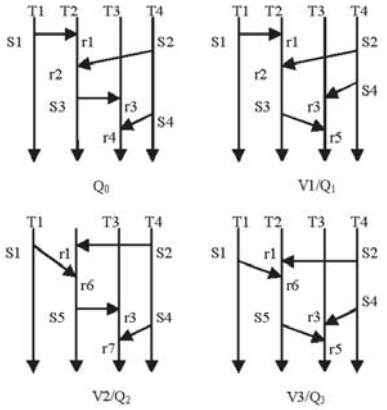


Figure 1. Example of reachability testing [7]

associated to a communication primitive (send or receive). The model considers both blocking and non-blocking receives, such that all possible interleaving between send-receive pairs can be represented. $Prog$ is associated with a Parallel Control Flow Graph ($PCFG$), which is composed of the CFG^p (for $p = 0 \ldots n-1$) and by the representation of the communication between the processes. A synchronization edge (sync-edge) (n_i^a, n_j^b) links a *send* node in a process a to a *send* node in a process b. These edges represent the possibility of communication and synchronization between processes.

A set of coverage testing criteria is defined, based on ($PCFG$): All-Nodes; All-Edges; All-Nodes-R; All-Nodes-S and All-Edges-S (related to control and synchronization information) and All-C-Uses; All-P-Uses; All-SUses; All-S-C-Uses and All-S-P-Uses (related to data and communication information) [14]. The All-Edges-S criterion requires that the test set executes paths that cover all the sync-edge associations of the concurrent program under testing; the All-S-uses criterion requires that the test set executes paths that cover all the *s-use* associations. An *s-use* is an association between a node n^p, that contains a definition of a variable x, and a sync-edge that contains a communications use of x.

An example of a $PCFG$ is shown in Figure 2. There are four processes, consisting of two different codes. Synchronization pairs are represented by dotted lines — for example, the pair $(2^0, 2^m)$ is one sync-edge between process p^0 and p^m. Each sync-edge is associated with one or more s-use associations, and related to a variable represented in $PCFG$.

Nondeterminism is the key issue addressed in this test model. As it is impossible to determine statically when a synchronization is feasible, a conservative approach is assumed, where every pair of send and receive events which have the appropriate types are considered

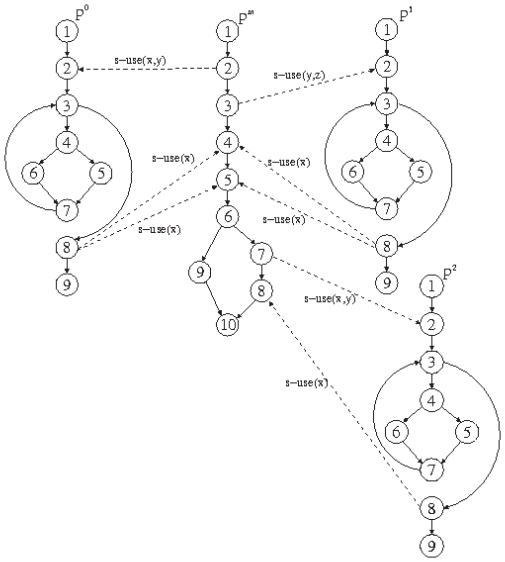


Figure 2. Example of a Parallel Control Flow Graph

as a possible matching. Considering the example of Figure 1, the required sync-edges are: (s_1^{T1}, r_1^{T2}), (s_2^{T4}, r_1^{T2}), (s_4^{T4}, r_1^{T2}), (s_1^{T1}, r_2^{T2}), (s_2^{T4}, r_2^{T2}), (s_4^{T4}, r_2^{T2}), (s_1^{T1}, r_3^{T3}), (s_2^{T4}, r_3^{T3}), (s_3^{T2}, r_3^{T3}), (s_4^{T4}, r_3^{T3}), (s_1^{T1}, r_4^{T3}), (s_2^{T4}, r_4^{T3}), (s_3^{T2}, r_4^{T3}), (s_4^{T4}, r_4^{T3}). Some sync-edges are infeasible (e.g., (s_1^{T1}, r_3^{T3})) but are required. Using controlled execution [2], it is possible to force the execution of feasible sync-edges.

It is not necessary to execute all combinations of possible synchronization as long as at least one execution of each sync-edge pair is included. A problem in this approach is the high number of infeasible sync-edges that are generated and need to be analysed. Nevertheless, it is interesting because it uses information generated statically to direct the selection of test cases and to assess the coverage of the program under test. We believe that the choice of the test case can influence the results obtained, improving the overall testing activity quality. This has led directly to the test strategy presented in the next section.

3. Proposed test strategy

In this paper we propose a test strategy that combines both reachability testing and coverage testing to execute synchronization events. The main motivation of this strategy is to improve coverage testing; however, the approach can also be applied to improve reachability testing perfor-

mance. In this case, reachability testing can be applied using an approach that selectively exercises a set of sync-sequences according to a specified coverage testing criterion. Exhaustive testing is not always practical and they pointed out the need to use mechanisms to guide the selection of the sync-sequences during reachability testing [7].

In Figure 3 the proposed test strategy is illustrated. This figure does not show all the steps necessary to apply the coverage testing criterion, only those important to our strategy.

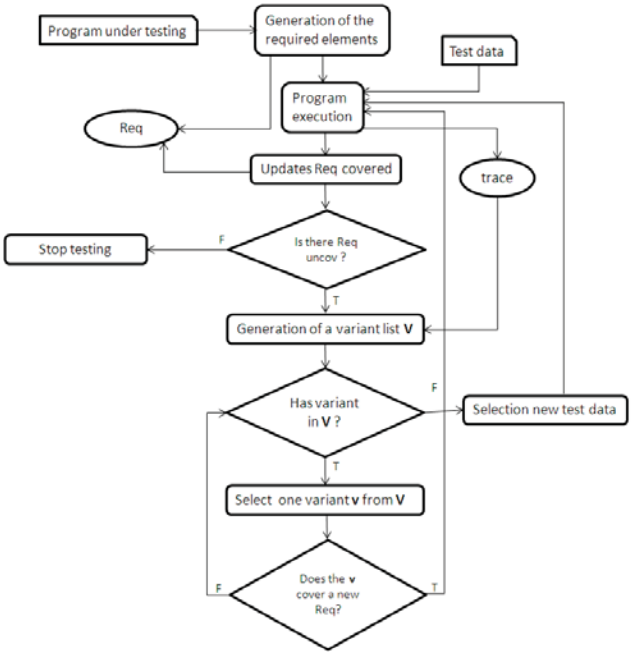


Figure 3. Test Strategy using Reachability Testing and Coverage Testing Criterion

First, a required elements list Req is generated from a given concurrent program, based on the all-s-uses and all-edges-s criteria. An initial test dataset is produced and the program is executed to generate an execution trace, containing a record of all the nodes and the sync-edges executed by the test dataset. The elements covered by the test dataset execution are marked in Req and any required element not yet covered identified. Usually, a procedure is used to select a new test dataset to improve the coverage of Req.

Considering reachability testing in this context, the next step is the generation of a list V of the variants, based on the sync-edges executed. For each sync-edge all possible variants are then generated. The difference here from reachability testing, is that only the variants required to cover a required element that is not yet covered are included. Therefore, when a new variant v is selected from V, it is verified only if it executes a new requirement of Req. Otherwise, another variant is selected or a new test dataset is generated (when V is empty). The procedure to execute a variant v is the same as that defined by Lei and Carver [7]: the controlled execution ensures that the synchronization of the v always occurs during the execution. After execution of the variant v, the execution trace is obtained and the required elements covered for this execution are marked in list Req. Considering now the variant v, new variants are generated and added to the list of the variants V. The procedure to execute variants or new test datasets is repeated while any required elements remain to be covered.

4. Experimental Study

In this section, we present an experimental study that indicates that the approach combining reachability and coverage criteria testing improve overall testing quality. The ValiMPI tool was used to conduct this study. ValiMPI is a tool developed to test concurrent programs implemented in MPI (Message Passing Interface), proposed originally to support the coverage testing mentioned in Section 2 [13, 6]. ValiMPI functionality has been extended to implement the reachability testing strategy proposed by Lei and Carver [7] and hence test the strategy proposed in this paper.

Eight different MPI programs were used in this study, implementing classical concurrency algorithms. The complexity is given by the number of sends s and receives r of each program: **sieve of Eratosthenes** - (7s and 9r) an algorithm for finding all prime numbers up to a specified integer [9]; **gcd** - (7s and 7r) to calculate the greatest common divisor of three numbers, using successive subtractions between two numbers until one of them is zero; **mmult** - (15s and 27r) to implement matrix multiplication using domain decomposition; **philosophers** - (11s and 10r) to implement the dining philosophers problem; **pairwise** - (16s and 16r) where each process n_i receives a data X_i and is responsible for computing the interactions $I(X_i, X_j)$,for $i \neq j$. For this, a structure with N channels is used, where each communication channel represents a pair source-destination — these channels are used to connect the N tasks into a unidirectional ring; **reduction** - (4s and 4r) to implement the reduction operation of distributed data, considering add, multiplication, greater than and less than operations; **qsort** - (28s and 52r) to implement quicksort, based on the parallel algorithm presented in Grama [5]; and **jacobi** - (23s and 37r) to implement Jacobi-Richardson iteration for solving a linear system of equations.

Three different test scenarios were executed:

1. **Selection of adequate test case using coverage criteria (CovT)**: using the criteria all-s-uses and all-edges-s, test cases were manually generated to exercise the required elements of these criteria, s-use associations

and sync-edges, respectively. Infeasible elements were identified to evaluate the coverage of an initial test set.

2. **Application of reachability testing (RT)**: using the initial test set generated during Scenario CovT, reachability testing was undertaken, according to the algorithm proposed by Lei and Carver [7].

3. **Application of our test strategy (RTCovT)**: using the criteria all-s-uses and all-edges-s, (related to synchronization), reachability testing was executed guided by the required elements of these criteria, following the steps discussed in Section 3.

Table 1 presents the sync-sequences generated by Reachability Testing (column RT) and by our test strategy (column RTCovT), using two different test sets: T_1, generated by CovT and containing test cases adequate to execute both sync-edges and s-uses; and T_2, which is a subset of T_1 containing only effective test cases (i.e. T_2 contains only test case contributing to the execution of the sync-edges). RT executes, for each test case, all variants of the each sync-edge, even those variants already executed previously. For this reason, the number of the sync-sequences generated by RT is higher than the sync-sequences generated by our test strategy. These results indicate that is possible to reduce the cost of the reachability testing using coverage testing. A fundamental problem with reachability testing is to decide when the testing activity can be considered complete; our test strategy contributes to the work on this problem. We compared the advantages of using our test strategy compared to the alternatives discussed previously.

Table 1. Number of the sync-sequences executed

Programs	T1	T1		T2	T2	
		RT	RTCovT		RT	RTCovT
sieve	10	60	13	4	20	7
gcd	13	24	14	7	14	9
mmult	8	48	11	4	24	5
philosophers	1	1680	2	1	1680	2
pairwise	4	4	4	1	1	1
reduction	4	4	4	1	1	1
qsort	3	794	18	3	266	16
jacobi	9	1710	13	6	377	11

Table 2 shows the results of the coverage obtained using the coverage testing (CovT) and our test strategy (RTCovT) for all-edges-s and all-s-uses criteria. For this analysis, for each program, the same test set T (generated on an *ad-hoc* basis) was used to execute the two test scenarios and the two coverage criteria. Our test strategy (RTCovT) indicates the potential to improve the criteria coverage because our strategy executes a greater number of sync-edges and s-uses than the traditional coverage testing, establishing that it is a good strategy to reduce the overall application cost of the test. Some of the programs in the test had no improvement in coverage because in these cases the T set already covered all feasible elements for the criteria.

Table 2. Coverage using coverage criteria and the test strategy

Programs	All-Edges-S		All-S-Uses	
	CovT	RTCovT	CovT	RTCovT
sieve	80.95%	90.48%	56.67%	76.67%
gcd	100.00%	100.00%	80.00%	85.00%
mmult	93.33%	93.33%	94.74%	94.74%
philosophers	100.00%	100.00%	75.00%	75.00%
pairwise	100.00%	100.00%	83.33%	83.33%
reduction	100.00%	100.00%	100.00%	100.00%
qsort	51.61%	89.25%	43.54%	67.35%
jacobi	100.00%	100.00%	84.06%	85.51%

Table 3 shows the results for the Jacobi algorithm example as different test cases ($tc1$ to $tc9$) are processed. Our testing strategy always provides better test coverage and maximal coverage is achieved after only five test cases have been considered. Table 4 provides similar results for the mmult algorithm example with test cases $tc1$ to $tc8$. For this example maximum coverage is achieved after only two of the test cases have been considered.

Table 3. Evolution of the jacobi program coverage

testcases	All-Edges-S		All-S-Uses	
	CovT	RTCovT	CovT	RTCovT
$tc1$	19.30%	19.30%	8.70%	8.70%
$tc2$	38.60%	49.12%	26.09%	34.78%
$tc3$	82.46%	94.74%	60.87%	71.01%
$tc4$	84.21%	96.49%	68.12%	78.26%
$tc5$	94.74%	100.00%	78.26%	82.61%
$tc6$	94.74%	100.00%	78.26%	82.61%
$tc7$	94.74%	100.00%	78.26%	82.61%
$tc8$	100.00%	100.00%	82.61%	84.06%
$tc9$	100.00%	100.00%	84.06%	85.51%

5. Conclusions

In this paper we have presented a new test strategy to validate concurrent programs, using a combination of coverage criteria and reachability testing. The coverage criteria are used both to select test cases and to determine the execution of new synchronizations, while the reachability testing is used to select appropriate synchronizations to be executed.

Table 4. Evolution of the mmult program coverage

testcases	All-Edges-S		All-S-Uses	
	CovT	RTCovT	CovT	RTCovT
$tc1$	60.00%	93.33%	63.16%	89.47%
$tc2$	60.00%	93.33%	68.42%	94.74%
$tc3$	73.33%	93.33%	78.95%	94.74%
$tc4$	86.67%	93.33%	89.47%	94.74%
$tc5$	86.67%	93.33%	89.47%	94.74%
$tc6$	86.67%	93.33%	89.47%	94.74%
$tc7$	86.67%	93.33%	89.47%	94.74%
$tc8$	86.67%	93.33%	94.74%	94.74%

The combination of the two approaches has the potential to deliver significant reduction in the overall testing cost. Due to the high number of synchronizations in a typical concurrent program, the execution of these synchronizations using reachability testing alone can be impractical; while in the case of the coverage criteria used by itself, these synchronizations generate a high cost because of the number of infeasible synchronizations that must be analyzed.

The test strategy described in this paper contributes in two ways: 1) by using structural criteria to minimize the number of sequences in reachability testing; and 2) by guiding the generation of test cases based on the structural criteria, using reachability testing to increase the test coverage. An experimental study has been undertaken to evaluate this approach. The results indicate that is promising to adopt this test strategy, with an improvement in test coverage in every case considered.

Finally, we plan to evaluate the proposed test strategy in terms of revealing faults. Preliminary results have demonstrated that our strategy is effective in detecting faults. The test sets discussed above were evaluated using the fault taxonomy presented in [4] and 84.8% of the seeded defects were revealed on average. Further studies are being developed using different fault taxonomies and comparing the effectiveness of our strategy with the effectiveness of reachability testing.

6 Acknowledgments

The authors would like to thank CAPES and FAPESP, Brazilian funding agencies, for the financial support, under Capes process 1191/10-1 and FAPESP processes: 2008/04614-5, 2010/02839-0.

References

[1] R. Carver and Y. Lei. Distributed reachability testing of concurrent programs. *Concurrency and Computation: Practice and Experience*, 22(18):2445–2466, 2010.

[2] R. Carver and K.-C. Tai. Replay and testing for concurrent programs. *IEEE Software*, pages 74–86, Mar. 1991.

[3] S. K. Damodaran-Kamal and J. M. Francioni. Nondeterminacy: Testing and debugging in message passing parallel programs. In *3rd ACM/ONR Workshop on Parallel and Distributed Debugging*, pages 118–128, New York, May 1993.

[4] O. Edelstein, E. Farchi, E. Goldin, Y. Nir, G. Ratsaby, and S. Ur. Framework for testing multi-threaded java programs. *Concurrency and Computation: Practice and Experience*, 15(3-5):485–499, 2003.

[5] A. Grama, G. Karypis, V. Kumar, and A. Gupta. *Introduction to Parallel Computing*. Addison Wesley, 2003.

[6] A. C. Hausen, S. R. Vergílio, S. Souza, P. Souza, and A. Simão. A tool for structural testing of MPI programs. In *8th IEEE Latin-American Test Workshop*, march 2007.

[7] Y. Lei and R. H. Carver. Reachability testing of concurrent programs. *IEEE TSE*, 32(6):382–403, June 2006.

[8] S. Lu, W. Jiang, and Y. Zhou. A study of interleaving coverage criteria. In *Proceedings of the ACM SIGSOFT symposium on the foundations of software engineering*, pages 533–536, New York, NY, USA, 2007. ACM.

[9] M. J. Quinn. *Parallel Computing : Theory and Practice*. McGraw-Hill, New York, 2nd. edition, 1994.

[10] S. Rapps and E. J. Weyuker. Selecting software test data using data flow information. *IEEE Transaction Software Engineering*, 11(4):367–375, Apr. 1985.

[11] C. Robinson-Mallett, R. M. Hierons, J. Poore, and P. Liggesmeyer. Using communication coverage criteria and partial model generation to assist software integration testing. *Software Quality Control*, 16(2):185–211, 2008.

[12] F. S. Sarmanho, P. S. L. Souza, S. R. S. Souza, and A. S. Simão. Structural testing for semaphore-based multithread programs. In *International Conference on Computational Science, LNCS*, volume 5101, pages 337–346, 2008.

[13] S. R. S. Souza, S. R. Vergilio, P. S. L. Souza, A. S. Simão, T. G. Bliscosque, A. M. Lima, and A. C. Hausen. Valipar: A testing tool for message-passing parallel programs. In *International Conference on Software knowledge and Software Engineering (SEKE05)*, pages 386–391, Taipei-Taiwan, 2005.

[14] S. R. S. Souza, S. R. Vergilio, P. S. L. Souza, A. S. Simão, and A. C. Hausen. Structural testing criteria for message-passing parallel programs. *Concurrency and Computation: Practice and Experience*, 20:1893–1916, mar 2008.

[15] J. Takahashi, H. Kojima, and Z. Furukawa. Coverage based testing for concurrent software. In *28th International Conference on Distributed Computing Systems Workshops, 2008.*, pages 533–538, June 2008.

[16] R. N. Taylor, D. L. Levine, and C. Kelly. Structural testing of concurrent programs. *IEEE Transaction Software Engineering*, 18(3):206–215, Mar. 1992.

[17] W. E. Wong, Y. Lei, and X. Ma. Effective generation of test sequences for structural testing of concurrent programs. In *10th IEEE International Conference on Engineering of Complex Systems (ICECCSt'05)*, pages 539– 548, 2005.

[18] C.-S. D. Yang. *Program-Based, Structural Testing of Shared Memory Parallel Programs*. PhD thesis, University of Delaware, 1999.

Program slicing spectrum-based software fault localization*

Wanzhi Wen, Bixin Li, Xiaobing Sun, Jiakai Li

School of Computer Science and Engineering, Southeast University, Nanjing 211189, China
Key Lab of Computer Network & Information Integration (Southeast University), Ministry of Education
{wen, bx.li,sundomore,jiakai_li}@seu.edu.cn

Abstract

Spectrum-based fault localization technique mainly utilizes testing coverage information to calculate the suspiciousness of each program element to find the faulty element. However, this technique does not fully take consideration of dependences between program elements , thus its capacity for efficient fault localization is limited. This paper combines program slicing with program spectrum technique, and proposes a program slicing spectrum-based software fault localization (PSS-SFL) technique. Firstly, PSS-SFL analyzes dependences between program elements, and deletes some elements unrelated to the failed test outputs; then it builds the program slicing spectrum model and defines a novel suspiciousness metric for each slice element; finally, the faulty element is located according to the suspiciousness metric results. Experimental results show that PSS-SFL can be effective and more precise to locate the fault than program spectrum-based Tarantula technique.

***Keywords*-Fault localization, program slicing spectrum, program slicing, program spectrum**

I. Introduction

A software fault is an incorrect step, process, or data definition in a computer program [1]. Software fault localization is to locate the fault that cause software failure. According to Collofello's research, in an attempt to reduce the number of delivered errors, it is estimated that most companies spend between 50% and 80% of their software development effort on testing [2], while software fault localization is one of the most complex and difficult tasks during the process of reducing the number of errors. There are two traditional ways to locate software faults: the first way is to insert print statements into the program, and then analyze the suspicious statements according to the print result; the second way is to set a breakpoint at some statement, and then single step to the next statement and determine whether the statement is faulty or not. Both ways above have to be performed manually. When the program is complex and large-scale, it is difficult to fully analyze the fault and the work will be very huge. So these two traditional ways above are not very efficient.

Program slicing technique was firstly proposed by Weiser [3]. It can abstract program's statements according to a specific criterion, so the fault in the program can be limited to a small region, that is, a relevant slice. Program slicing technique includes static program slicing technique and dynamic program slicing technique. Software fault localization based on static program slicing [4][5] analyzes the data flow and control flow of the program statically to reduce the search domain of faults. However, because the static program slicing is overly conservative, the precision of locating fault is very low. Based on static program slicing technique, dynamic technique [6][7] introduces more precise slicing criterion and the search domain of faults can be further reduced.This paper firstly abstracts the dynamic slicing criterion according to the failed test information, then constructs a fault-related slice to locate the fault.

In recent years, program spectrum-based software fault localization [8][9] was proposed, which can be applicable to the large-scale program and easy to implement. A program spectrum characterizes, or provides a signature of a program behavior [10]. Generally speaking, the collection of program spectrum is very simple, moreover, its storage is easy, so it can be suitable for situations with limited resources. Program spectrum-based software fault localization technique first statistically analyzes program spectrum and computes the probability that each

*Supported partially by the National Natural Science Foundation of China under Grant No. 60773105 and no. 60973149, and partially Supported by Doctoral Fund of Ministry of Education of China under Grant No. 20100092110022, and partially by National High Technology Research and Development Program under Grant No.2008AA01Z113.
Correspondence to: bx.li@seu.edu.cn

program element may include faults, commonly known as suspiciousness; then program elements are checked according to the descending order of the suspiciousness until the fault is found. Researches show that even for the lowest quality of programs, only 20% of the program code remains to be investigated to locate the fault with the program spectrum-based technique [9][11]. However, this technique does not fully take consideration of dependences between program elements, thus its capacity for efficient fault localization is limited. This paper combines program slicing with program spectrum technique, and proposes a program slicing spectrum-based software fault localization (*PSS-SFL*) technique. The main contributions of this paper are as follows:

(1) We delete program elements that have no dependence with the fault program elements by program slicing technique and construct a fault-related slice according to the test information,and then propose a program slicing spectrum-based software fault localization model.

(2) Based on the popular suspiciousness metric technique, we propose a new suspiciousness metric technique by introducing the frequency and contribution of each program element.

(3) We verify the effectiveness of *PSS-SFL* technique with our experiment and compare *PSS-SFL* technique with the popular Tarantula technique based on program spectrum.

II. Preliminaries

A. Dynamic program slicing

A dynamic program slice [6][7] consists only of statements that influence the value of a variable for a specific program input. The location, variable and input are referred as the dynamic slicing criterion as Definition 1, then a dynamic program slice is abstracted from the program dependence graph as Definition 2 [6][12].

Definition 1 (Dynamic slicing criterion): Given a program P, P's variable set V and input set I, a dynamic slicing criterion C of the program P is a triple $< s, v, i >$, where $s \in P$ is a specific statement in P, $v \in V$ denotes a subset of the program variables and $i \in I$ means a specific input to the program P.

Definition 2 (Program dependence graph): Given a program P,P's program dependence graph is a two-tuple $< N, E >$, where N is a set of nodes which denote the statements in P, E is a set of edges which consist of control dependence edges and data dependence edges. A control dependence edge $< v_i, v_j > \in E$ shows node v_i is executed depending on the output of the predicate expression at node v_j and a data dependence edge $< v_i, v_j > \in E$ indicates that the computation performed at vertex v_i directly depends on the value computed at vertex v_j.

Table I. *PSS-SFL* **model**

Program	Test cases						Suspiciousness
	Failed			Passed			
	T_1	...	T_s	T_{s+1}	...	T_m	
e_1	$b_{1,1}$	...	$b_{1,s}$	$b_{1,s+1}$	...	$b_{1,m}$	p_1
e_2	$b_{2,1}$	...	$b_{2,s}$	$b_{2,s+1}$	...	$b_{2,m}$	p_2
...	...	...	...	...	...	...	...
e_n	$b_{n,1}$	...	$b_{n,s}$	$b_{n,s+1}$	...	$b_{n,m}$	p_n

B. Program spectrum-based software fault localization

Recently, many researchers have proposed various fault localization techniques based on program spectrum[8][9][10][11][13][14][15][16]. Generally speaking, these techniques mainly include three steps: (1) collect coverage information of program elements based on test executions; (2) compute the element suspiciousness based on coverage information; (3) locate the fault according to the suspiciousness in a descending order. Program spectrum is usually defined as follows:

Definition 3 (Program spectrum): Given a program P, n is the number of elements in P and $T = T_F \cup T_P$ is a set of test executions where $T_F = \{T_1, ..., T_s\}$ is the set of failed test executions and $T_P = \{T_{s+1}, ..., T_m\}$ is the set of passed ones, then program spectrum is a two dimensional matrix $M_{n,m}$: $\forall b_{i,j} \in M_{n,m}$,

$$\mathrm{b}_{i,j}=\begin{cases} 1 & T_j \text{ covered } e_i \\ 0 & \text{otherwise} \end{cases}, (1 \leq \mathrm{i} \leq \mathrm{n}, 1 \leq \mathrm{j} \leq \mathrm{m})$$

A visual model of program spectrum-based software fault localization is shown in the following table. Rows of the table denote corresponding elements in the program and columns $T_1, \ldots, T_m$ indicate the number of test executions is m including s failed ones and $m-s$ passed ones. $b_{i,j}$ is the element of program spectrum matrix with value 1 or 0 which indicates whether T_j executed the element e_i or not. The last column is the suspiciousness of its corresponding element. The metric of suspiciousness is usually based on the following assumptions: the more failed tests and the less passed ones that executed the element e_i, the greater likelihood that e_i is faulty, that is, in the table, the larger value of $\sum_{i=1}^{s} b_{i,j}$ and the smaller value of $\sum_{i=s+1}^{m} b_{i,j}$, the greater value of suspiciousness(e_i). So $suspiciousness(e_i)$ is generally defined as follows [8]:

Definition 4 (Suspiciousness):

$$suspiciousness(e_i) = \frac{failed(e_i)\%}{failed(e_i)\% + passed(e_i)\%}$$

In the definition above, $failed(e_i)\%$ is the percentage of failed tests that executed e_i in total failed tests, and $passed(e_i)\%$ is the percentage of passed tests that executed e_i in total passed tests. In general, different researchers have different definitions to $failed(e_i)\%$ and $passed(e_i)\%$, but their targets are all to locate

the faulty element more precisely by the computation of $suspiciousness(e_i)$. This paper defines a new suspiciousness metric based on popular Tarantula's suspiciousness metric by defining a different $failed(e_i)\%$ and $passed(e_i)\%$.

III. program slicing spectrum-based software fault localization

A. Program slicing spectrum model

Traditional program spectrum-based fault localization technique considers all elements of a test execution, but the elements are not all related to the test output,so we introduces the program slicing technique to delete the unrelated elements to the fault and improve the precision of fault localization.

To more effectively locate faults, we firstly collect fault related slice according to the failed test executions:

Definition 5 (Fault-related slice): Given a set of test executions T=$T_F \cup T_P$, where T_F =$T_1, \ldots, T_s$ is the set of failed test executions and T_P=$T_{s+1}, , T_m$ is the set of passed ones, Let $Slice(T_i)$ $(i = 1, \ldots, m)$ be the dynamic slice corresponding to the test execution T_i, then fault-related slice is an elements set $Slice = Slice(T_1) \cup Slice(T_2) \cup \ldots \cup Slice(T_s)$.

Based on the above definition, program slicing spectrum is defined as follow:

Definition 6 (Program slicing spectrum): Given a set of test executions $T = T_F \cup T_P$, where $T_F = \{T_1, \ldots, T_s\}$ is a set of failed test execution and $T_P = \{T_{s+1}, \ldots, T_m\}$ is the set of passed ones, let the number of elements in $Slice$ be n, that is n=$|Slice|$, then program slicing spectrum is a two-dimensional $M_{n,m}$: $\forall f_{i,j} \in M_{n,m}$(1≤i≤n, 1 ≤j≤m), $f_{i,j}$=$\begin{cases} k & \text{the frequency that } T_j \text{ exceuted } e_i \text{ in } Slice \\ 0 & \text{otherwise} \end{cases}$

B. suspiciousness model

Program spectrum-based software fault location technique is performed based on the suspiciousness metric. Definition 4 gives a popular suspiciousness model. This paper improves the traditional computing algorithm of $failed(e_i)\%$ and $passed(e_i)\%$ by introducing the execution frequency and contribution of each program element. The popular definition of $failed(e_i)\%$ and $passed(e_i)\%$ is proposed in Tarantula technique as follows:

$$failed(e_i)\% = \frac{a_{11}(e_i)}{a_{11}(e_i)+a_{01}(e_i)}, passed(e_i)\% = \frac{a_{10}(e_i)}{a_{10}(e_i)+a_{00}(e_i)}$$

In the formula above,$a_{11}(e_i)$ represents the number of failed tests that executed the element e_i,$a_{01}(e_i)$ represents the number of failed tests that didn't execute the element e_i,$a_{10}(e_i)$ represents the number of passed tests that executed the element e_i, $a_{00}(e_i)$ represents the number of passed tests that didn't execute the element e_i. Obviously, Tarantula technique computes the suspiciousness of each program element by counting the number of failed tests and passed tests that executed the element. When different tests executed the same element, the contribution of the element to each test result is considered to be equal. That is, in Table I, Tarantula technique just considers if e_i was executed by each test and just counts the number of 1 of the line e_i. In reality, the number of program elements that each test executed is different and the frequency that each program element is executed is also different, so the contribution of a program element to different test result is unequal. For example, if a test just executed a program element, then the contribution of this program element to the test result is 100%; while this test executed two program elements and each element was executed one time, the contribution of each program element to the test result is 50%. The contribution can be computed from the column of program spectrum matrix. Below is our definition of $failed(e_i)\%$ and $passed(e_i)\%$:

$$failed(e_i)\% = \frac{\sum_{j=1}^{m} P_{T_j} \times C_{i,j}}{\sum_{j=1}^{m} P_{T_j}}, passed(e_i)\% = \frac{\sum_{j=1}^{m} (1-P_{T_j}) \times C_{i,j}}{\sum_{j=1}^{m} (1-P_{T_j})},$$

$$\text{where } C_{i,j} = \frac{f_{i,j}}{\sum_{k=1}^{n} f_{k,j}}, P_{T_j} = \begin{cases} 1 & T_j \text{ is failed} \\ 0 & T_j \text{ is passed} \end{cases}$$

In the formula above, $C_{i,j}$ represents the contribution of the program element e_i to the test execution T_j where $f_{i,j}$ is the frequency of e_i that is executed by T_j and details are shown in Definition 6. P_{T_j} with value 1 represents that T_j is a failed test execution, otherwise P_{T_j} is assigned a value 0. Denominators of $failed(e_i)\%$ and $passed(e_i)\%$ represent the number of failed test executions and the number of passed ones; their numerators represent the sum of the contributions of program elements to each failed test execution and passed one. These are different from the suspiciousness metric in Tarantula technique.

When the suspiciousness of each program element is obtained, the fault can be located according to the suspiciousness in a descending way.

C. Program slicing spectrum-based software fault localization model

According to the definition of program slicing spectrum and suspiciousness model, a visual model of program slicing spectrum-based software fault localization is shown in Table II. Rows of the table denote corresponding elements in the fault-related slice $Slice$ and columns $T_1, \ldots, T_m$indicate m test executions where the number of failed ones is s and passed ones is $m - s$. $f_{i,j}$ (1 ≤i≤n, 1 ≤j≤m) is the element of program slicing spectrum matrix with value k or 0 which means the

Table II. *PSS-SFL* **model**

Slice	Test cases						Suspiciousness
	Failed			Passed			
	T_1	...	T_s	T_{s+1}	...	T_m	
e_1	$f_{1,1}$	...	$f_{1,s}$	$f_{1,s+1}$	...	$f_{1,m}$	p_1
e_2	$f_{2,1}$	...	$f_{2,s}$	$f_{2,s+1}$	...	$f_{2,m}$	p_2
...	...	...	...	...	...	...	...
e_n	$f_{n,1}$	...	$f_{n,s}$	$f_{n,s+1}$	...	$f_{n,m}$	p_n

frequency that T_j executed the element e_i . The last column is the suspiciousness of each corresponding element by the computation in the above section. Obviously, *PSS-SFL* technique greatly improves traditional program spectrum-based technique. It reduces the scale of the traditional spectrum, introduces the frequencies of program elements and defines contributions for suspiciousness metric.

Program slicing technique firstly abstracts corresponding slicing criterion, and then generates the slice by traversing program dependence graph. *PSS-SFL* technique generates the fault-related slice according to the failed test executions information, and then constructs program spectrum according to history test executions and finally computing the suspiciousness of each element in the fault-related slice.

IV. Experiment

In this section, we will verify the effectiveness of *PSS-SFL* and compare its precision of locating faults with popular Tarantula techniques.

A. Object of analysis

Many current researches on software fault localization use siemens suite as their object. Siemens suite consists of seven C programs and each program has several hundred lines of code. Because each program in siemens suite is mainly composed of branch structures, program spectrum-based software fault localization techniques generally have a high precision on these programs and *PSS-SFL* has not obvious advantages. This section uses a practical JAVA tool program *JAVA Hierarchical Slicing and Applications* (*JHSA*) as our object of analysis.*JHSA* program includes three packages, twenty-six classes and 11201 lines of code in all, which is used to construct a hierarchical system dependence graph for JAVA program, and to compute its hierarchical slice. We totally select 178 faulty versions for this experiment.

B. Experiment design and analysis

In this study, experiment data collection and analysis are conducted in four steps: first, collect program elements

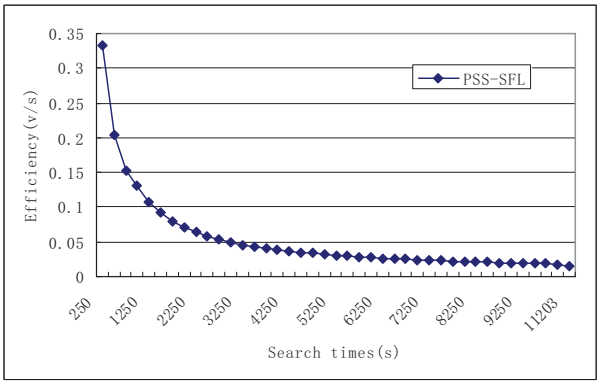


Figure 1. The effectiveness of *PSS-SFL*

coverage information by the tool eclemma and count the frequencies of the program elements; second, abstract the fault-related slice by a slicing tool *JHSA* according to the failed tests; third, construct the program slicing spectrum matrix based on the steps above; finally, compute the suspiciousness as depicted in the above section and locate faults according to the suspiciousness of each program element in a descending order.

To evaluate the effectiveness of *PSS-SFL*, an efficiency score is used in this experiment. Given v is the number of versions in which faults have been successfully located and s is the number of program elements that have been searched for locating faults in each version.we evaluate the effectiveness with following formula:

$$Efficiency = \frac{v}{s}$$

In the formula above, efficiency means that the number of versions in which faults have been successfully located when searching s elements in each version. Here, s can be also referred as search times. In the ideal case, if the value of s is 1 and the value of v is the number of total versions, efficiency denotes that this technique can once find faults in all versions and its efficiency is the highest.

C. Result and analysis

In this section, we verify the effectiveness of *PSS-SFL* and compare its precision of locating faults with popular Tarantula techniques with the above efficiency score.

Figure 1 shows the relation between the efficiency of *PSS-SFL* and search times. In the figure, with the increase of search times, the efficiency reduced. This implies *PSS-SFL* technique can quickly locate most faults at the beginning, and only a small number of faults need to be searched many times further, so the technique is very effective.

Figure 2 shows the relation between search times and the number of versions and compares *PSS-SFL* with Tarantula technique. In the figure, when the value of search times

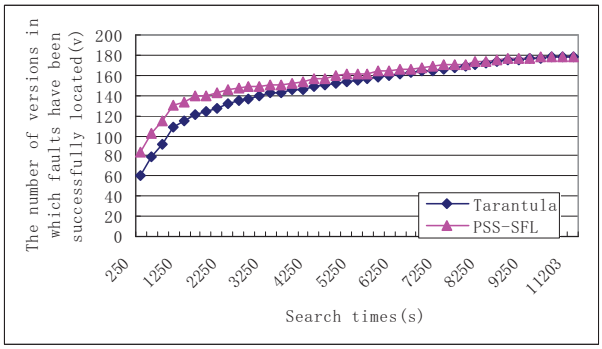


Figure 2. The comparation between *PSS-SFL* and Tarantula technique

is about 250, the number of versions in which faults have been found with *PSS-SFL* is nearly 80, while the value with Tarantula is about 60. In this experiment, the number of all versions is 178, so when searching 250 times, *PSS-SFL* can find faults in nearly half versions and Tarantula about one third. Moreover, Figure 2 shows *PSS-SFL* can be more effectively and precisely locate faults most of the time than Tarantula technique.

D. Threads to validity

According to the experiment above, *PSS-SFL* technique can effectively and precisely locate faults. However, there are some threads to the experiment.

First, the scale of the object of analysis is still relatively small. It is a real application program, so, to some extent, it can verify the effectiveness and precision of *PSS-SFL*. However, it still needs to be verified for the larger scale program. In addition, with the increase of sequential structures in programs, the precision of Tarantula technique will reduce. Although *PSS-SFL* introduce program slicing technique to reduce this degradation to some extent, but this degradation also exists in the *PSS-SFL* technique.

Second, if program fault is caused by the omission of some element, we can locate the fault to its prior element close to the position of omission element. Then, we can find the corresponding fault from this prior element, but this needs some manual analysis.

Third, because suspiciousness metric is based on the statistical number of failed test executions and passed test executions, the precision of suspiciousness will be reduced with the increase of the number of faults in a version. This problem exists in traditional program spectrum-based software fault location technique and also exists in *PSS-SFL* technique.

V. Related work

Because of the importance and difficulty of software fault localization, more and more software engineering researchers pay attention to this field. They proposed various techniques for locating faults such as techniques based on program slicing [3][4][5][17][18][19][20][21] and program spectrum [8][9][10][11][13][14][15][16]. This paper combines the program slicing and program spectrum, and proposes *PSS-SFL* technique, which can more effectively locate faults.

Most of program slicing based-software fault localization techniques locate faults using set operation between slices. Dicing technique [17] computes the difference between a failed slice and a passed slice to confine the fault to a small region; execution slice [18][19] is some program execution blocks according to some specific inputs, it can be abstracted more effectively than traditional slice, and this slicing technique generally computes the intersection and union between slices to locate faults. The idea of the union is first to compute the union of all passed slices, then to get the difference between a failed slice and the union; the way of the intersection is first to compute the intersection of all passed slices, then to obtain the difference between the intersection and a failed slice. Renieris proposed another difference between a failed execution slice and a passed slice that has a closest distance to the failed slice [20]. In addition to set operations, DeMillo proposed a critical slicing for locating faults based on ”statement deletion” mutant operation [21]. *PSS-SFL* technique is different from these techniques: Although this technique abstracts dynamic program slice for fault-related slice, it constructs a program slicing spectrum model and finally computes the suspiciousness to locate faults. *PSS-SFL* technique is more suitable for general cases.

Program spectrum includes program frequency spectrum and program hit spectrum. Their difference is that a frequency spectrum introduces the execution frequency of each program element and a hit spectrum only considers if program elements are covered or not. Traditional program spectrum-based software fault localization is mainly based on program hit spectrum. There are usually two methods to improve the effectiveness of program hit spectrum-based software fault localization. One is to improve the suspiciousness model. The common suspiciousness models mainly have Tarantula [8],Jaccard [9],Ochiai [9],etc. In addition, Wong [13] introduced the weight of test cases into the suspiciousness metric to improve the effectiveness. The other is to optimize the test cases by deleting unrelated or redundant test cases [22][23][24]. *PSS-SFL* technique is different from these techniques: *PSS-SFL* introduces program slicing technique to delete elements that have no dependence with faults; moreover, *PSS-SFL* is based on program frequency spectrum and has a different sus-

piciousness model by introducing the contribution of each program element.

VI. Conclusion and future work

This paper proposed an effective fault location technique, *PSS-SFL* technique. First, this technique abstracts fault-related slice and deletes elements that have no dependences with faulty elements to improve the precision of locating faults; second, it introduces the execution frequency and the contribution of each program element to improve the traditional suspiciousness metric; finally, an experiment is conducted to verify the effectiveness of this technique.

Although our experiment has verified the effectiveness of *PSS-SFL* on a practical program, the efficiency will be reduced with the increase of faults in a single program as other software fault localization technique, so our future work will focus on improving the efficiency of locating faults in multi-faults program.

References

[1] IEEE. IEEE standard glossary of software engineering terminology, 1990.

[2] J.S.Collofello and S.N.Woodfield. Evaluating the effectiveness of reliability-assurance techniques. *Journal of Systems and Software*, 9(3):191–195, 1989.

[3] M.Weiser. Program slicing. *IEEE Transactions on Software Engineering*, 10(4):352–357, 1984.

[4] M.Weiser. Programmers use slices when debugging. *Communications of the ACM*, 25(7):446–452, 1982.

[5] J.R.Lyle and M.Weiser. Automatic program bug location by program slicing. In *Proceedings of International Conference on Computers and Applications*, pages 877–883, 1987.

[6] H.Agrawal and J.R.Horgan. Dynamic program slicing. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 246–256, 1990.

[7] H.Agrawal, R.A.DeMillo, and E.H.Spafford. Debugging with dynamic slicing and backtracking. *Software - Practice Experience*, 23(6):589–616, 1993.

[8] J.A.Jones, M.J.Harrold, and J.Stasko. Visualization of test information to assist fault localization. In *Proceedings of the 24th International Conference on Software Engineering*, pages 467–477, 2002.

[9] R.Abreu, P.Zoeteweij, and A.J.C.van Gemund. On the accuracy of spectrum-based fault localization. In *Proceedings of Testing:Academic and Industrial Conference-Practice and Research Techniques*, pages 89–98, 2007.

[10] M.J.Harrold, G.Rothermel, and R.Wu. An empirical investigation of program spectra. In *Proceedings of the 1998 ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering*, pages 83–90, 1998.

[11] J.A.Jones and M.J.Harrold. Empirical evaluation of the tarantula automatic fault localization technique. In *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering*, pages 273–282, 2005.

[12] J.Ferrante, J.Ottenstein, and J.D.Warren. The program dependence graph and its use in optimization. *ACM Transactions on Programming Languages and Systems*, 9(3):319–349, 1987.

[13] E.Wong, Y.Qi, L.Zhao, and et al. Effective fault localization using code coverage. In *Proceedings of International Computer Software and Applications Conference*, pages 449–456, 2007.

[14] R.Abreu, P.Zoeteweij, and A.J.C.van Gemund. An evaluation of similarity coefficients for software fault localization. In *Proceeding of the 12th IEEE Pacific Rim International Symposium on Dependable Computing*, pages 39–46, 2006.

[15] R.Abreu, A.Gonzlez, P.Zoeteweij, and et al. Automatic software fault localization using generic program invariants. In *Proceedings of the ACM symposium on Applied computing*, pages 712–717, 2008.

[16] R.Abreu and A.Gonzlez. Exploiting count spectra for bayesian fault localization. In *Proceedings of the 6th International Conference on Predictive Models in Software Engineering*, 2010.

[17] T.Y.Chen and Y.Y.Cheung. Dynamic program dicing. In *Proceedings of the Conference on Software Maintenance*, pages 378–385, 1993.

[18] H.Agrawal, J.R.Horgan, S.London, and et al. Fault localization using execution slices and dataflow tests. In *Proceedings of the 6th IEEE International Symposium on Software Reliability Engineering*, pages 143–151, 1995.

[19] W.Wong and Y.Qi. Effective program debugging based on execution slices and inter-block data dependency. *Journal of Systems and Software*, 79(7):891–903, 2006.

[20] M.Renieris and S.P.Reiss. Fault localization with nearest neighbor queries. In *Proceedings of the 18th IEEE International Conference on Automated Software Engineering*, pages 30–39, 2003.

[21] R.A.DeMillo and E.H.Spafford H.Pan. Critical slicing for software fault localization. In *Proceedings of the 1996 ACM SIGSOFT international symposium on Software testing and analysis*, pages 121–134, 1996.

[22] Y.B.Yu, J.A.Jones, and M.J.Harrold. An empirical study of the effects of test suite reduction on fault localization. In *Proceedings of the International Conference on Software Engineering*, pages 201–210, 2008.

[23] D.Hao, L.Zhang, Y.Pan, and et al. On similarity awareness in testing based fault localization. *Automated Software Engineering*, 15(2):207–249, 2008.

[24] D.Hao, T.Xie, L.Zhang, and et al. Test input reduction for result inspection to facilitate effective fault localization. *Automated Software Engineering*, 17(1):5–31, 2010.

Interface Testing Using a Subgraph Splitting Algorithm: A Case Study

Sergiy Vilkomir, Ali Asghary Karahroudy, Nasseh Tabrizi
Department of Computer Science
East Carolina University
Greenville, USA
{vilkomirs, asgharykarahroudy10, tabrizim}@ecu.edu

Abstract— **One of several obstacles in automated test generation is the presence of dependencies among test data where not all combinations of test parameter values are feasible. In previous attempts, a subgraph splitting algorithm has been suggested where a graph model is created for test generation in order to reflect dependencies among test data. This paper extends these results and considers a new improvement of the algorithm that can significantly simplify the final model. We investigate this approach in a case study of an interface test case generation for the VirtualECU software system. Detailed graph models are developed to reflect all dependencies and generate all feasible test cases during VirtualECU interface testing.**

Keywords-testing; dependencies; subgraph splitting

I. INTRODUCTION

Testing of modern software systems is a complex process that requires a high level of automation, which is why the development of methods and tools for automated testing has become an important research area. One major obstacle to automated test generation is the presence of dependencies in test data, where not all combinations of test parameter values are feasible.

Different approaches to test generation with dependencies exist in the literature [1-4]. In previous works [6, 7], the first author of this paper has suggested a subgraph splitting algorithm, which reflects dependencies among test data in terms of a graph model. A tool for automatic model construction [8] was also developed. The graph model is used as a basis for test case generation then. Generation of test cases from the graph can be done manually or automatically using existing out-of-the-shelf tools, such as JUMBL [5]. This paper extends the previous results taking into consideration the following new materials:

- Improvement of the subgraph splitting algorithm by adding a new step (merging nodes), which can significantly simplify the final model.
- Application of the algorithm during interface testing.
- Reflection of more complex "one-on-two" dependencies.
- Practical application of the algorithm and interface test case generation for a real case study: VirtualECU [9] software system.

The rest of this paper is organized as follows: In Section 2.A, the specifics of dependencies during interface testing is analyzed. Section 2.B reviews the subgraph splitting algorithm and suggests a new step for merging nodes. VirtualECU, the software system that is used as a case study for the application of our approach, is described in Section 3. Section 4 contains detailed diagrams of graph models, created by the subgraph splitting algorithm for dependencies between interface options of the VirtualECU. Generation of test cases from these models is also considered in this section. Conclusions are presented in Section 5.

II. PROPOSED APPROACH

A. Dependencies During Interface Testing

In [6, 7], a graph input space model for test case generation and a subgraph splitting algorithm for creation of this model are suggested. Different types of input variables (discrete and continuous numerical data, symbols, strings, etc.) are considered and used as test parameters. The model reflects dependencies between values of input variables and is used for generation of feasible test cases.

For many types of software, a direct use of input variables is unnatural. Rather, a user chooses some interface options from menu bars, tool bars, buttons, etc., to actuate software functionalities. However, from the point of view of testing, these situations are very close. Similar to input variables, interface options can also be considered as values of test parameters. The choice of one interface option can affect possibilities for another. In other words, it is possible to have dependencies between interface options. Based on this, we can conclude that the subgraph splitting algorithm can be applied for interface testing.

Input information for the algorithm includes a list of test parameters and their values, and dependencies between these values. For interface testing, the test parameters are not directly defined and will be determined at the first stage of modeling. Some values of the test parameters may not be defined explicitly by the user. For example, some software functionalities can be different across user categories, but the category itself is not chosen explicitly. Users do not need to have any knowledge about the used gradation, and the users' category can be determined automatically based on user IDs during the log-in process.

Each dependency for the subgraph splitting algorithm should be described using "if-then" implications for test parameters values. Because of the presence of dependencies, some user actions that correspond with test parameters can be restricted or not even be applicable. A general description of the subgraph splitting algorithm is given in Section 2.A and a detailed example of the use of this algorithm for interface testing is discussed in Section 4.

B. Subgraph Splitting Algorithm

The initial start point of the algorithm is a linear direct labeled graph, which describes values of test parameters but does not include any dependencies between them. Any node in the graph corresponds with one test parameter. An ingoing edge of this node is labeled with values of this parameter. The subgraph splitting algorithm then uses a four-step approach to reflect each dependency at a time. The main step is subgraph splitting after which one test parameter corresponds with several nodes. The labels of edges are then changed according to dependency restrictions.

It is necessary to mention that the constructed graph does not coincide with other type of graphs widely used in software engineering and testing. It is not a control-flow or data-flow graph, neither is it a dependency graph used in constraint programming. Connected nodes do not represent any relationships between test parameters; rather they just reflect some arbitrary predefined order of the parameters. The graph represents dependencies between test parameters in such a way that

- Each path through the graph correlates with a set of independent feasible test cases.
- Fixing one value from the label of each edge of some path gives a feasible test case, i.e., a test case that satisfies all dependencies.
- The set of test cases, generated from all paths of the graph is a complete set of all feasible and only feasible test cases.

The algorithm contains the following four steps for each dependency. A more detailed description of the first three steps can be found in [7]. The fourth step is the new improvement of the algorithm, which allows simplifying a graph model in some situations. The steps of the algorithm are explained below for the dependency between test parameters $a \in A$ and $d \in D$: If $a \in A1$ then $d \in D1$, if $a \in A2$ then $d \in D2$, where A and D are sets of interface options, $A = A1 \cup A2$, $D = D1 \cup D2$, and $A1 \cap A2 = \emptyset$.

- *Step 1: Splitting a subgraph.* The subgraph includes all nodes for test parameters between a and d, including a but not including d. The subgraph is splitting (duplicating) and the new subgraph is connected with other nodes in the same way as the original subgraph. Adding the split subgraph creates new paths in the graph mode, which reflects the dependency between parameters.

- *Step 2: Labeling ingoing and outgoing edges of split subgraphs.* If some ingoing edge is labeled with a set I, then the label for this edge for the original subgraph changes to $I \cap A1$ and the label for the corresponding edge for the duplicated subgraph changes to $I \cap A2$. In much the same way, if some outgoing edge is labeled with a set O, then the label for this edge for the original subgraph changes to $O \cap D1$ and the label for the corresponding edge for the duplicated subgraph changes to $O \cap D2$.

- *Step 3: Eliminating dead nodes and edges.* After step 2, some edges can be labeled with the empty set. Such edges are considered as dead and should be eliminated from the graph model. After eliminating edges, some nodes can be without ingoing or outgoing edges. Such nodes are considered dead and also should be eliminated together with corresponding edges. This cycle should be repeated until all dead edges and nodes are eliminated.

- *Step 4: Merging nodes.* We suggest here this new step because it is possible that after Step 2, outgoing edges from two nodes may go to the same node and have the same labels. In such situations, these two nodes should be merged. This action simplifies the graph model and reduces the number of paths through the graph, eliminating paths that generate the same sets of test cases.

III. A Case Study: VirtualECU Software

VirtualECU is a software system for educational support processes developed in the Computer Science (CS) Department at East Carolina University (ECU), North Carolina. The system is composed of two different parts. The first part can be accessed by all visitors and serves as an informational site for the graduate programs offered by the CS department. The other part is designed to establish connections between the students and faculty of the department. It facilitates delivering course materials, lecture sharing, quiz and test taking, grade recording, etc. Accessing the latter part is restricted by user accounts and their special privileges. This second part of VirtualECU is our focus point in the case study.

There are three different user groups in VirtualECU: administrators, instructors, and students. Each group is assigned specific access levels and permissions. Users need to log into the system before being able to access the services. VirtualECU functions with a bidirectional interactive mechanism. Instructors design, create, and upload course materials such as lectures, quizzes, and tests. They may also put a time limit on those materials. Students have access to these materials; they take quizzes, submit assignments, etc. Instructors review and grade student submissions and update records, which can also be checked by students. There are two presentation modes in VirtualECU: List view (Figure 1) and Calendar view (Figure 2). In the List view, a list of existing materials is shown to the user as hyperlinks to allow access to details about course components. The Calendar view helps users access course materials using date tags (upload date, due date, etc.).

The system interface allows users doing typical actions, such as "open," "download," "upload," "delete," etc. Some scenarios may include a sequence of actions. Availability of actions depends on the group of users and course objects. For example, a student can download a lecture, but not a quiz, an administrator or instructor can change the date and time of some course objects, but a student cannot perform this action, etc. A more detailed discussion of the actions and their inter-dependencies are considered in the next section.

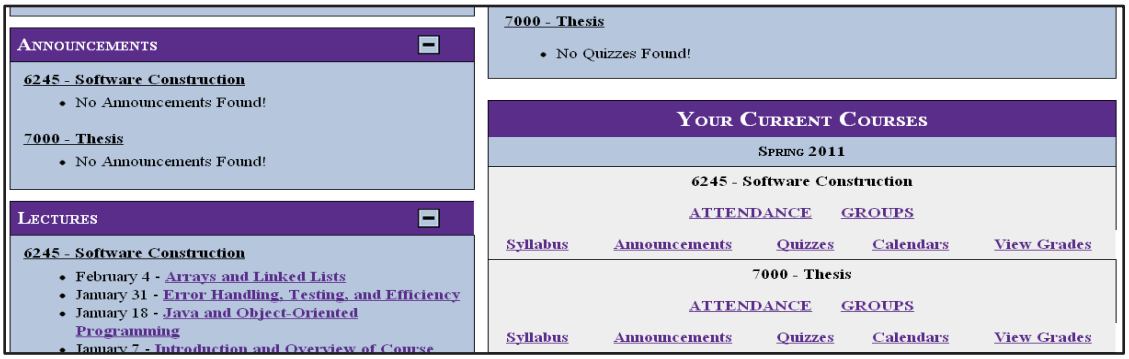


Figure 1. Fragment of VirtualECU List view

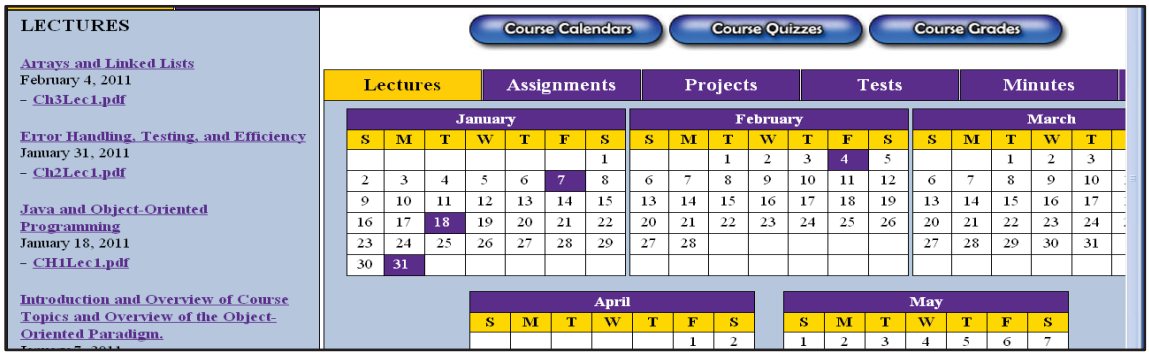


Figure 2. Fragment of VirtualECU Calendar view

TABLE I. TEST PARAMETERS AND THEIR VALUES

N	Test parameter	Values
1	User (*user)*	U={Student (std), Instructor (instr), Administrator (adm)}
2	Course object (*object)*	O={Quiz (q), Lecture (lec), Project (pr), Assignment (as), Announcement (an), Test (test), Note (note), Grade (grade), Attendance (at), Course Listing (col), Groups (gr)}
3	Time (*time*)	T={Passed (p), Ready(r), Waiting (w), not exist (ne), time independent (ind)}
4	Presentation (*pres*)	P={List (list), Calendar (cal)}
5	Action 1 (*act1*)	A1={Download (dw), Upload (up), Delete (del), Rename (ren) , Edit (ed), Open (op), N/A (na)}
6	Action 2 (*act2*)	A2={Select answer and save (ss), Select answer and continue (sc), Jump (jum), N/A (na)}
7	Action 3 (*act3*)	A3={Close (cl), Submit (sub)}

IV. VIRTUALECU TEST SPACE MODELING

A. Test parameters and dependencies

As a first stage of modeling, we grouped all interface options into eight test parameters. The parameters and their values are presented in Table 1. Provided in brackets are short identifiers that are used later in graph diagrams (Section 4.B).

There are two types of dependencies between VirtualECU test parameters: "one-on-one" and "one-on-two" dependencies. For the "one-on-one" dependency, values of one specific test parameter depend only on values of another one parameter. For example, parameter *time* depends on parameter *object*. For the "one-on-two" dependency, values of one specific test parameter depend on values of two other parameters at the same time, and this dependency cannot be presented as a composition of two "one-on-one" dependencies. For example, parameter *action1* depends on parameters *user* and *object*, but dependencies *user-action1* and *object-action1* do not exist separately.

There are two ways to deal with "one-on-two" dependencies. As a first approach, instead of two separate dependees, we can create one derived parameter-vector. For example, we can consider one parameter *user&object* with Cartesian product $U \times O$ as a set of values and then directly apply the subgraph splitting algorithm. The second approach is possible when a number of possible values of one of dependee is small. In this case, instead of one model with "one-on-two" dependency, it is possible to create several models (separately for each value of this parameter) with "one-on-one" dependency. In our case study, parameter *user* has only three values, so the second approach is applicable. Because of the limited space of the paper, we consider here graph models only for *user*=instructor. However, model for other users are similar; therefore, the models provided here fully illustrate the method of model creation by the subgraph splitting algorithm.

There are six dependencies among VirtualECU test parameters (Table 2). Dependencies 1, 2, 4, and 5 are for all users and dependencies 3 and 6 are formulated only for instructors.

TABLE II. DEPENDENCIES AMONG TEST PARAMETERS

N	Dependency	Values of dependee	Values of dependant
1	*object - time*	O_1={q, pr, as}	{p, r, w, ne}
		O_2={ lec, an, note, grade, at, gr, test}	{r, ne}
		{col}	{ne, ind}
2	*pres - act1*	{list}	A1\{up}
		{cal}	A1
3	*time - act1*	{p, w, r}	A1\{up}
		{ne}	{up}
		{ind}	{del, ed}
4	*user – act1*	{instr, adm }	A1\{na}
		{std }	A1
5	*user – act2*	{instr, adm }	{na}
		{std }	A2
6	*act1 – act3*	{dw, del, ren, up, na}	{cl}
		{op, ed}	A3

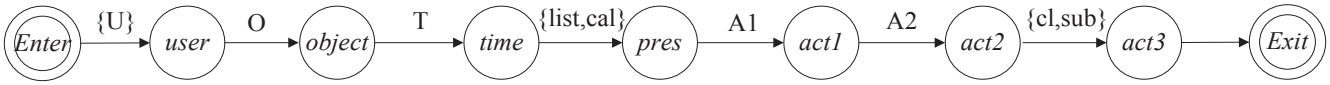


Figure3. Model for test parameters and their values (without dependencies)

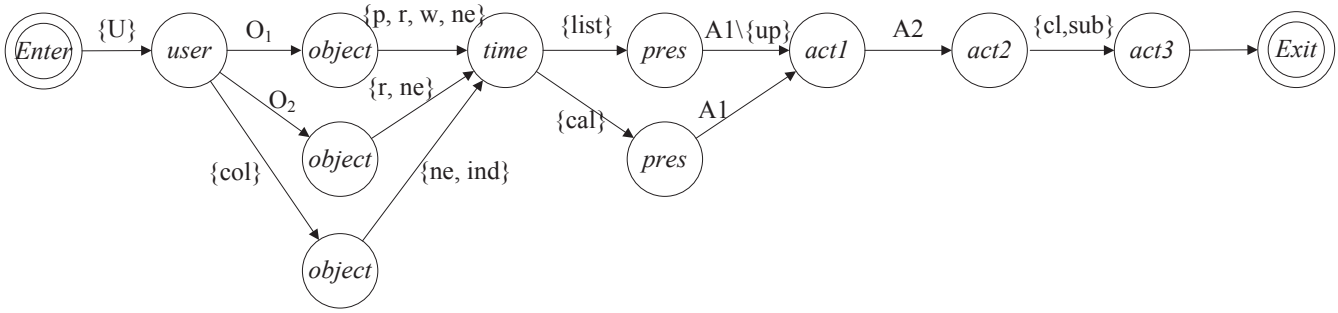


Figure 4. Model for dependencies 1 (between *object* and *time*) and 2 (between *pres* and *act1*)

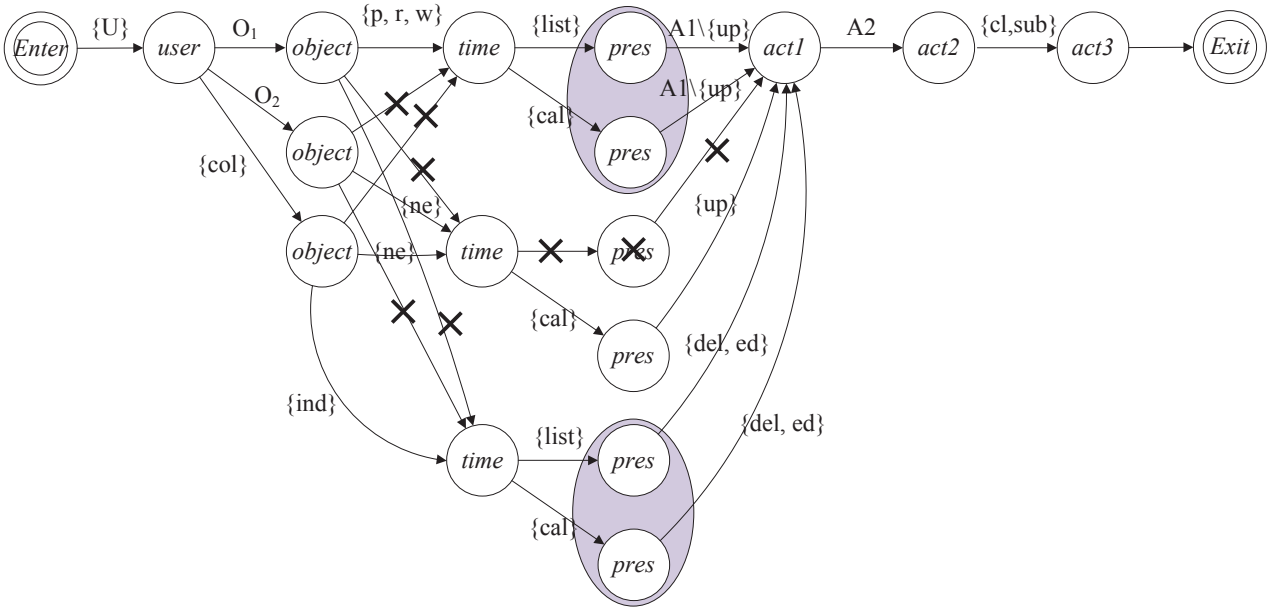


Figure 5. Model for dependency 3 (eliminating dead nodes and edges and merging nodes)

B. Graph models for dependencies

A graph model for dependencies is based on a simple linear graph where nodes represent test parameters, and ingoing arcs are labeled with all possible values of these parameters (Fig.3).

To reflect dependency 1 between and *time* parameters, *object* node should be split. For dependency 2 between *pres* and *act1* parameters, *pres* node should be split. Dependency 1 is described by three "if-then" implications (Table 2); therefore, there are three nodes for *object*. Dependency 2 is described by two "if-then" implications, and so *pres* node should be duplicated (Figure 4). There are no dead arcs, dead nodes, and nodes that should be merged.

For dependency 3 between *time* and *act1* parameters, the subgraph with nodes *time* and *pres* should be split into three identical subgraphs. After labeling ingoing arcs, some of the arcs are dead and should be eliminated (marked with crosses in Fig. 5). Also, there are two pairs of *pre* nodes, which outgoing arcs have the same labels (oval forms in Fig. 5). According to step 4 of the subgraph splitting algorithm, these nodes should be merged. The model for dependencies 1, 2, and 3, after eliminating dead nodes and edges and merging nodes, is given in Fig. 6.

Dependency 4 restricts the value of *act3* for instructors to be "na." Dependency 5 shows that the value of *act1* for instructors cannot be "na." Both these dependencies require only relabeling of the ingoing arc of *act1* and *act3* without splitting subgraphs. For dependency 6 between *act1* and *act3* parameters, one cycle of step 3 of the subgraph splitting algorithm is necessary for eliminating dead edges (we skip this diagram because of the limited space of the paper). The final model for dependencies 1 - 6 is presented in Fig. 7.

C. Test case generation

The final model in Fig. 7 contains six different paths through the graph. Each path represents a group of test cases, where the values of test parameters are labels of arcs of this path (Table 3).

The values of different parameters are independent and can be combined in any combinations. Together, these test cases form a set of all feasible and only feasible tests that satisfy all dependencies. For instructor mode, the total number of feasible test cases is 140.

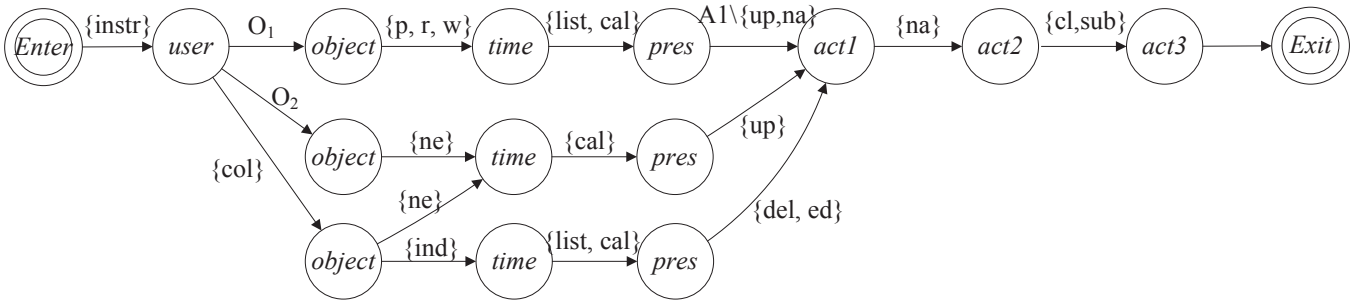


Figure 6. Model for dependency 3 between *time* and *act1* parameters

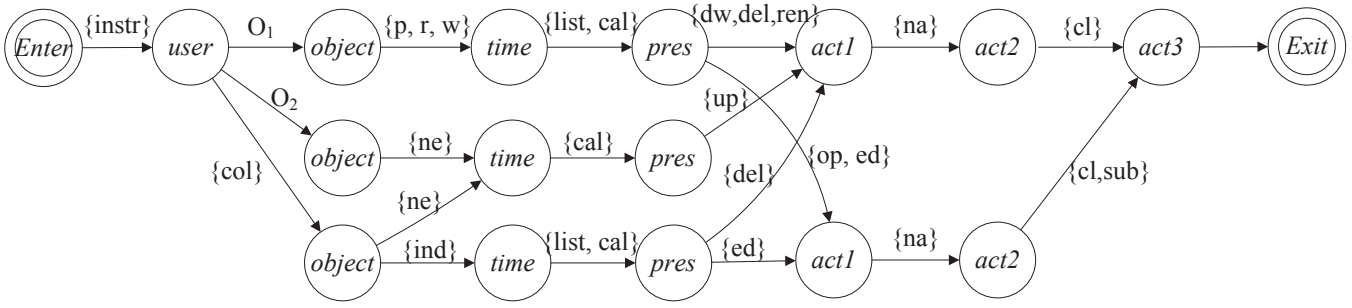


Figure 7. Final model for dependencies 1 - 6

TABLE III. FEASIBLE TEST CASES

Path	user	object	time	pres	act1	Act2	Act3	Number of tests
1	instr	O1	{p, r, w}	{list, cal}	{dw,del, ren}	{na}	{cl}	54
2	instr	O1	{p, r, w}	{list, cal}	{op, ed}	{na}	{cl, sub}	72
3	instr	O2	{ne}	{cal}	{up}	{na}	{cl}	7
4	instr	{col}	{ne}	{cal}	{up}	{na}	{cl}	1
5	instr	{col}	{ind}	{list, cal}	{del}	{na}	{cl}	2
6	instr	{col}	{ind}	{list, cal}	{ed}	{na}	{cl, sub}	4
Total								140

The benefit of using the created model is that this model allows generation of the test cases according to different strategies. Applying existing tools (i.e. JUMBL) to our model, it is possible to randomly generate any fixed number of feasible tests, or provide path/node/arc coverage, or generate test cases with highest probabilities according to a user profile, etc. The model can also be successfully used for statistical testing.

V. CONCLUSIONS

A subgraph splitting algorithm is a method for software input space modeling and test case generation. In this paper, a new area of this algorithm application, specifically interface testing, is considered. The main challenge for such application is representing all interface options as a set of test parameters and their values. The subgraph splitting algorithm then allows us to create a graph model, which reflects all dependencies between interface options and generates only feasible test cases.

The improved algorithm makes the model more compact and convenient for practical applications, as illustrated for a case study of the VirtualECU software. The model developed here presents all 140 feasible test cases for VirtualECU and allows further test selection according to various test criteria.

ACKNOWLEDGMENT

This research is supported by the "Google Faculty Research Award" from Google, Inc.

REFERENCES

[1] A. Beer, S. Mohacsi, “Efficient Test Data Generation for Variables with Complex Dependencies,” Proceedings of the 1st International Conference on Software Testing, Verification, and Validation, 9-11 April 2008, pp. 3–11.

[2] A. Calvagna and A. Gargantini, “A Logic-Based Approach to Combinatorial Testing with Constraints,” Tests and Proofs, Springer Berlin / Heidelberg, LNCS 4966, 2008, pp. 66–83.

[3] M. B. Cohen, M. B. Dwyer, and J. Shi, “Interaction testing of highly-configurable systems in the presence of constraints,” Proceedings of the 2007 international symposium on Software testing and analysis (ISSTA '07), 2007, ACM, pp. 129–139.

[4] M. Grindal, J. Offutt, J. Mellin, “Managing Conflicts When Using Combination Strategies to Test Software,” Proceedings of the 18th Australian Software Engineering Conference (ASWEC’07), Melbourne, Australia, April 2007, pp. 255–264.

[5] S. Prowell, “JUMBL: A Tool for Model-Based Statistical Testing,” Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS’03), Big Island, HI, USA, January 2003.

[6] S. Vilkomir, “Statistical testing for NPP I&C system reliability evaluation,” Proceedings of the 6th American Nuclear Society International Topical Meeting on Nuclear Plant Instrumentation, Controls, and Human Machine Interface Technology (ICHMI 2009), Knoxville, TN, USA, April 5–9, 2009.

[7] S. Vilkomir, T. Swain, and J. Poore, “Software Input Space Modeling with Constraints among Parameters,” Proceedings of the 33rd Annual IEEE International Computer Software and Applications Conference (COMPSAC 2009), Seattle, Washington, July 20–24, 2009, pp. 136–141.

[8] S. Vilkomir, K. Abdelfattah, S. Gummadi, “MIST: Modeling Input Space for Testing Tool,”. Proceedings of the 13th IASTED International Conference on Software Engineering and Applications (SEA 2009), Cambridge, MA, USA, November 2–4, 2009, pp. 210–217.

[9] Virtual ECU, http://virtual.ecu.edu

Machine Learning-based Software Testing: Towards a Classification Framework

Mahdi Noorian[1], Ebrahim Bagheri[1,2], and Wheichang Du[1]
University of New Brunswick, Fredericton, Canada[1]
Athabasca University, Edmonton, Canada[2]
m.noorian@unb.ca, ebagheri@athabascau.ca, wdu@unb.ca

***Abstract*—Software Testing (ST) processes attempt to verify and validate the capability of a software system to meet its required attributes and functionality. As software systems become more complex, the need for automated software testing methods emerges. Machine Learning (ML) techniques have shown to be quite useful for this automation process. Various works have been presented in the junction of ML and ST areas. The lack of general guidelines for applying appropriate learning methods for software testing purposes is our major motivation in this current paper. In this paper, we introduce a classification framework which can help to systematically review research work in the ML and ST domains. The proposed framework dimensions are defined using major characteristics of existing software testing and machine learning methods. Our framework can be used to effectively construct a concrete set of guidelines for choosing the most appropriate learning method and applying it to a distinct stage of the software testing life-cycle for automation purposes.**

I. Introduction

Software Testing (ST) is an investigation process which attempts to validate and verify the alignment of a software system's attributes and functionality with its intended goals. Software testing is a labour intensive and costly process and as mentioned in [2], [3], a testing process may require up to 50% of the development resources. Due to this fact, automated testing approaches are desired to reduce this cost and time. Besides, automation can significantly enhance the testing process performance. Hence, for future software systems development, steps need to be taken towards the development of automated testing strategies [4].

Several interesting attempts have already been made for automating the software testing process. Machine Learning (ML) as a sub domain of AI [12] is widely used in various stages of the software development life-cycle [19], especially for automating software testing processes [5]. In [1], [17], evolutionary algorithms have been employed for automating test case generation. In [16], Artificial Neural Networks (ANN) have been used to build a model for estimating the effectiveness of the generated test cases. Briand *et al.* in [8] has proposed a method based on the C4.5 decision tree algorithm for predicting potential bugs in a software system and localizing the bugs in order to reduce the debugging process time. All research works show that the employment of machine learning techniques is a promising approach for automating testing processes. But, still some major questions remain that need to be addressed and further investigated such as:

- What types of machine learning methods can be effective in different aspects of software testing?
- What are the strengths and weaknesses of each learning method in the context of testing?
- How can we determine the proper type of learning method for the stages of a testing process?
- Where are the critical points in a software testing process in which ML can positively contribute?

The general purpose of our ongoing research is to provide a specific set of guidelines for automating software testing processes with the aid of machine leaning techniques. To come up with such set of guidelines, it is required to systematically analyse the current research work and find answers to some of the abovementioned questions.

In the first step, we propose a framework which can be used to classify the current research work in the conjunction of ML and ST. In addition, to support our framework, some works are reviewed. This classification framework can assist ST practitioners to analyse and understand the emerging applications of the ML-ST domain. Such structured classification framework can be useful for defining and constructing a set of guidelines for automating software testing processes.

The reminder of this paper is as follows. In Section II , the proposed classification framework is presented. In addition, the dimensions of the framework are discussed in detail. Section III is devoted to presenting some work in the area of ML and ST. In Section IV we discuss how this set of representative work can be classified according to our proposed framework. We conclude the paper with conclusions and direction for future work in Section V.

II. The Proposed Framework for Classification

During the past decade, several works have been introduced based on machine learning techniques where learning algorithms have been applied to the various stages of software testing. In order to clarify the stand point of ML in the area of ST, we propose a classification framework. In this section, we take a look at the proposed framework and its dimensions.

A. The Framework's Dimensions

The most important factors that can be used to distinguish the current research work are represented in Fig. 1. In its highest layer, the framework consists of two main categories

TABLE I
TESTING GENERAL ACTIVITY DIMENTION

Sub-dimension	Possible value
A: Test Planning	Testing cost estimation [9]
B: Test Case Management	Test case prioritization [1] Test case design [1] Test case refinement [6] Test case evaluation [16]
C: Debugging	Fault localization [18] Bug prioritization [18] Fault prediction [15]

(Testing and ML). According to the defined categories, six main dimensions and some sub-dimensions are obtainable in the classification framework. The framework's dimensions are as follows:

Dimension 1-Testing approach: According to the older testing terminology [2], we define the *testing approach* dimension with three possible values: black-box, white-box, and gray-box [2]. Based on the black-box approach, testing can be performed using the external description of a software system such as the software specification. In a white-box approach, the internal properties of a software system like source code can be used for testing purposes. The grey-box approach is a combination of the two, which considers both internal and external properties at the same time.

Dimension 2-Testing general activity: In the second dimension, the standard testing process life-cycle [13] inspired us to define the *a) test planning*, *b) test case management*, and *c) debugging* sub-dimensions. With respect to the testing process life-cycle, we found out that these three phases are critical and that automation can effectively assist them in order to reduce the cost and time of the whole testing process. The possible activities that can be automated based on ML in *a*, *b*, and *c* are presented in Table I.

In the *test planning* sub-dimension, *testing cost estimation* can help test managers to predict testing process cost and time and provide good testing plan to manage the testing process efficiently. *Test case management* includes several tasks such as *test case prioritization*, which intends to prioritize the test input space in terms of test case effectiveness; *test case design*, which intends to generate high quality test cases; *test case refinement*, which intends to map the current specification of a software system to the existing test cases in order to reuse the available test cases; and *test case evaluation* which intends to measure the quality of the generated test cases. In the *Debugging* sub-dimension, *fault localization* can help to find the exact location of the program that is defected. In addition, *bug prioritization* intends to prioritize the revealed faults based on their severities; later test engineer can focus on more critical faults accordingly. *Fault prediction* can assist test engineers in the debugging stage, in the sense that potential faults for a given program are predicted.

Dimension 3-Testing level: Software development process includes a range of stages form "requirement analysis" to "implementation" [13]. The development process goes further even after the implementation stage and concentrates on software maintenance. In Dimension 3, we have employed the following widely accepted testing levels in order to classify existing work: *acceptance testing*, *system testing*, *integration testing*, *module testing*, *unit testing*, and *regression testing*, which refer to requirement analysis, architectural design, subsystem design, detailed design, implementation, and maintenance in the software development stages, respectively.

Dimension 4-Learning technique: In the machine learning area, various types of learning methods have been introduced and each of them has its own specific characteristics. We adopt Mitchel [12] learning method classification and define *learning technique* dimension values as follows: Decision Trees (DT), Artificial Neural Networks (ANN), Genetic Algorithms (GA), Bayesian Learning (BL), Instance Base Learning (IBL), Clustering, and Hybrid methods, which can be combination of several other learning methods.

Dimension 5-Learning property: Each learning method has its own specific characteristics. This dimension is devoted to evaluating learning methods properties in various aspects. To do so, three sub-dimensions; *Training data properties*, *Supervision*, *Time generalization* are defined. The *Training data properties* evaluate the properties of a data set. The gathered data for learning process can be small or large in terms of its quantity. Data can be noisy or accurate in terms of errors that might exist in a data set. In addition, learning can be supervised or unsupervised; therefore, the *Supervision* sub-dimension is defined. For a given target function the *time generalization* sub-dimension exposes how its generalization is; which can be either eager (at learning phase) or lazy (at classification phase). In terms of online and offline learning, also *time generalization* shows how a learning system updates its approximation of the target function after the initial training data are used. The last sub-dimension, *automation degree*, is responsible for addressing the learning system in terms of the degree of automation. The designed system can be fully or partially automatic.

Dimension 6-Elements learned: In each learning system for software testing automation, various types of data can be used to build the target function. The training data can be collected in different stages of the software testing process or software development life-cycle. The learning elements could be *software metrics*, *software specification*, *CFG (control flow graph)*, *call graph*, *test case*, *execution data*, *failure reports*, and/or *coverage data*. The *elements learned* dimension is defined to distinguish the type of learning element that being used for an individual learning model in the testing process.

Fig. 1 shows the framework dimensions, their sub-dimensions and some possible values for them. The main dimensions are shown with ellipse and numbered from 1 to 6. The sub-dimensions are represented with ellipse as well. In addition, the possible values of each dimension are shown with rectangles.

III. THE EXAMPLES OF ML IN SOFTWARE TESTING

In this section, some research work in the area of ML and ST has been selected for supporting the proposed classification framework. In Section IV, we discuss how this work can be classified with proposed framework. Note that, Dimension

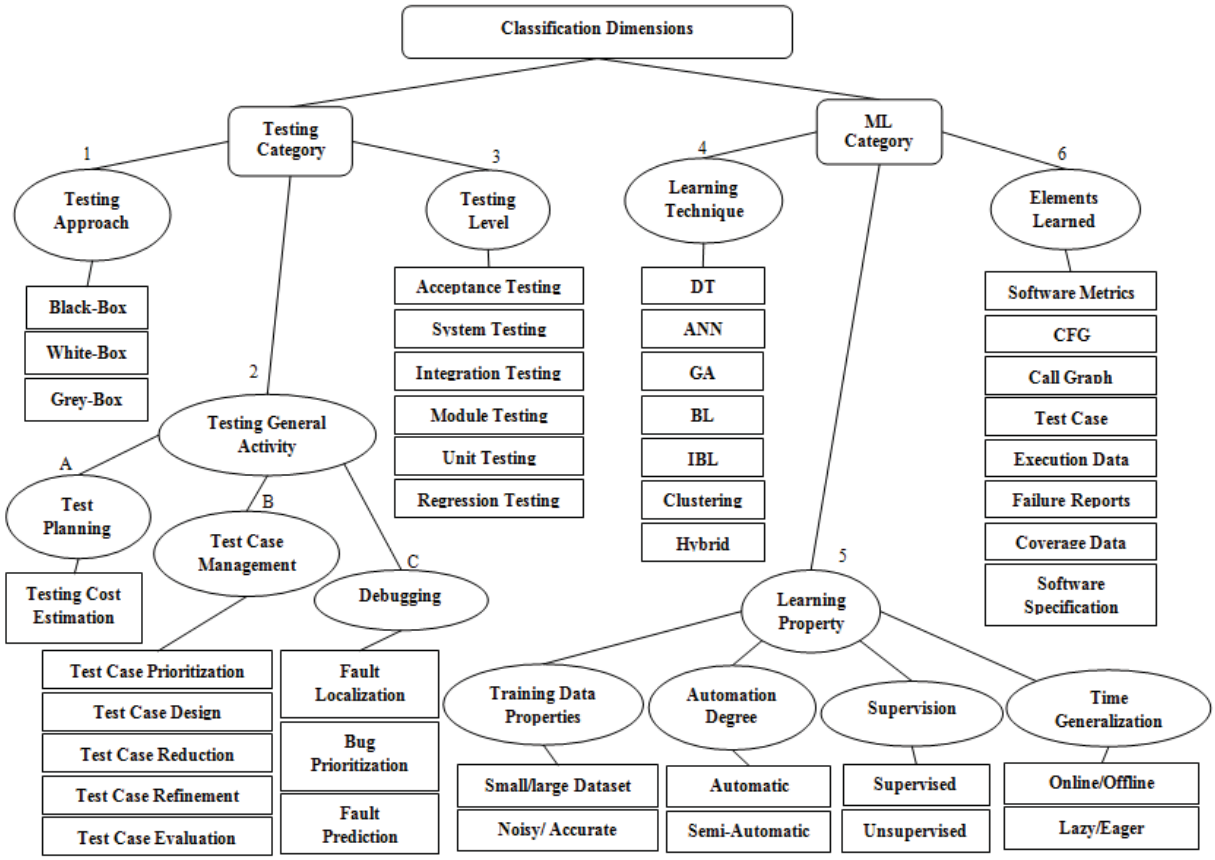


Fig. 1. Classification framework dimensions.

2 has been chosen as the main dimension for classification purposes. We have tried to select and present at least one work according to sub-dimensions *a*, *b*, and *c*. In the following three subsections, we have chosen and briefly introduced three interesting works (due to space limitation) that have employed machine learning techniques for test planning, test case management, and debugging, respectively. Later, we will show how these three works that fall under different aspects of software testing can be effectively described using our proposed classification framework. In our future longer publication, we will comprehensively classify the relevant related literature within our proposed framework. The following three works are only introduced to show the potential of our framework.

A. Test planning

In [9], machine learning techniques have been used to identify the main factors of software testing where they have been employed to predict the testing process time. Understanding the important factors of software under the test can help test managers to prepare a good test plan. The developed methodology consists of four phases; 1) Database formation; 2) Data collection; 3) Classification of software and 4) Analysing the results. In the first phase, databases of 25 software systems were constructed. The sample software were collected from various sources with different attributes. The second phase is devoted to real data extraction form software systems. According to the predefined factors and for a given software system, values were extracted for each individual factor. The list of determined factors included code complexity measures, measures of programmer and tester experience, testing time and a number of other attributes. The complete attributes list includes twenty-seven attributes. In the third phase, COBWEB/3 [11] was used to classify the software systems and build a model to predict the cost of testing for the new software system. COBWEB/3 builds a decision tree. In the last phase, the classification results need to be analyzed to get specific time prediction.

B. Test Case Management

In the context of software evolution, sometimes the available test cases need to be refined to address the current test requirements of the software system. The process of test case refinement is called *test case re-engineering* [6]. In [6], [7], a semi-automatic methodology based on a machine learning approach is provided to explore the limitations and find the possible redundancies of the available test cases and then refine them for future usage. As illustrated in Fig. 2, the MELBA (MachinE Learning based refinement of BlAck-box test specification) methodology is an iterative process and it consists of five main activities.

In the Activity 1, test cases are transformed into the abstract level using the Category-Partition (CP) strategy [14], so they would be ready to be used by the machine learning method in Activity 2. The CP method helps to model the input domain

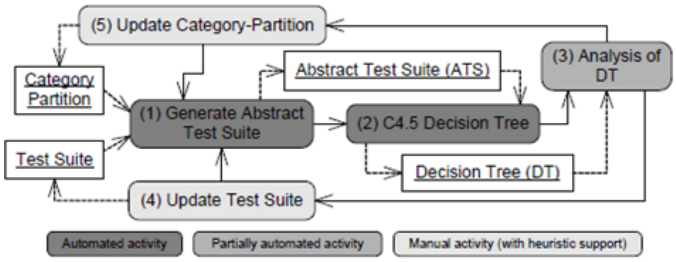


Fig. 2. Overview of the MELBA process [6].

and as discussed in [14], CP requires defining the sets of categories and choices. The categories are the properties of the input and environment parameters of the software system and the choices are the possible values for each category. The abstraction process is conducted using the defined categories and choices. An abstract test case shows an output equivalence class and the pairs of (category, choice). In Activity 2, the machine learning algorithm (C4.5 decision tree) is used to learn the rules for classifying the abstracted test cases. Activity 3 is the important part of MELBA, which is devoted to analysing the learnt rules. The results of the analyzed tree could lead to determine the problems that cause the redundant test cases or the needs for adding new test cases (Activity 4). In addition, the learnt rules may indicate that CP needs to be updated (Activity 5). This iterative process will continue until no more problems could be identified in the trees.

C. Debugging

In software debugging, fault localization refers to process of finding the exact locations of the program that are defected and contain faults. In [18], the Back-Propagation (BP) neural network method [10] is used to effectively pinpoint the program faults. According to proposed approach, first the BP neural network is trained with both the coverage data and the execution result, then the trained network is used to predict and rank the suspicious statements of the code, in terms of its potential to have faults. The proposed method for fault localization consists of the following steps:

1) Building up the dataset: In order to create dataset, it is required to run the program with several test cases and collect the coverage data for each execution. Beside this, the execution results are also need to be collected. This coverage data (coverage vectors) and its execution results can be used for training phase.

2) Training of the BP neural network: In this stage BP neural network is built with k input-layer neurons, three hidden-layer neurons, and one output-layer neuron. The sigmoid function is used as the transfer function. In order to perform training process, coverage vector of each test cases $t_1...t_n$ is used as input data and their corresponding $r_1...r_n$ testing result is used as expected output.

3) Prioritizing the defected statements in program: the trained network can be used for prioritization purpose. Assume, we have a set of virtual test cases $v_1...v_n$ which they cover statements $s_1...s_n$ respectively. It means that, the execution of virtual test case v_i *(i=1...m)* covers only s_i statement. With respect to this assumption, we can say that if the execution of test case v_i fails, then the probability that s_i is defected is high. Here, the trained network can be used to predict the expected output for each virtual test case (r_{vi}). The value of r_{vi} is between 0 and 1. The larger the value of r_{vi}, the more probable s_i is defected. Hence, the statements $s_1...s_m$ can be ranked based on $r_{v1}...r_{vm}$ values in descending order. At the end, the bugs can be identified by examining statements one by one form top of the list.

IV. Classification based on Proposed Framework

In Section II, we introduced a classification framework. There, we discussed the framework's dimensions, sub-dimensions, and some possible values of them. This framework can be used to classify and highlight the main features that need to be considered in the junction of ML and ST area. Following that, in Section III, we presented some representative work in the ML and ST domains.

In this section, we show how the proposed framework can be applied on the three sample presented works. Table II summarizes the results of this classification for each individual work that we have reviewed. In this table, the columns show the dimensions and the related sub-dimensions of the framework and the rows represent the research work that we reviewed. In addition, each cell in the junction of a column and a row indicates the value(s) of that dimension for the corresponding research work.

To clarify, from Table II it can be understood that the automation in [18] has been done in the *debugging* stage of testing process and the *task type* was *fault localization*, which *automatically* finds the suspicious statements in the source codes. In addition, in terms of testing level, this work has been conducted at the *unit testing* level. With respect to the *testing approach* dimension, this work was carried out based on the *white-box* approach, because the statements of the source code are considered for testing purposes. From the ML perspective it can be derived that this work has benefited from the *ANN* learning algorithm which is a *supervised* learning method. Furthermore, the elements which were used as input in the learning process were *coverage data*.

For the work presented in [9], automation has been performed in the *test planning* stage whose *task type* is *testing cost estimation*. The *DT (Decision Tree)* type learning algorithm was used to build a model for estimating testing process cost. From the *automation degree* point of view, this work proposed a *semi-automatic approach*, since in some parts of this method we can see the need for test engineer's intervention. Furthermore, *software metrics* have been employed as input for the learning process. In this work, the learning task has been conducted using a *small size* of dataset, which can considerably decrease the learning process time.

Finally, in [6], [7], *testing general activity* is addressed by *test case management* value, which means, for automation purposes the learning method was applied in the *test management* stage in the software testing life-cycle. The *task type* in this work was *test case refinement*. The proposed method in

TABLE II
CLASSIFICATION OF SOME RESEARCH WORK IN ML AND ST BASED ON PROPOSED FRAMEWORK

Research Work	Testing Category				ML Category					
						Learning Property				
	Testing General Activity	*Task Type*	*Testing Level*	*Testing Approach*	*Learning Technique*	*Training data properties*	*Automation Degree*	*Supervision*	*Time generalization*	*Elements Learned*
[9]	Test planning	Testing cost prediction	-	-	DT	Small dataset	Semi-automated	Unsupervised	Offline/Eager	Software metric
[6][7]	Test case management	Test case refinement	Regression testing	Black-box	DT	Small dataset	Semi-automated	Supervised	Offline/Eager	Test Case/software specification
[18]	Debugging	Fault localization	Unit testing	White-box	ANN	-	Automatic	Supervised	Offline/Eager	Coverage data

this work, in terms of testing level, can be categorised in the *regression testing* level, because this work focused on reusing the existing test cases for new updated version of the current software system. With regards to the *testing approach*, this work used the *black-box* method, since it considers only the software specification for testing purposes. From the machine leaning point of view, this work employed a *semi-automatic* approach using *DT*- based learning technique. The dataset size was quite small and the elements which have been used to build dataset were *test cases* and *software specification documents*.

Table II and the above classification show how our proposed framework is useful in categorizing and explaining work in the intersection of ML and ST.

V. Conclusion and Future work

Testing is critical task in the software development process and considerably imposes cost and time restrictions on the development process. Therefore, automating the testing process can significantly increase testing process performance. In the context of software testing, different types of data can be collected in various stages of testing. Machine learning techniques can be used to find patterns in this data and use them for automation purposes.

In this paper, we have introduced a framework for classifying the current research works in ML and ST. Some sample works of ML in ST have been reviewed and appropriately classified according to the proposed framework. The proposed framework provides us with the opportunity to systematically investigate and extract the prominent information from existing research works in ML and ST.

Considering the introduced framework, our main attempt for future research is reviewing and classifying all current work in the area of ML and ST. The collected information from classification process will assist us to construct the sets of concrete guidelines for applying ML methods in the ST process. These guidelines will help test engineers to choose the most efficient and appropriate learning method for automation of a target testing stage. Our initial probe has shown that our classification framework is quite strong in providing the means to capture various aspects of work in ML and ST.

References

[1] Moataz A. Ahmed and Irman Hermadi. Ga-based multiple paths test data generator. *Comput. Oper. Res.*, 35:3107–3124, October 2008.

[2] Paul Ammann and Jeff Offutt. *Introduction to software testing*. Cambridge University Press, 2008.

[3] Boris Beizer. *Software testing techniques (2nd ed.)*. Van Nostrand Reinhold Co., New York, NY, USA, 1990.

[4] Antonia Bertolino. Software testing research: Achievements, challenges, dreams. In *2007 Future of Software Engineering*, FOSE '07, pages 85–103, Washington, DC, USA, 2007. IEEE Computer Society.

[5] Lionel C. Briand. Novel applications of machine learning in software testing. *Quality Software, International Conference on*, 0:3–10, 2008.

[6] Lionel C. Briand, Yvan Labiche, and Zaheer Bawar. Using machine learning to refine black-box test specifications and test suites. In *Proceedings of the 2008 The Eighth International Conference on Quality Software*, Washington, DC, USA, 2008. IEEE Computer Society.

[7] Lionel C. Briand, Yvan Labiche, Zaheer Bawar, and Nadia Traldi Spido. Using machine learning to refine category-partition test specifications and test suites. *Inf. Softw. Technol.*, 51, November 2009.

[8] Lionel C. Briand, Yvan Labiche, and Xuetao Liu. Using machine learning to support debugging with tarantula. In *Proceedings of the The 18th IEEE International Symposium on Software Reliability*, ISSRE '07, pages 137–146, Washington, DC, USA, 2007. IEEE Computer Society.

[9] Thomas J. Cheatham, Jungsoon P. Yoo, and Nancy J. Wahl. Software testing: a machine learning experiment. In *Proceedings of the 1995 ACM 23rd annual conference on Computer science*, CSC '95, pages 135–141, New York, NY, USA, 1995. ACM.

[10] Laurene Fausett, editor. *Fundamentals of neural networks: architectures, algorithms, and applications*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1994.

[11] Kathi Een Mckumcr, Kevin Thompson, Uncl As, Kathleen Mckusick, and Kevin Thompson. Cobweb/3: A portable implementation, 1990.

[12] Tom M. Mitchell. *Machine learning*. McGraw Hill series in computer science. McGraw-Hill, 1997.

[13] Glenford J. Myers and Corey Sandler. *The Art of Software Testing*. John Wiley & Sons, 2004.

[14] T. J. Ostrand and M. J. Balcer. The category-partition method for specifying and generating fuctional tests. *Commun. ACM*, 31:676–686, June 1988.

[15] Susan A. Sherer. Software fault prediction. *Journal of Systems and Software*, 29(2):97 – 105, 1995.

[16] A. von Mayrhauser, C. Anderson, and R. Mraz. Using a neural network to predict test case effectiveness. In *Aerospace Applications Conference, 1995. Proceedings., 1995 IEEE*, number 0, pages 77 –91 vol.2, February 1995.

[17] Joachim Wegener, Andre Baresel, and Harmen Sthamer. Evolutionary test environment for automatic structural testing. *Information and Software Technology*, 43(14):841 – 854, 2001.

[18] W. Eric Wong and Yu Qi. Bp neural network-based effective fault localization. *International Journal of Software Engineering and Knowledge Engineering*, 19(4):573–597, 2009.

[19] Du Zhang and Jeffrey Tsai. Machine learning and software engineering. *Software Quality Journal*, 11:87–119, 2003. 10.1023/A:1023760326768.

A Model-based Approach to Regression Testing of Component-based Software *

Chuanqi Tao, Bixin Li
School of Computer Science and Engineering
Southeast University, Nanjing, China
chuanqi.tao@sjsu.edu, bx.li@seu.edu.cn

Jerry Gao
School of Computer Engineering
San Jose State University, San Jose, USA
gaojerry@email.sjsu.edu

***Abstract*—Component-based software systems consist of various components, such as third-party components and in-house built components. Due to the component changes, a software system is usually affected at both component level and system level. Related existing research does not address the issue of systematic regression testing of component-based software, especially at system level. This paper proposes a systematic regression testing method from components to system. The paper discusses component API changes, interaction changes, and architecture changes. In addition, it presents a component-based change impact analysis method based on the proposed component firewalls and uses a decision table as its test model. The provided approach is applied throughout the regression testing process. Finally, the paper reports our case studies based on a realistic component-based software system. The study results show that the approach is feasible and effective.**

***Keywords*-component-based software regression testing; software maintenance; retest model; change and impact analysis; test cases update**

I. Introduction

Component-based software is widely used nowadays. The modern software system is primarily constructed based on reusable components, such as third-party components and in-house built components. During software maintenance, when a component is updated or upgraded, it must be retested. This refers to regression testing, which is an important task of software maintenance. Its major objective is to gain the quality confidence for the updated software whenever it is changed. According to [1], regression testing is a major task of software maintenance and it accounts for more than one-third of its total costs. For any component-based software, its regression testing can be conducted in a hierarchical manner that is from the component-level unit retest, component reintegration, to the system level regression testing.

In regression testing of component-based software, the research topics still focus on *re-test model*, *change impact analysis* and *test case update*. However, changes made to a component could bring impact on the other parts of the component, which means the change impact could affect other components of the system or the whole system behaviors. In addition, component-based regression testing should take practical test models into account. For instance, if a decision table-based method has been selected as a test model to generate test cases, regression testing should consider how to identify test cases change and impact based on the decision table.

*Supported partially by the National Natural Science Foundation of China under Grant No. 60773105 and no. 60973149, and partially Supported by Doctoral Fund of Ministry of Education of China under Grant No. 20100092110022, and partially by National High Technology Research and Development Program under Grant No. 2007AA01Z141 and No. 2008AA01Z113.

Research in the past seldom discussed how to identify component changes and impacts in component-based software. Although some papers discussed regression testing of component [2–5], these papers didn't address regression testing at different levels from component to system. Nowadays, practitioners in the real world are looking for systematic solutions to support component and system regression testing and component evolution.

This paper addresses those needs above by providing a systematic approach to regression testing of component-based software based on re-test models. Re-test models are usually used to present the dependency relationships amongst components, assist engineers to define re-test criteria and re-integration strategies, and facilitate automatic test generation. To support regression testing, related modes need to be chosen. This paper proposes several models according to diverse views of component testing. In those models, the relationships between functions or data at component level, between components at system level are all taken into account.

Change identification is the first step of regression testing. A clear classification of change types can support effective impact analysis. The various change types are summarized at different levels in this paper. Impact analysis is an important task in regression testing. The component firewall concept is utilized as a primary method for our change impact analysis. This paper presents diverse firewalls based on the re-test models at both component level and system level. Test case update is one of the goals of regression testing. As a complete regression testing solution, the original test cases should be updated for reusable test cases, obsolete test cases and new test cases. To identify the affected test cases, the affected components or parts should be mapped into the corresponding test cases. Thus, the test models which are used for generating test cases need to be analyzed. Since the *decision table*-based testing is an important component testing method and the project of our case study also utilizes *decision table* as the major testing method, therefore, we regard *decision table* as a basic test model for our regression testing in this paper. The related test case update is based on the *decision table*. Therefore, the process to perform regression testing of component-based software from component to system is as following steps:

Step 1 Change identification at component level, which refers to API function changes, API data changes, structural changes, etc.

Step 2 Impact analysis at component level, which refers to component function firewall, component data function firewall and component API function firewall.

Step 3 Test cases update at component level, which refers to component decision table-based test cases reused, deleted and added.

Step 4 Change identification at system level, which refers to component interaction changes and architecture changes.

Step 5 Impact analysis at system level, which refers to extended composition tree firewall and component interaction firewall.

Step 6 Test cases update at system level, which refers to system decision table-based test cases reused, deleted and added.

The major contributions of this paper are summarized below:

(1) A systematic solution to regression testing of component-based software from component level to system level is presented, including the process of regression testing : change identification, change impact analysis and test cases update.

(2) Several new component firewalls are introduced for component change impact and test case reuse in component-based software regression testing.

(3) Practical application experiments are performed in our case studies. Nearly a hundred students joined related empirical studies.

This paper is organized as follows. Section 2 introduces a brief review of the related work in regression testing of component-based software. In section 3, the re-test models are presented. Section 4 analyzes the change types and provides a systematic approach to component-based software change impact analysis using firewall. Section 5 discusses test cases update. Section 6 reports the case study. Conclusion and future work are summarized in the end.

II. Related Work

In the past decades, a number of papers have been published for regression testing issues for conventional programs and object-oriented software. Those papers primarily focus on three issues: regression test selection techniques [6, 7, 9, 10], regression testing cost-effectiveness analysis [11, 12], and object-oriented re-integration [13–15].

A lot of papers focus on the component-based software testing issues. Currently, component-based software testing mainly focuses on component testability, component test adequacy and coverage, component-based software integration, performance testing, and configuration testing. Recently, a few papers addressed the regression testing problems existed in component-based software [2, 3, 16–18]. They can be generally classified into the following three groups.

The first group is regression test selection of component-based software. For example, Harrold et al. proposed an approach to regression testing of COTS components using component meta-data [16]. They utilized three types of meta-data to perform the regression test selection. However, the method needs additional information from component which may be not available in practice. Similarly, Orso et al. also discussed two techniques for regression testing of component-based software [2]. The first is code-based and the second is specification-based. Both techniques are based on the provided component meta-data. To support the approach, the additional information is needed, including the version information, change data and coverage measurement facilities. Zheng et al. proposed an *Integrated-Black-box Approach for Component Change Identification* for COTS(Commercial-off-the-shelf) software [17]. For the third-party component, the internal software information could be available from component specification, user interface and reference manual. To support the approach, binary code and document should be visible. They assumed that when components change and only binary code and documentation are available, regression test selection can safely be based upon the glue code that interfaces with sections of the component that changed. Robinson et al. proposed a firewall method for regression testing of user-configurable software. They constructed a firewall to identify the impacted area in system based on setting changes and configurable element changes respectively, then created or selected test cases to cover the impacts [19].

The second group is UML-based. For instance, Wu et al. presented a UML technique for regression testing of component-based software [4]. They adopted UML diagrams, which represent changes to a component, to support regression testing. Class diagrams, Collaboration diagrams, and Statechart diagrams are considered to be as the re-test models.

The third group is the systematic method based on API models. Gao et al. focused on component API-based changes and impacts, and proposed a systematic re-test method for software components based on a component API-based test model [3]. In addition, Mao et al. proposed an improved regression testing method based on built-in test design for component-based system [5].

The open questions and challenges of regression testing of component-based software are primarily as follows.

-How to identify the diverse component changes in a systematic way?

-How to do impact analysis from component to system?

-How to update both component and system test cases?

This paper addresses those problems above. Unlike previous work that only focused on component change analysis and impact at the component level, this paper proposes a model-based approach for regression testing. A firewall approach is utilized for impact analysis from component to system to find out ripple effects on other components and system behaviors. Moreover, this paper also introduces a systematic methodology for regression testing of component-based software from component to system.

III. Component-based Software Re-test Models

Component and system can be viewed from different perspectives for testing. For instance, a component can have *white-box* view, *black-box* view, *API* view or *performance* view. A system can have *integration* view, *configuration* view, *function* view or *performance* view. Component testing researchers can choose different views according to their test plan and test goal. Various views of components correspond to different re-test models.

At component level, the changes could be structure changes or internal logic changes. If each updated component is assumed to provide the component internal information as its meta-data, from the white-box view, the white-box re-test models like *data function dependency* or *function dependency* could be adopted for component re-test. Since component function is usually used through API function or data call, the API models can be borrowed for component re-test. From the black-box view, the component could be tested using *decision table-based* testing or *state-based* testing. Hence, the related test models could be adopted for component testing. In addition, from the performance view, the scenario-based testing models can also be applied. However, performance is out of the scope of this paper.

At system level, the relationship between components could be *interaction*, *composition*, *message communication*, etc. Thus, re-

lated system-level models are proposed for system regression testing. For example, from the integration view, *component interaction graph* can be used as interaction models for system. As component system is usually configurable, the *composition* and *configuration* models are needed to support the system testing. From the *function* view, *decision table-based* or *state-based* methods could also be adopted for system function testing.

The reason why we adopt those models to support the regression testing of component-based software from component to system is explained above. In the next few subsections, we will define and introduce those models in detail.

A. System-level Re-test Models

At the system level, from the integration view and configuration view, we propose two re-test models. Now we introduce those two models in detail.

1) ***Extended Composition Tree****: Extended Composition Tree* (*ECT*) describes the *composition* and *configuration* relationship amongst components or inside sophisticated components. *ECT* extends traditional *composition tree* through adding the *configuration* relationship [20]. For instance, an elevator system is composed of the *car*, the *floor panel*, the *controller*, and etc. Sophisticated component *car* contains *user panel*, *door*, *car controller*, etc. Regarding configuration, a *door* component can be configured with *Single door* or *Double door*.

ECT consists of tree nodes and links. Tree nodes present single or sophisticated components including configurable components. Tree links in *ECT* present the composition relation or configuration relation between tree nodes.

Definition 1 An *Extended Composition Tree(ECT)* can be defined a directed graph $ECT = (N, E, R)$, *where* $N = \{N_1, N_2, ..., N_n\}$ *is a finite set of nodes,* $R = \{POW, EXT, EOR, AND, Switch, Multiplex\}$ *is the set of relations,and* $E = E_{POW} \cup E_{EXT} \cup E_{EOR} \cup E_{Multiplex} \cup E_{AND} \cup E_{Switch}$ *is the set of edges.*$E_{POW} \subseteq N \times N \times R$ *is the set of directed edges representing the* part-of-whole *relation between the components. For any two components* $C_1, C_2 \in N$, $\langle C_1, C_2, POW\rangle \in E_{POW}$ *indicates that component* C_2 *is a part of* C_1. $E_{EXT} \subseteq N \times N \times R$ *is the set of directed edges representing the* extend *relation between the components. For any two components* $C_1, C_2 \in N$, $\langle C_1, C_2, EXT\rangle \in E_{EXT}$ *indicates that component* C_2 *extends* C_1 *through composition.*

Regarding *configuration* relation, we already defined the related models in our previous work [20].

2) ***Component Interaction Graph****:* Wu et al. introduced the *component interaction graph* to depict interactions and dependance relationships among components , which is mainly call relationship [21]. Here, *Component Interaction Graph (CIG)* addresses the relationships include *message-communication*, *usage*, etc. Component interaction graph is defined as *Definition 2* below.

Definition 2 *Component interaction graph(CIG) for software components is a directed graph* $CIG = (N, E, R)$, *where* N *is the set of nodes representing the components,* $R = \{MSG, USA\}$ *is the set of interaction relations, and* $E = E_{MSG} \bigcup E_{USA}$ *is the set of edges defined below.* $E_{USA} \subseteq N \times N \times R$ *is the set of directed edges representing the* usage *interaction relation between the components.* $E_{MSG} \subseteq N \times N \times R$ *is the set of directed edges representing the* message-based *interaction relation between the components.*

	Precondition				Action				Postcondition			
	$prec_1$	$prec_2$	·····	$prec_m$	A_1	A_2	·····	A_k	$postc_1$	$postc_2$	·····	$postc_n$
T1	T	F	·····	–	X		·····		T	–	·····	–
T2	F	T	·····	–		X	·····	X	–	F	·····	T
···			·····				·····				·····	
Ti	T	T	·····	F		X	·····		T	F	·····	T

Figure 1. A Sample of Component Semantic Decision Table

For instance, *For any two components* $C_1, C_2 \in N$, $\langle C_1, C_2, MSG\rangle \in E_{MSG}$ *indicates that component* C_1 *sends message to* C_2.

B. Re-test Models for Test Cases Reuse

1) ***Component Semantic Decision Table Test Model****: Decision Table* is a black-box testing method focusing on validating business rules, conditions, constraints and corresponding responses and actions of a software component [22, 23]. The basic approach is to identify and list all possible conditions and their combination cases as well as responding actions and outputs for each case, then define test cases to cover each case. In this paper, we propose a new *Component Decision Table* for component. We call this *Component Semantic Decision Table* (*CSDT*). For each case, there exists *precondition*, *action* and *postcondition*.

According to *component black-box view*, the data value of *preconditions* could be *Incoming data*, *Incoming message*(such as GUI input data or component internal data) or *API call parameter*, and the data value of *postcondition* could be *Outgoing data*(such as database table, GUI output, output data file, etc.), *Outgoing message* or *Outgoing call data*. All these *preconditions* and *postconditions* could be obtained from component specification. *Action* in *CSDT* stands for component *function*. Figure 1 shows a sample component semantic decision table. The *precondition*, *postcondition* and *action* are explained in *Definition 3*.

Definition 3 *The CSDT includes three sets, which is presented below.* $prec = \{prec_1, prec_2, ..., prec_m\}$, $postc = \{postc_1, postc_2, ..., postc_n\}$, *where* prec *denotes a* precondition *set, which includes precondition* $prec_1, prec_2, ..., prec_n$, *and* postc *denotes a* postcondition *set, which includes postcondition* $postc_1, postc_2, ..., postc_n$. $action = \{A_1, A_2, ..., A_l\}$, *where* action *denotes a* action *set, which includes action* $A_1, A_2, ..., A_l$.

2) ***System Semantic Decision Table Test Model****:* In our component system specification, there is a *feature-component table*. The table describes the relation between system features and corresponding components. These features can be observed at the system level. For each feature, there is a corresponding *decision table*.

The system features can be defined as a set Ft, $Ft = \{ft_1, ft_2, ..., ft_n\}$, where Ft denotes the function feature set of component-based system, which includes feature $ft_1, ft_1, ..., ft_n$. Each *feature* is supported by a set of components. The relation between feature and components can be expressed below.

$ft_i \longrightarrow C(ft_i) = \{C_1, C_2, ..., C_m\}$, *where* $C(ft_i)$ *denotes the set of components which support feature* ft_i.

Each system decision table can be used for each feature, which is presented below.

$ft_i \longrightarrow SDT_i$, where SDT_i stands for system decision table.

$SDT_i \longrightarrow STd_i$, *where* STd_i *stands for test case set for decision table* SDT_i.

Type	Specific Changes
ADP	Add a data parameter
AF	Add a function
ARD	Add a 'return' data
DDP	Delete a data parameter
DF	Delete a function
DRD	Delete a 'return' data
CDT	Change data type or name
CFL	Change the function logic
CFS	Change the interface signature

Type	Specific Changes
AMO	Add Message-based (MB) relation (outgoing)
AMI	Add MB (incoming)
AMB	Add MB (both way)
AU	Add Usage relation
DMO	Delete MB (outgoing)
DMI	Delete MB (incoming)
DMB	Delete MB (both way)
DU	Delete Usage relation
CMS	Change MB semantic

Figure 2. Component Function and Interaction Change Types

Type	Specific Changes
APW	Add a 'part of whole' relationship (add C1 as a part of C2)
AE	Add an 'extends' relationship (add C1 to extend C2)
ACDV	Add a 'configurable' data value
ACDT	Add a 'configurable' data type
ACF	Add a 'configurable' function
DPW	Delete a 'part of whole' relationship (delete C1 from C2)
DE	Delete an 'extends' relationship (delete C1 from C2)
DCDV	Delete a 'configurable' data value
DCDT	Delete a 'configurable' data type
DCF	Delete a 'configurable' function
CCDV	Change a 'configurable' data value
CCDT	Change a 'configurable' data type
CCF	Change a 'configurable' function

Figure 3. Architecture Change Types

IV. Change Impact Analysis

We have summarized the common changes existed in component-based software in Figure 2 and 3. Component changes could be classified at two levels: **a)** the component level, and **b)** the system level. Change impact analysis is based on the models proposed in section 3. We introduce several component change firewalls, which include change, add and delete firewalls. The basic procedure to perform a component change impact consists of the following steps:

1) Identify the impacts of a changed component function or data on other component functions at component level; 2) Identify the component change impacts on its API; 3) Identify the component change impact on its precondition, action, and postcondition; 4) Identify the component change impact on other components based on architecture at system level; 5) Identify the component change impact on other components based on interaction at system level.

For any component, we assume $ECT = (N, E, R)$ be the old version of its *Extended Composition Tree*, $ECT' = (N', E', R')$ be its new version, $CIG = (N, E, R)$ be the old version of its *Component Interaction Graph*, and $CIG' = (N', E', R')$ be its new version.

A. Component function, precondition and post-condition Firewall

Through the computation of *CFFW* and *CDFW*, we can get the affected component functions based on invocation dependencies and data *define-use* dependencies [3]. Various change types correspond to different impact. Now we try to present the change impact analysis corresponding to the summarized component level change types in Figure 2 using firewall.

- For ADP or ARD (the changed data is assumed as D_i):
  $CDFW_{add}[D_i] = \{F_j | (\exists F_j)(\exists F_k)((\langle F_k, F_i, du\rangle \in R'_d - R_d) \wedge (F_k \in F) \wedge (\langle F_j, F_k, du\rangle \in R'_d *))\}$
  Where R'_d is the data define-use relation derived from DFDG'.
- For AF (the changed function is assumed as F_i):
  $CFFW_{add}[F_i] = \{F_j | (\exists F_j)(\exists F_k)((\langle F_k, F_i\rangle \in E' - E) \wedge (F_k \in F) \wedge (\langle F_j, F_k\rangle \in R'_f *))\}$
  Where R'_f is the dependence relation for FDG'.
- For DDP and DRD (the changed data is assumed as D_i):
  $CDFW_{delete}[D_i] = \{F_j | (\exists F_j)(\exists F_k)((\langle F_k, F_i, du\rangle \in R_d - R'_d) \wedge (F_k \in F) \wedge (\langle F_j, F_k, du\rangle \in R' *_d))\}$
- For CDT (the changed data is assumed as D_i):
  $CDFW_{change}(D_i) = \{F_i | (\langle F_i, D_j\rangle \in R_{def}) \wedge ((\langle F_j, D_j\rangle \in R_{use}) \wedge (\langle F_j, F_i, du\rangle \in R *_{dr})\}$
  Where $R *_{dr}$ is the transition closure of R_{dr}, which is the binary relation that define the data define-use relation between the residual component functions. R_{dr} can be defined as:
  $R_{dr} = R_d \cap (F \times F \times \{du\}) \cap (F' \times F' \times \{du\})$
- For CFL and CFS (the changed function is assumed as F_i):
  $CFFW_{change}(F_i) = \{F_j | (F_j, F_i \in F) \wedge (\langle F_j, F_i\rangle \in R *_{fr})\}$
  Where $R *_{fr}$ is the transition closure of R_{fr}, which is the binary relation that define the dependencies between the residual component functions. R_{fr} can be defined as:
  $R_{fr} = R_f \cap (F \times F \times \{du\}) \cap (F' \times F' \times \{du\})$, where x is the Cartesian product operation.

Regarding API function firewall, our previous work already discussed it [3]. The definition is as follows.

$CAW_{API}(C) = \{F_i | \forall F_i((F_i \in F'_{API} - F_{API}) \wedge ((F_i \in CFFW(C)) \vee (F_i \in CDFW(C))))\}$

Where $CAW_{API}(C)$ includes all API functions which may be affected by component function changes, deletions and additions, as well as alters in the *data-define-use* relations between them.

Now we extend $CAW_{API}(C)$ by adding the *precondition*, *post-condition*, and *action*. Here, action corresponds to API function. *precondition* and *postcondition* change information can be obtained from the component requirement. The new firewall is called ECAW(*extended component API function firewall*). Thus, the new firewall set can be represented as below.

$ECAW(C) = \{\langle prec_i, postc_j, action_k\rangle | (prec_i$ is changed, added or deleted) $\vee (postc_j$ is changed, added or deleted) $\vee (action_k \in CAW_{API}(C))\}$

Where $prec_i$, $postc_j$, $action_k$ denote *precondition*, *postcondition* and *action* respectively. $\langle prec_i, postc_j, action_k\rangle$ stands for a vector set, which include the affected *preconditions*, *postconditions* and *actions*.

B. Extended Composition Tree Firewall

Extended Composition Tree Firewall (ECTF) in *Extended Composition Tree* after changing components or configurations, refers to a set of components which might be affected by changing, adding and deleting components or configurations based on composition

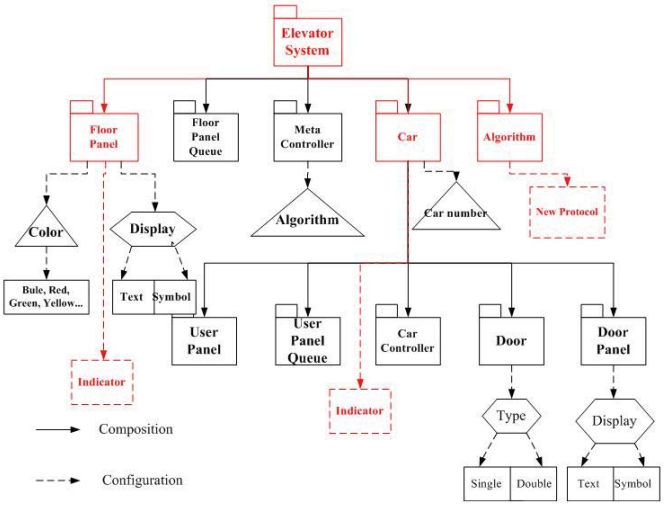


Figure 4. A Sample Extended Composition Tree Firewall in the Elevator System

and configuration dependencies. *ECTF* is performed on architecture. Thus, it is used for identifying the affected architecture at the system level. For instance, in the elevator system, if the *door* component is changed, then the *car* component, which has the composition relation with *door*, will be affected. The affected test cases corresponding to this change mainly refers to unit test cases.

The *Extended Composition Tree Firewall* can be computed based on *Extended Composition Tree*. Now we introduce changed, added and deleted extended composition tree firewalls respectively.

The firewall for a changed, added, and deleted component C_i can be computed below.

$ECTF_{change}[C_i] = \{(C_j | C_j \in C) \wedge (\langle C_j, C_i \rangle) \in R_{ECTr}*\}$

Where $R_{ECTr}*$ is the transitive closure of R_{ECTr}, which is the binary relation that defines the composition dependencies between the components. R_{ECTr} can be defined as follows:

$R_{ECTr} = R_{ECT} \cap (N \times N) \cap (N' \times N')$

Where $\times$ is the Cartesian product operation.

$ECTF_{add}[C_i] = \{C_j | (\exists C_j)(\exists C_k)((\langle C_k, C_i \rangle \in E' - E) \wedge (C_k \in N) \wedge (\langle C_j, C_k \rangle \in R'_{ECT}))\}$

Where R'_{ECT} is the composition dependence relation for *ECT'*.

$ECTF_{delete}[C_i] = \{C_j | (\exists C_j)(\exists C_k)((\langle C_k, C_i \rangle \in E - E') \wedge (C_k \in N) \wedge (\langle C_j, C_k \rangle \in R'_{ECT}))\}$

Now we present the change impact analysis corresponding to the summarized architecture change types in Figure 3 using firewall. The left side of arrow represents change type set and the right side of arrow denotes the corresponding firewall.

$(APW, AE, ACDV, ACDT, ACF) \rightarrow ECTF_{add}[C_i]$

$(DPW, DE, DCDV, DCDT, DCF) \rightarrow ECTF_{delete}[C_i]$

$(CCDV, CCDT, CCF) \rightarrow ECTF_{change}[C_i]$

For instance, in the elevator system, after adding an *indicator* in the *car* component and the *floor panel* component and a new protocol , the *extended composition tree firewall* (ECTF) is shown in highlighted in Figure 4.

C. *Component Interaction Graph Firewall*

Component Interaction Graph Firewall (CIGF) in *Component Interaction Graph* after changing components, refers to a set

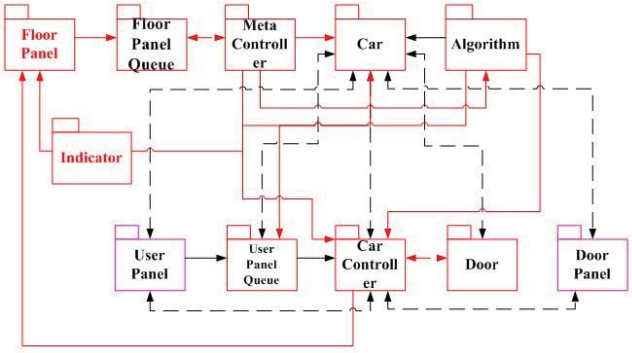


Figure 5. A Sample Component Interaction Graph Firewall in the Elevator System

of components which maybe affected by changing, adding and deleting components based on interaction dependence. The *Component Interaction Graph Firewall* can be computed based on given *Component Interaction Graph*. For instance, in the elevator system, after adding an *indicator* in the *car* component and the *floor panel* component, the *component interaction graph firewall* (CIGF) is shown in highlighted in Figure 5. However, if we only consider component changes, the size of the *Component Interaction Graph Firewall* could be very large. Some components might be redundant for regression testing. API is the function interaction for other components to call. The Algorithm is shown in Algorithm 1. In the case of the interaction change types in Figure 2, the firewall could be computed by algorithm of *Component Interaction Graph firewall*.

Algorithm 1 Component Interaction Graph Firewall

Declare: C_0: changed component;
CIG: component interaction graph;
CIG': modified component interaction graph;
$CIGF_{API}(C_i.F_i, CIG, CIG')$: Component interaction graph firewall
$CIGF_{API}(C_i.F_i, CIG, CIG')$
{
switch (change type)
  case 'Add functions':
    $CFFW_{add}[C_i.F_i]$;
    $CDFW_{add}[C_i.F_i]$;
    $S_F = CFFW_{add}[C_i.F_i] \cup CDFW_{add}[C_i.F_i]$; //function firewall inside C_i
    $C.F = ECAW[C_i.F_i]$; //API function firewall
    break;
  case 'delete functions':
    $CFFW_{delete}[C_i.F_i]$;
    $CDFW_{delete}[C_i.F_i]$;
    $S_F = CFFW_{delete}[C_i.F_i] \cup CDFW_{delete}[C_i.F_i]$;
    $C.F = ECAW[C_i.F_i]$;
    break;
  case 'change functions':
    $CFFW_{change}[C_i.F_i]$;
    $CDFW_{change}[C_i.F_i]$;
    $S_F = CFFW_{change}[C_i.F_i] \cup CDFW_{change}[C_i.F_i]$;
    $C.F = ECAW[C_i.F_i]$;
    break;
Mark each function in C.F visited;
put *C.F* in CIGF;
$C_j = CIG[C_i].rlink$; // $\langle C_j, C_i \rangle \in R_{CIG}$
While ($C_i.f_i$ in $C.F$) do
{
if $(C_j.F_j, C_i.f_i) \in P(C_j) \wedge C_j.F_j$not visited
then
$CIGF_{API}(C_j.F_j, CIG, CIG')$
}
}

V. Test Cases Update

The final goal of regression testing is to refresh the existing test cases for the previous version, which means selecting reusable test cases and scripts, deleting out-of-date test cases and scripts, and adding new test cases and scripts.

As we mentioned above, *decision table* based testing is the primary method to generate test cases in this paper. Now we need to analyze how to map the change impact firewall into affected test cases in the given test suite. We introduce the concept of test firewall to emphasis the test cases updating. The procedure to perform test firewall analysis can be divided in two steps: 1) Identify the change impacts on component decision table-based test cases at component level; 2) Identify the change impacts on system decision table-based test cases at system level.

A. Decision table-based test firewall at component level

In any of the given *component semantic decision table* (CSDT), if any of the *preconditions*, *postconditions* and *actions* are added, deleted or changed, then the corresponding test cases could be affected.

According to the requirement changes, we can identify the *component semantic decision table* (CSDT) changes easily. As the test model defined in section 3, the CSDT changes could come from three sets: *precondition*, *action* and *postcondition*. The problem here is how to identify the reused, changed and new test cases according to the changes.

Each entry value in a row of a given decision table CSDT could be "True", "False", or "-". "True" means the corresponding *precondition* or *postcondition* has to be true; "False" means the corresponding *precondition* or *postcondition* has to be false. "-" means the corresponding *precondition* or *postcondition* could be either true or false. The table includes three sets, which is presented below. The entry value of precondition and postcondition can be defined as a set respectively like below.

$vprec = \{vprec_1, vprec_2, ..., vprec_m\}$, where *vprec* denotes the entry value set of *precondition*.

$vpostc = \{vpostc_1, vpostc_2, ..., vpostc_n\}$, where *vpostc* denotes the entry value set of *postcondition*.

Assumptions *The value of precondition associated with test cases could be* $vprec_i(i \leq m) =$ "T"or"F"or" − ". *The value of postcondition associated with test cases could be* $vpostc_j(j \leq n) =$ "T"or"F"or" − ". *Assuming* $T_p \longrightarrow \langle vprec, A, vpostc \rangle$, *where* T_p *stands for decision table based test case. Vprec,* $A(A \subseteq action)$ *and* $vpostc_j$ *stand for the associated precondition prec, action A and postcondition postc respectively.*

Through the computation of ECAW, we can get the firewall at component level. Assuming CTd_{v1} denotes the original component decision table test cases set, now we need to get the updated version CTd_{v2}. According to the assumptions, we have the following rules for component decision table test firewall.

Rule 1 (corresponding to change firewall):
For any i, j, k
(1)**If** $((prec_i$ or $postc_j \in ECAW_{change}(C)) \wedge (T_p \rightarrow vprec_i =$ "T"or"F" $\vee\ T_p \rightarrow vpostc_j =$ "T"or"F"$))\vee\ (T_p \rightarrow A_k \in ECAW_{change}(C))$
then T_p need to be changed in CTd_{v2}.

(2) **If** $((prec_i$ or $postc_j \in ECAW_{change}(C)) \wedge (T_p \rightarrow vprec_i =$ " − " $\wedge\ T_p \rightarrow vpostc_j =$ " − "$))\wedge\ (T_p \rightarrow A_k \notin (ECAW_{change}(C) \cup ECAW_{add}(C) \cup ECAW_{delete}(C)))$
then T_p need to be reused in CTd_{v2}.
Rule 2 (corresponding to add firewall):
For any i, j, k
(1)**If** $((prec_i$ or $postc_j \in ECAW_{add}(C)) \wedge (T_p \rightarrow vprec_i =$ "T"or"F" $\vee\ T_p \rightarrow vpostc_j =$ "T"or"F"$))\vee\ (T_p \rightarrow A_k \in ECAW_{add}(C))$
then new test case T_p need to be added in CTd_{v2}.
(2) **If** $((prec_i$ or $postc_j \in ECAW_{add}(C)) \wedge (T_p \rightarrow vprec_i =$ " − " $\wedge\ T_p \rightarrow vpostc_j =$ " − "$))\wedge\ (T_p \rightarrow A_k \notin (ECAW_{change}(C) \cup ECAW_{add}(C) \cup ECAW_{delete}(C)))$
then T_p need to be reused in CTd_{v2}.
Rule 3 (corresponding to delete firewall):
For any i, j, k
(1)**If** $((prec_i$ or $postc_j \in ECAW_{delete}(C)) \wedge (T_p \rightarrow vprec_i =$ "T"or"F" $\vee\ T_p \rightarrow vpostc_j =$ "T"or"F"$))\vee\ (T_p \rightarrow A_k \in ECAW_{delete}(C))$
then T_p need to be deleted in CTd_{v2}.
(2) **If** $((prec_i$ or $postc_j \in ECAW_{delete}(C)) \wedge (T_p \rightarrow vprec_i =$ " − " $\wedge\ T_p \rightarrow vpostc_j =$ " − "$))\wedge\ (T_p \rightarrow A_k \notin (ECAW_{change}(C) \cup ECAW_{add}(C) \cup ECAW_{delete}(C)))$
then T_p need to be reused in CTd_{v2}.

B. Decision table-based test cases firewall at system level

At the system level, we still adopt decision table model to perform test case update. In component-based system, there is a *feature-component* table, which describes the relation between system feature and corresponding components. These features can be observed at the system level. For each feature, there is a corresponding *decision table*. Here, the proposed firewall like *component interaction graph firewall* could be used to compute the affected components at the system level.

First we need to analyze if the changes affect the corresponding components. If so, then we find out the affected system features corresponding to the affected components. According to the relation between features and decision table, the affected decision table at feature level can be identified. Then, we analyze the changes to precondition, postcondition and action of decision table, and identify the test cases firewall at system level.

According to the *System Semantic Decision Table Test Model*, we propose the following formulas to compute the test cases firewall at system level. Here, C_o denotes modified component. CIGF and ECTF represent related change impact firewalls which are introduced above. Assuming the original system test cases set is STd_{v1}, we need to obtain the updated version STd_{v2}.

$STd_{reuse} = \{STd_i | (\forall C_k (k \leq m) \in C(ft_i))(C_k \in CIGF_{reuse}(C_o) \vee ECTF_{reuse}(C_o))\}$

Where SDT_{reuse} could be reused in SDT_{v2}. C_o denotes modified component. $C(ft_i)$ denotes the set of components that support feature ft_i. SDT_i represents the decision table corresponding to feature ft_i. The number of total components is m.

$STd_{delete} = \{STd_i | (\forall C_k (k \leq m) \in C(ft_i))(C_k \in CIGF_{delete}(C_o) \vee ECTF_{reuse}(C_o))\}$

Where STd_{delete} could be deleted in STd_{v2}.

$STd_{change} = \{STd_i | (\exists C_k (k \leq m) \in C(ft_i))(C_k \in CIGF_{change}(C_o) \vee ECTF_{reuse}(C_o))\}$

Where STd_{change} need to be changed in STd_{v2}.

$STd_{new} = \{STd_i | (\forall C_k (k \leq m) \in C(ft_i))(C_k \in CIGF_{add}(C_o) \vee ECTF_{reuse}(C_o))\}$

Where STd_{new} need to be added in STd_{v2}.

VI. Case Study Report

To better understand our approach, we have performed a case study by applying the systematic regression testing from component level to system level onto a real component-based elevator system. We have used two software testing classes and two master project teams in San Jose State University (SJSU) to perform the related experiments. To make the study more typical, we conducted a complete component-based software testing process, including the original test cases design, test strategy, test coverage, etc. The decision table-based test cases are designed for both component and system. All of the test cases were executed adequately.

A. Study Objectives

The case study focuses on the following items: (a) Perform a systematic regression testing of the new component system version using the proposed approach, to verify the feasibility of the approach; (b) Check the effectiveness of the proposed approach; (c) Discover bugs after regression testing.

B. Study Subject

We have performed some case studies by applying the proposed approach in a component-based software, which is a component-based elevator system. The elevator system consists of several components, which are *car*, *user panel*, *door*, *door panel*, *userpanel queue*, *car controller*, *floor panel* and *metacontroller*. We have used two software testing classes and two master project teams in San Jose State University (SJSU) to perform the related experiments. The test cases are decision table-based. In the new version, we have made some changes such as adding a component 'Indicator' in the component 'Car' to show the current floor where the car locates, adding a component 'Indicator' in the component 'Floor Panel' to show the current floor where the car locates, adding another kind of elevator algorithm to current system, like FCFS, SCAN, etc., to make elevator scheduling configurable, and so on. The original version of the system is well designed with adequate decision table-based component test cases and system test cases. In the new system version, we conducted regression testing from component level to system level, to obtain updated decision table-based test cases, including reused test cases, deleted test cases and new test cases.

C. Study result report and discussion

Since we have many testing groups to work on the case study, we selected some good study results from four groups to report the test case reuse and bug checking. Those groups performed the experiments strictly using our approach. To perform a complete regression testing process, we classified the regression test cases into newly created test cases, reusable test cases and deleted test cases. Those groups also reported the bugs found in regression testing.

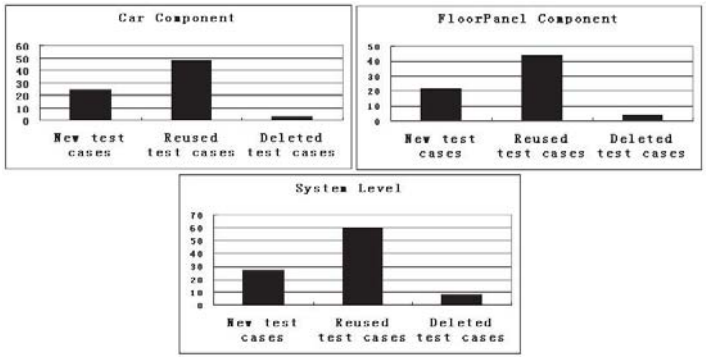


Figure 6. Regression Testing Result

Table I
Bug Report of Group 1

Level	*Test case design*	*Test case executed*	*Test coverage*	*Bugs*
Car	*79*	*79*	*100%*	*5*
Floor Panel	*59*	*59*	*100%*	*4*
System	*159*	*159*	*100%*	*11*

Figure 6 shows the regression testing results. We have obtained test cases update results from component level, i.e. *car* component and *floor panel* component to system level.

In the figure, there are three subgraphs, which represents the study results of car component, floor panel component and system respectively. The horizontal axis represents new test cases, reused test cases and deleted test cases respectively. The data of test cases are represented by a vertical bar. The height of the bar depicts the number of new, reusable, and deleted test cases.

From the figure, we can find, after regression testing, some of the original test cases could be reusable and some could be deleted. In addition, new test cases need to be created for the new version system to achieve adequate testing. For instance, in Figure 6, 25 test cases for *car* component, 23 test cases for *floor panel* component and 29 test cases for system level are newly created. This is because we have added a new component *indicator* in the elevator system. The *indicator* is added to both *car* component and *floor panel* component.

We also find most of the test cases are reusable. For instance, 48 test cases for *car* component, 44 test cases for *floor panel* component and 60 test cases for system level are reusable. The explanation is that most of the components in the system are reused in the new version. Several test cases are obsolete and deleted after regression testing due to the program changes.

Table 1 represents the bug report. The test case design includes all the new test cases and reused test cases. All of those test cases are executed with a 100% coverage. The bugs come from both component level and system level. For example, 4 bugs in *floor panel* component and 5 bugs in *car* component are found. In addition, 11 bugs are found in *system*. Thus, the modification does bring affection and impact on both the component and system.

From the result of case study, we can see the proposed approach can obtain reusable, deleted and new test cases at both levels. In addition, we also have bug reports. Hence, our approach is feasible when applied to real component-based software system.

In the case study, Most of the test cases are reusable. In terms

of the statistics of studies, nearly over 70% of the test cases are reusable. That means most of our original test cases are kept for further testing. Thus, we still can achieve a relatively high test coverage. Some test cases are deleted after regression testing, which meets the demand of reducing the number of obsolete original test cases for cost-effectiveness. Moreover, in the case study, several program bugs are reported by many groups. We also found the bugs do existed in the system after modifications. This indicates that our approach is effective.

VII. Conclusions and future work

This paper has presented a systematic regression testing technique for component-based software from component level to system level. We analyzed the whole process of regression testing, including change identification, change impact analysis and test case updating.

We proposed several re-models to support regression testing. The diverse types of changes are considered in this paper, such as component level API changes, interaction changes, and system level architecture changes. For impact analysis, the firewall concept is borrowed and extended to analyze affected program parts according to the proposed re-test models. For test models, we provide a commonly used component testing method-decision table-based testing. The change types are mapped to impact analysis, and then the affected parts are mapped to the affected test cases which are decision table-based. In addition, we performed case studies on a realistic component-based software system. The studies results show that our approach is feasible and effective.

The future extension of this research is to apply the approach into different component testing methods and models, such as state-based testing, scenario-based testing, etc. In addition, we will study how to use the approach to address automation regression test issues and develop automatic component-based regression testing tools.

References

[1] H. K. N. Leung and L. J. White. Insights into regression testing. In *Proceedings of International Conference on Software Maintenance*, pages 60–69, 1989.

[2] A. Orso et al. Using component metacontents to support the regression testing of component-based software. In *Proceedings from the ICSE Workshop in Component-based software engineering*, 2001.

[3] J. Gao et al. A systematic regression testing method and tool for software components. In *Proceedings of the 30th Annual International Computer Software and Applications Conference*, pages 455–456, 2006.

[4] Y. Wu et al. Techinques of maintaining evolving component-based software. In *IEEE International Conference on Software Maintenance (ICSM 2000)*, 2000.

[5] C. Y. Mao. Regression testing for component-based software via built-in test design. In *ACM Symposium on Applied Computing*, 2007.

[6] A. Orso et al. Scaling regression testing to large software systems. In *Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pages 241–251, 2004.

[7] J. Bible et al. A comparative study of coarse- and fine- grained safe regression test-selection techniques. *ACM Transactions on Software Engineering and Methodology*, 10(2):149–183, 2001.

[9] G. Rothermel and M. J. Harrold. Analyzing regression test selection techniques. *IEEE Transactions on Software Engieering*, 22(8):529–551, 1996.

[10] G. Rothermel and M. J. Harrold. A safe, efficient regression test selection technique. *ACM Transactions on Software Engineering and Methodology*, 6(2):173–210, 1997.

[11] G. Rothermel et al. The impact of test suite granularity on the cost-effectiveness of regression testing. In *Proceedings of International Conference on Software Engineering*, pages 19–25, 2002.

[12] C. Q. Tao, B. X. Li, and X. B. Sun. A hierarchical model for regression test selection and cost analysis of java programs. In *To appear in the proceedings of 2010 Asia Pacific Software Engineering Conference (APSEC2010)*, 2010.

[13] L. White and H. K. N. Leung. A firewall concept for both control-flow and data-flow in regression integration testing. In *Proceedings of the IEEE International Conference on Software Maintenance*, pages 262–271, 1992.

[14] D. Kung and Jerry Gao. Change impact identification in object-oriented software maintenance. In *Proceedings of the IEEE International Conference on Software Maintenance*, pages 202–211, 1994.

[15] D. Kung and Jerry Gao. On regression testing of object-oriented programs. *Journal of Systems and Software*, 32(1):21–40, 1996.

[16] M. J. Harrold et al. Using component metacontents to support the regression testing of component-based software. In *IEEE International Conference on Software Maintenance*, pages 716–725, 2001.

[17] J. Zheng et al. Applying regression test selection for cots-based applications. In *Proceedings of International Conference on Software Engineering*, pages 512–522, 2006.

[18] S. Papagiannakis L. Mariani and M. Pezze. Compatibility and regression testing of cots-component-based software. In *The 29th International Conference on Software Engineering*, 2007.

[19] B. Robinson and L. White. Testing of user-configurable software systems using firewalls. In *International Symposium on Software Reliability Engineering*, pages 177–186, 2008.

[20] J. Gao et al. Model-based test complexity for software installation testing. In *International Conference on Software Engineering and Knowledge Engineering (SEKE08)*, 2008.

[21] Y. Wu et al. Techniques for testing component-based software. In *The 7th International Conference on Engineering of Complex Computer Systems*, pages 222–232, 2001.

[22] Jerry Gao et al. Testing and quality assurance for component software. *Massachusetts: Artech House, Inc*, 2003.

[23] Lee Copeland. A practitioner's guide to software test design. *Massachusetts: Artech House, Inc*, 2004.

Multiple Fault Localization with Data Mining

Peggy Cellier and Mireille Ducassé
IRISA/INSA of Rennes
Campus Beaulieu 35042 Rennes cedex, France
Email: firstname.lastname@irisa.fr

Sébastien Ferré and Olivier Ridoux
IRISA/University of Rennes 1
Campus Beaulieu 35042 Rennes cedex, France
Email: firstname.lastname@irisa.fr

***Abstract*—We propose an interactive fault localization method based on two data mining techniques, formal concept analysis and association rules. A lattice formalizes the partial ordering and the dependencies between the sets of program elements (e.g., lines) that are most likely to lead to program execution failures. The paper provides an algorithm to traverse that lattice starting from the most suspect places. The main contribution is that the algorithm is able to deal with any number of faults within a single execution of a test suite. In addition, a stopping criterion independent of the number of faults is provided.**

I. Introduction

Executing a test suite can be very costly, especially when the test oracle, that tells whether a run is correct, is not fully automated. In the multiple fault localization context, being able to explain as many as possible of the failures detected by the test suite execution is therefore an important property. However, it is in general impossible to tell how many faults are present in a buggy program. In order to deal with multiple faults, Liblit *et al.* [11] propose a ranking based on a statistic treatment of inserted predicates. Zheng *et al.* [16] propose a method based on bi-clustering in order to group failed executions and to identify one feature that characterizes each cluster. Jones *et al.* [9], with their *parallel debugging*, propose to cluster failed executions in order to distribute each cluster of executions to different debuggers. Ideally, one cluster should represent one fault, and instead of searching the faults one by one, what the authors call *sequential debugging*, the faults can be searched in parallel by different persons. In all those methods, there is, however, no guaranty that each cluster represents only one fault. Furthermore, the dimension used to measure suspicion tends to ignore the actual structure of the program. On the one hand, lines with close suspicion scores may be uncorrelated; on the other hand, correlated lines may have very different suspicion scores (e.g, a condition and the branches it controls).

In a previous work [4], we have proposed a new data-structure that can organize program elements in a multi-dimensional space: the *failure lattice*. The failure lattice is a partial ordering of the elements of the traces that are most likely to lead to failures. The traces can contain different kinds of information called *events*, for instance, executed lines or variable values. We assume that each execution trace contains, in addition to the events, the verdict of the execution, $PASS$ if the execution produces the expected results, and $FAIL$ otherwise. In a previous work [5], we have compared the exploration of the failure lattice with existing approaches that handle only a single fault. We have shown, in particular, that the method gives a comparable number of lines to analyze while providing a richer environment for the analysis.

```
public int Trityp(){
[57] int trityp ;
[58] if ((i==0) || (j==0) ||
        (k == 0))
[59]   trityp = 4 ;
[60] else
[61]  {
[62]    trityp = 0 ;
[63]    if ( i == j)
[64]      trityp = trityp + 1 ;
[65]    if ( i == k)
[66]      trityp = trityp + 2 ;
[67]    if ( j == k )
[68]      trityp = trityp + 3 ;
[69]    if (trityp == 0)
[70]    {
[71]      if ((i+j <= k) ||
              (j+k <= i) ||
              (i+k <= j))
[72]          trityp = 4 ;
[73]      else
[74]          trityp = 1 ;
[75]    }
[76]    else
[77]    {
[78]      if (trityp > 3)
[79]          trityp = 3 ;
[80]      else
[81]          if ((trityp == 1)
                   && (i+j > k))
[82]            trityp = 2 ;
[83]          else
[84]            if ((trityp == 2)
                     && (i+k > j))
[85]              trityp = 2 ;
[86]            else
[87]              if((trityp == 3)
                     && (j+k > i))
[88]                trityp  = 2 ;
[89]              else
[90]                trityp = 4 ;
[91]    }
[92]   }
[93]   return(trityp) ;}
static public
string conversiontrityp(int i){
[97]  switch (i){
[98]     case 1:
[99]         return "scalen";
[100]    case 2:
[101]        return "isosceles";
[102]    case 3:
[103]        return "equilateral";
[104]    default:
[105]        return "not a ";}}
```

Fig. 1. Trityp Java program.

In this paper, we propose an interactive fault localization algorithm that explores the *failure lattice* and can deal with multiple faults. The contribution of this paper is twofold. Firstly, the exploration stops when all the detected failures are explained. That stopping criterion is independent of the number of faults. Secondly, a programmer can exploit all the results of a single execution of a test suite, thus reducing the number of executions of the test suite.

In the remaining of the paper, Section II presents the running example. Section III presents the failure lattice introduced in [4]. Section IV introduces an extended labelling of the failure lattice for the multiple fault problem. Section V defines the proposed algorithm to explore the data structure and locate multiple faults in a program. Section VI and Section VII discuss the characteristics of the algorithm with respect to fault dependencies and statistical indicators. Finally Section VIII presents the related work.

II. Running Example

To illustrate our method, we use a classical benchmark for test generation methods, the Trityp Java program shown in Figure 1. It classifies sets of three segment lengths into four categories: *not a triangle*, *scalene*, *isosceles*, *equilateral*. The program contains one class with 130 lines of code. The

Fault Id	Faulty line
1	`[84] if ((trityp == 3) && (i+k > j))`
2	`[79] trityp = 0 ;`
3	`[74] trityp = 0 ;`
4	`[90] trityp == 3 ;`
5	`[66] trityp = trityp+20 ;`
6	`[64] trityp = i+1 ;`

TABLE I
FAULTS FOR THE TRITYP PROGRAM.

	Executed lines				Verdict	
	line 57	line 58	...	line 105	PASS	FAIL
Execution 1	×	×		×	×	
Execution 2	×	×		×		×
...						

TABLE II
CONTEXT OF FAULT LOCALIZATION.

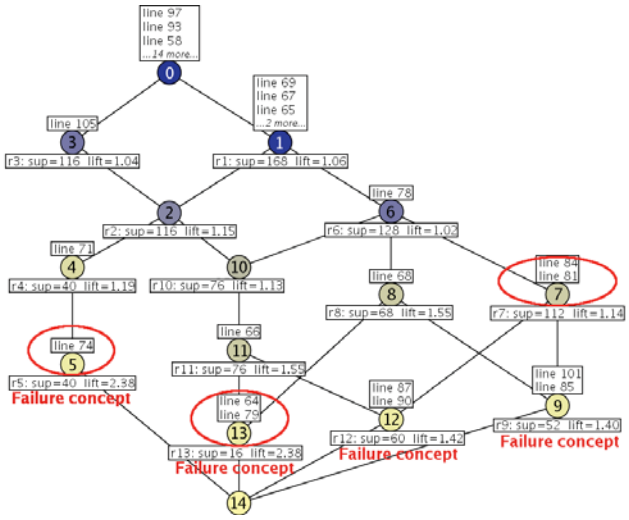


Fig. 2. Failure lattice for program Trityp with faults 1, 2 and 3.

faulty programs used in this article are a courtesy of Petit and Gotlieb [14] and they can be found on the web[1]. Table I presents six faults for the Trityp program that are used in different blends in the following.

III. DATA MINING AND FAILURE LATTICE

In this section, we give background knowledge about two data mining techniques in order to explain how to read a failure lattice. The construction is explained in [4].

Data mining is a process to extract relevant information from a huge amount of data. Two data mining techniques are used to compute the failure lattice: Association Rules and Formal Concept Analysis (FCA). The input of those techniques is a *formal context*, i.e. a binary relation describing elements of a set of *objects* by subsets of a set of *attributes*. Table II is an example of context. The objects are the executions. The attributes are the program lines and the verdicts. Each execution is described by its execution trace.

1) Association Rules: Searching for association rules [1] allows interesting regularities to be found. An *association rule* has the form: $P \rightarrow C$, where P and C are sets of attributes. P is called the *premise* of the rule and C the *conclusion*. Any pair of premise and conclusion forms an association rule, but some of them are more relevant than others. In order to measure the relevance of computed rules, statistical indicators are used. In the following, we use two indicators, *support* and *lift*, that will be defined when they are actually needed.

2) Formal Concept Analysis (FCA): Formal Concept Analysis [7] allows relevant clusters to be computed. In FCA, the set of all objects that share a set of attributes is called the *extent* of the set of attributes. For example, in the context of Table II, $extent(\{line\ 57, line\ 58\})$ is the set of executions that execute $line\ 57$ and $line\ 58$ and $extent(\{FAIL\})$ denotes all failed executions. The set of all attributes shared by all elements of a set of objects is called the *intent* of the set of objects. For instance, the intent of a set of executions, S, is the set of lines that appear in the traces of all executions of S and the verdict if it is the same for all executions of S. In FCA, a *formal concept* is defined as a pair (O, A), where the set of objects, O, is the extent of the set of attributes, A, and A is the intent of O. The set of concepts of a context can be represented by a concept lattice where each attribute and each object labels only one concept[2]. Namely, each concept is labelled by the attributes and the objects that are characteristic to its intent and extent.

The failure lattice is computed thanks to combining association rules and formal concept analysis. The set of execution traces is a context where the objects are the executions and where each execution is described by the events occuring during the execution. From that context, association rules are computed: $e_i,\ e_j, \cdots \rightarrow FAIL$. Those rules can be read as "when events $e_i,\ e_j, \cdots$ appear in a trace most of the time the execution fails". Those rules, extracted from the execution trace context, form the context used to compute the failure lattice. In that second context, the objects are the association rules and each rule is described by the events that appear in its premise. Some rules are more specific than others and that partial order is highlighted in the failure lattice. Figure 2 displays an example of failure lattice for the faulty Trityp program which is obtained by combining faults 1, 2 and 3. Note that, in the sequel, for the examples, the events in execution traces are the executed lines and verdicts.

The premise of a rule r, in the failure lattice, can be obtained by collecting the attributes labelling all the concepts above the concept that is labelled by r. In our particular case, there is only one rule with a given premise, so a concept can only be labelled by one rule in the failure lattice. We will therefore give the same number to a concept and the rule that labels it. In Figure 2, r_3 is $line\ 105, line\ 97, line\ 93, line\ 58, ... \rightarrow FAIL$. Two pieces of information are added to the rule: support and

[1]http://www.irisa.fr/lande/gotlieb/resources/Java_exp/trityp/

[2]There are some tools that automatically compute concept lattices and their labelling. For instance, ToscanaJ (http://toscanaj.sourceforge.net/) is used for the lattices that appear in this paper.

lift values. The support of a rule is the number of failed executions that have in their trace all events of the premise of the rule. The support of r_3 means that 116 failed executions execute lines 105, 97, 93, 58, ... The lift of a rule indicates if the occurrence of the premise in a trace increases the probability to fail. A lift value greater than 1 means that the observation of the premise in a trace increases the probability to fail.

In Figure 2, we see that r_2 is more specific than r_3, indeed its premise contains the premise of r_3. It is denoted by $r_2 < r_3$. Therefore, c_3 is a superconcept of c_2.

Note that in the illustrations, the red ellipses point to the concepts labelled by faulty lines. Note also that we illustrate the method with pictures of concept lattices, but it is only for the sake of exposing the logic of the process. In fact, failure lattices are in general very large and are not legible. They however need not to be computed in totality, neither need to be globally exposed to the user.

IV. The Failure Lattice for Multiple Faults

This section presents how the labelling of the lattice is extended by failure concepts and support clusters. Properties for the multiple fault problem are given.

A concept c is a *failure concept* of the failure lattice if there exists some failed execution that contains all events of the intent of c in their trace but not all the events of the intent of subconcepts of c. For instance in Figure 2, concept 5 is a failure concept, indeed it covers 40 failed executions that are not covered by a more specific concept. We also say that those 40 failed executions are *covered* by rule 5. Concept 4 is not a failure concept because the 40 failed executions that it covers are already covered by concept 5 which is a more specific concept. In the same way, concept 2 is not a failure concept because among the 116 failed executions that are covered by it, 40 are already covered by concepts 4 and 5 and the remaining 76 are covered by concept 10, which are all more specific concepts than concept 2. In the example, there are four failure concepts: 5, 13, 12 and 9. We note $FAILURES$ the set of failure concepts.

We call *support cluster* a maximal set of connected concepts labelled by rules which have the same support value. We note $cluster(c)$ the support cluster that contains c. For instance, in Figure 2, concepts 4 and 5 belong to the same support cluster. It means that lines 71 and 74 are executed by exactly the same 40 failed executions. Note that there is at most one failure concept by support cluster.

The failure lattice has two properties which are essential for the design of the algorithm of next section.

*Property 1 (**global monotony of support**):* Let c_i and c_j be two concepts of a failure lattice. If $c_j < c_i$ then $sup(r_j) \leq sup(r_i)$. For instance, in Figure 2, concepts 11 and 13 are ordered, $c_{13} < c_{11}$ and $sup(r_{13}) = 16 < sup(r_{11}) = 76$.

Proof: The fact $c_j < c_i$ implies that the intent of c_j strictly contains the intent of c_i. The intent of concept c_i (resp. c_j) is the premise, p_i (resp. p_j), of rule r_i (resp. r_j), thus $p_i \subset p_j$. Conversely $extent(p_j) \subset extent(p_i)$. Thanks to the definition of the support $sup(r_j) \leq sup(r_i)$ holds. □

*Property 2 (**local monotony of lift**):* Let c_i and c_j be two concepts of a failure lattice. If $c_j < c_i$ and $sup(r_j) = sup(r_i)$ then $lift(r_j) > lift(r_i)$. For instance, concept 4 and 5 are ordered, $c_5 < c_4$, $sup(r_5) = sup(r_4) = 40$ and $lift(r_5) = 2.38 > lift(r_4) = 1.19$. It means that r_5 and r_4 cover the same failed executions but r_5 covers less passed executions than r_4.

Proof: In the previous proof we have seen that $c_j < c_i$ implies that $extent(p_j) \subset extent(p_i)$. The definition of the lift of a rule $r = p \rightarrow FAIL$ is $lift(r) = \frac{sup(r)}{\|extent(p)\|\|extent(FAIL)\|} * \|\mathcal{O}\|$. As $sup(r_j) = sup(r_i)$, and $\|extent(p_j)\| < \|extent(p_i)\|$, $lift(r_j) > lift(r_i)$ holds. □

V. Exploration of the Failure Lattice for Multiple Faults

In this section, we propose an algorithm to explore the failure lattice in order to find clues to understand faults. The algorithm presents events to a *debugging oracle*, currently most likely a human person. We assume *the competent debugger hypothesis*, namely when a set of events that indicates a fault is presented to the debugging oracle he will detect the fault. The same kind of hypothesis is assumed in [2].

When locating faults in a program, relevant parts of the program are the ones specific to failed executions. In the failure lattice, that information belongs to the most specific rules which label concepts at the bottom of the failure lattice. Furthermore, as stated by Property 1, the bottom concepts are the ones with the lowest support and, as stated by Property 2, the bottom concepts inside a support cluster are the ones with the highest lift. As a consequence, the rule lattice is explored bottom-up, starting from the more specific rules with the lowest support and highest lift.

We define the *fault context* of a concept c as the set of events that label strict subconcepts of c. For example, in Figure 2 the fault context of concept 11 is lines 64, 79, 87 and 90. For the localization task, as the lattice is traversed bottom-up, when exploring concept c, the events of its fault context have already been explored. Each time a concept c is explored, the debugging oracle has to say if the lines that label c, in addition to the fault context of c, indicates a fault. In that case we say that the failed executions that label c (and thus subconcepts of c) are *explained* by that fault. By definition, the concepts belonging to the support cluster of c are also *explained*, indeed, the concepts of a same support cluster cover exactly the same failed executions.

The exploration of the failure lattice stops when all failure concepts in $FAILURES$ are explained by at least one fault. Namely, when all failed executions that are covered by a concept in the failure lattice are *explained*.

The exploration strategy is specified by Algorithm 1. The failure lattice is traversed bottom-up, starting with the failure

Algorithm 1 Failure lattice traversal

1: $C_{next} := FAILURES$
2: $C_{failure} := FAILURES$
3: **while** $C_{failure} \neq \emptyset \wedge C_{next} \neq \emptyset$ **do**
4: **let** $c \in C_{next}$ **in**
5: $C_{next} := C_{next} \setminus \{c\}$
6: **if** the debugging_oracle locates no fault in the label of c given the fault context of c **then**
7: $C_{next} := C_{next} \cup \{$upper neighbours of $c\}$
8: **else**
9: **let** $Coverage$ = subconcepts(c) $\cup$ cluster(c) **in**
10: $C_{next} := C_{next} \setminus Coverage$
11: $C_{failure} := C_{failure} \setminus Coverage$
12: **end if**
13: **end while**

Iteration	C_{next}	$C_{failure}$
0	$\{c_5, c_{13}, c_{12}, c_9\}$	$\{c_5, c_{13}, c_{12}, c_9\}$
1	$\{c_{13}, c_{12}, c_9\}$	$\{c_{13}, c_{12}, c_9\}$
2	$\{c_{12}, c_9\}$	$\{c_{12}, c_9\}$
3	$\{c_9, c_7, c_{11}\}$	$\{c_{12}, c_9\}$
4	$\{c_7, c_{11}, c_8\}$	$\{c_{12}, c_9\}$
5	$\{c_{11}, c_8\}$	$\{\}$

TABLE III
EXPLORATION OF THE FAILURE LATTICE OF FIG. 2.

concepts (step 1). At the end of the failure lattice traversal, $C_{failure}$ - the set of failure concepts not *explained* by a fault (step 2) - is empty or all concepts have been already explored (step 3). When a concept, c (step 4), is chosen among the concepts to explore, C_{next}, the events that label the concept, are explored. Note that the selection of that concept is not deterministic. If no fault is located, then the upper neighbours of c are added to the set of concepts to explore (step 7). If, thanks to those new clues, the debugging oracle understands mistakes and locates one or several faults then all subconcepts of c and all concepts that are in the same support cluster are *explained*. Those concepts do not have to be explored again (step 10). The failure concepts that are subconcepts of c are *explained* (step 11).

At each iteration, $C_{failure}$ can only decrease or remain untouched. The competent debugger hypothesis makes sure that $C_{failure}$ ends at empty when min_sup is equal to 1.

For the example of Figure 2, the support threshold, min_sup, is equal to 4 failed executions (out of 400 executions, of which 168 failed executions) and the lift threshold, min_lift, is equal to 1. There are four failure concepts: 5, 13, 12 and 9. Table III presents the values of C_{next} and $C_{failure}$ at each iteration of the exploration. We choose to explore the lattice with a queue strategy, namely first in C_{next}, first out of C_{next}. Note that the stopping criterion of the algorithm depends only of the failure concepts, it is therefore valid whatever the chosen strategy.

At the begining, C_{next} and $C_{failure}$ are initialized as the

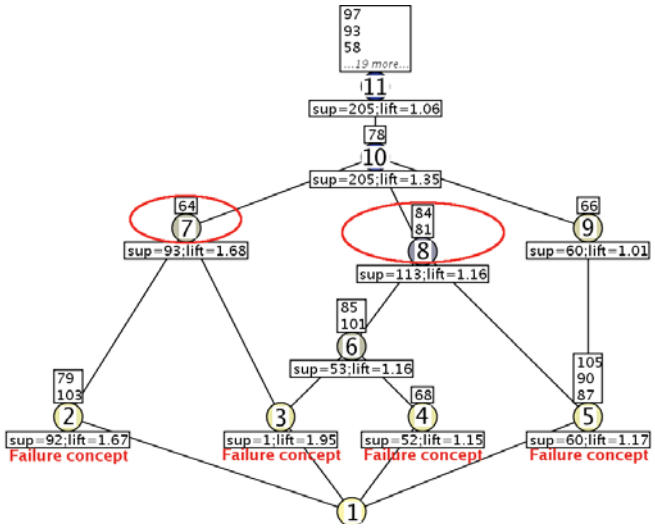


Fig. 3. Failure lattice for program Trityp with faults 1 and 6

set of all failure concepts (Iteration 0 in Table III). At the first iteration of the while loop, concept 5 is selected ($c = c_5$). That concept is labelled by line 74. Line 74 actually corresponds to fault 3. The debugger oracle locates fault 3. Concept 5, 4 and 14 are thus tagged as *explained*. The new values of C_{next} and $C_{failure}$ are presented at iteration 1 in Table III. At the second iteration, concept 13 is selected ($c = c_{13}$). That concept is labelled by lines 64 and 79. Line 79 actually corresponds to fault 2; the debugging oracle locates fault 2. Concept 13 is tagged as *explained*. At the third iteration, concept 12 is selected. That concept is labelled by lines 87 and 90. No fault is found. The upper neighbours, concepts 7 and 11, are added to C_{next} and $C_{failure}$ is unchanged. At the next iteration, concept 9 is selected. As in the previous iteration no fault is found. The upper neighbour, concept 8, is added to C_{next}.

Finally, concept 7 is selected. That concept is labelled by lines 81 and 84. By exploring those lines (new clues) in addition with the *fault context*, i.e. lines that have already been explored: 87, 90, 101 and 85, the debugging oracle locates fault 1 at line 84. The fault is the substitution of the test of trityp equal to 2 by a test of trityp equal to 3. Concepts 12 and 9 exhibit two concrete realisations (failures) of the fault at line 84 (Concept 7). Concepts 7, 12, 9 are tagged as *explained*. The set of failure concepts to explain is empty, thus the exploration stops. All four failures are explained after the debugging oracle has inspected nine lines.

VI. FAULT DEPENDENCIES

This section discusses the behavior of the algorithm with respect to fault dependencies. In the following, a fault is identified by its faulty line. We call *fault concept* of a fault F the most specific concept, c_F, such that it is labelled by the faulty line of F.

When two faults are independent, the execution of one fault implies that the other fault cannot be executed. The fault concepts of both faults thus appear in different support

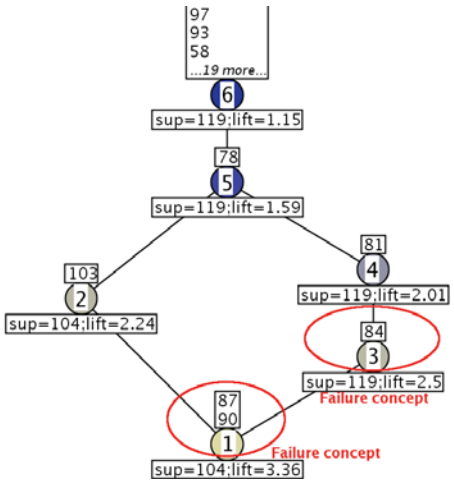


Fig. 4. Failure lattice for program Trityp with faults 1 and 4

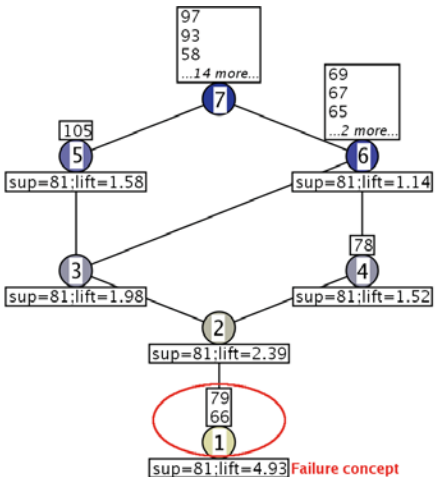


Fig. 5. Failure lattice for program Trityp with faults 2 and 5

clusters. That case is illustrated in Figure 2 which shows an example of three independent faults. The previous section has shown how the algorithm helps a debugging oracle locate the faults.

Two faults can be partially dependent, it means that some failed executions execute both faults. For example, Figure 3 presents the failure lattice for the Trityp program with faults 1 and 5. Fault 1 is at line 84, fault 6 is at line 64. The failure lattice contains four failure concepts: 2, 3, 4 and 5. There are also 8 support clusters. Running Algorithm 1, the exploration of concepts 2 and 3 does not allow a fault to be located, but gives clues when exploring concept 7. Concept 7 is labelled by line 64. Fault 6 is located. In the same way, exploring concepts 3, 4, 5 and 6 is not sufficient to locate a fault, but gives clues when exploring concept 8. Concept 8 is labelled by lines 84 and 81. Fault 1 is located. The two faults are not independent but they are represented separately, and can be distinguished.

Another example of partially dependent faults is given in Figure 4 that presents the failure lattice for the Trityp program with faults 1 and 4. Fault 1 is at line 84 and fault 4 is at line 90. Executing faulty line 90 implies to have executed faulty line 84 before, but executing line 84 does not always imply executing line 90. Fault 4 strongly depends on fault 1. The failure lattice contains two failure concepts, concepts 1 and 3. There are two support clusters. One support cluster has its support value equal to 104 failed executions. The second support cluster has its support value equal to 119 failed executions (all executions). Running Algorithm 1, concept 1 is explored. It is labelled by two lines: 87 and 90. Line 90 is fault 4. Concept 3 is labelled by line 84 then fault 1 is located. All failure concepts are tagged as *explained*. The search is finished, the two faults are located. The two faults are dependent but they are represented in different support cluster, and can be distinguished.

The last case is when the faults are always executed together in failed executions. From the point of view of the failed executions, they cannot be distinguished. Figure 5 presents the failure context for the Trityp program with faults 2 and 5. Fault 5, at line 66, sets triptyp to a value above 20, trityp is tested at line 78, with the value set at line line 66 the execution necessarily goes to line 79 which contains fault 2. The failure lattice contains only one *failure concept*, concept 1. All concepts belong to the same support cluster. The support value of that cluster is equal to 81, which is the number of failed executions. Running Algorithm 1, concept 1 is explored. It is labelled by two lines: 79 and 66. No distinction is done between both faults in the failure lattice. They are seen as a unique fault. Therefore, if the debugging oracle stops diagnosing when he finds one fault without exploring all the lines presented at that step, only one fault is found. Note, that the lines related to the other fault, are nevertheless present at that step and that the oracle can still find both faults in this example. However, in the worst case, when faults are executed by the same failed executions but by some different passed executions, the fault concepts are two different concepts in the same support cluster. It implies that when one of the faults is found, the search stops. All concepts in the support cluster are *explained*. In that case the first found fault hides the second one and the program has to be executed again after the correction of the first fault.

VII. Statistical Indicators

Computing association rules w.r.t. a support threshold (min_sup) filters out some rules. For example, rules that stress very specific lines may have a support under the threshold because they are present in a small number of failed traces. Those rules do not appear in the failure lattice. The failed executions that are covered by only pruned rules may not label concept in the failure lattice. The proposed process helps drawing conclusions from failures that label concepts in the failure lattice. A run of the fault localization process draws every conclusion from the failures that have a sufficient support. A new run can safely considers only the low support failures. This suggests a progressive approach for fault localization as successive runs of rule exploration with decreasing supports.

When the program contains a single fault, a rule is relevant with a lift value greater than 1. When the program contains several faults the lift threshold can be lower. Indeed, when a program contains several faults, verdict $FAIL$ is used as an abstraction of several more specific verdicts. All failures are not caused by the same faults, nevertheless we consider only one attribute, $FAIL$, to characterize failed executions when searching for association rules ($? \rightarrow FAIL$). If the failures were tagged by the faults that cause them, for example $FAIL_{F_i}$, the searched association rules would be $? \rightarrow FAIL_{F_i}$. However, we cannot assume that failures are tagged. When a rule $F_i \rightarrow FAIL_{F_i}$ would have a lift greater or equal to 1, the rule $F_i \rightarrow FAIL$ can have a lift lower than 1. The lift threshold can thus start equal to 1 at the beginning of the localization process and decrease below 1 if some faults remain hidden.

Note that, in both the above cases, the test suite does not need to be re-executed.

VIII. Related Work

In [5], we have compared our data structure, the failure lattice, with existing fault localization methods. In this section, we briefly recall the main results and further our navigation into the failure lattice, with the strategies of other methods. Renieris and Reiss [15] as well as Cleve and Zeller [6] have proposed four methods based on the differences between executions traces. In those cases, navigating is the action to explore the whole set of events without guide whereas our approach is guided by the structure of the failure lattice. The statistical methods, Tarantula [10], parallel debugging [9], Falcon [13], SBI [11], the bi-clustering method [16] and SOBER [12] rank events and there is not necessarly a relation between an event and the following one in the ranking. On the contrary, our proposed approach gives a context to understand the faults thanks to a partial ordering of the events, which takes into account their dependencies. In [3], the authors takes into account the program dependence graph in their scoring function thanks to the causal-inference theory, but as previously seen with the other methods, no context is provided with each statement.

For multiple faults, Jiang *et al.* [8] propose a method based on traces whose events are predicates. The predicates are clustered, and the path in the control flow graph associated to each cluster is computed. In the failure lattice, events are also clustered in concepts. The relations between concepts give information about the path in the control flow graph and highlight some parts of that path as relevant for debugging without computing the control flow graph. Other methods group together failed executions to separate the effects of different faults ([16], [9]). Those methods do not take into account the dependencies between faults whereas our method does. Finally, SBI [11] has a stopping criterion. SBI wants to take advantage of one execution of the test suite. SBI ranks predicates. When a fault is found thanks to the ranking, all execution traces that contain the predicates used to find the fault are deleted and a new ranking on predicates with the reduced set of execution traces is computed. Deleting execution traces can be seen as equivalent to tagging concepts, and thus the events of their labelling. The difference between SBI and our approach is that our approach does not need to compute the failure lattice several times.

IX. Conclusion

In this paper, an algorithm to locate multiple faults in programs is proposed. It is based on a data structure, the failure lattice, that gives a partial ordering of the events of the traces. At each step of the proposed algorithm, a debugging oracle, for example the human debugger, has to say if a fault is found given a set of events. The advantage of our algorithm is that the debugging oracle knows when to stop the exploration of the program.

References

[1] R. Agrawal, T. Imielinski, and A. N. Swami. Mining association rules between sets of items in large databases. In *Int. Conf. on Management of Data*. ACM Press, 1993.

[2] S. Ali, J.H. Andrews, T. Dhandapani, and W. Wang. Evaluating the accuracy of fault localization techniques. In *Proc. of the IEEE/ACM Int. Conf. on Automated Software Engineering*, pages 76–87. IEEE Computer Society, 2009.

[3] G.K. Baah, A. Podgurski, and M.J. Harrold. Causal inference for statistical fault localization. In *Proc. of the Int. Symp. on Software testing and analysis*, pages 73–84. ACM, 2010.

[4] P. Cellier, M. Ducassé, S. Ferré, and O. Ridoux. Formal concept analysis enhances fault localization in software. In *Formal Concept Analysis*, volume 4933. Springer-Verlag, 2008.

[5] P. Cellier, M. Ducassé, S. Ferré, and O. Ridoux. Dellis: A data mining process for fault localization. In *Software Engineering & Knowledge Engineering*, 2009.

[6] H. Cleve and A. Zeller. Locating causes of program failures. In *Int. Conf. on Software Engineering*. ACM Press, 2005.

[7] B. Ganter and R. Wille. *Formal Concept Analysis: Mathematical Foundations*. Springer-Verlag, 1999.

[8] L. Jiang and Z. Su. Context-aware statistical debugging: from bug predictors to faulty control flow paths. In *Proc. of the Int. Conf. on Automated Software Engineering*, pages 184–193. ACM Press, 2007.

[9] J. A. Jones, J.F. Bowring, and M.J. Harrold. Debugging in parallel. In *Int. Symp. on Software Testing and Analysis*, pages 16–26, July 2007.

[10] J. A. Jones, M.J. Harrold, and J. T. Stasko. Visualization of test information to assist fault localization. In *Int. Conf. on Software Engineering*, pages 467–477. ACM, 2002.

[11] B. Liblit, M. Naik, A. X. Zheng, A. Aiken, and M. I. Jordan. Scalable statistical bug isolation. In *Conf. on Programming Language Design and Implementation*. ACM Press, 2005.

[12] C. Liu, L. Fei, X. Yan, J. Han, and S.P. Midkiff. Statistical debugging: A hypothesis testing-based approach. *IEEE Transaction Software Engineering.*, 32(10):831–848, 2006.

[13] S. Park, R.W. Vuduc, and M.J. Harrold. Falcon: fault localization in concurrent programs. In *Proc. of the ACM/IEEE Int. Conf. on Software Engineering*, pages 245–254. ACM, 2010.

[14] M. Petit and A. Gotlieb. Uniform selection of feasible paths as a stochastic constraint problem. In *Int. Conf. on Quality Software*, IEEE, October 2007.

[15] M. Renieris and S. P. Reiss. Fault localization with nearest neighbor queries. In *Int. Conf. on Software Engineering*. IEEE, 2003.

[16] A. X. Zheng, M. I. Jordan, B. Liblit, M. Naik, and A. Aiken. Statistical debugging: simultaneous identification of multiple bugs. In *Int. Conf. on Machine Learning*, 2006.

Constructing Subtle Concurrency Bugs Using Synchronization-Centric Second-Order Mutation Operators

Leon Wu Gail Kaiser
Department of Computer Science
Columbia University
New York, NY 10027 USA
{leon,kaiser}@cs.columbia.edu

Abstract

Mutation testing applies mutation operators to modify program source code or byte code in small ways, and then runs these modified programs (i.e., mutants) against a test suite in order to evaluate the quality of the test suite. In this paper, we first describe a general fault model for concurrent programs and some limitations of previously developed sets of first-order concurrency mutation operators. We then present our new mutation testing approach, which employs synchronization-centric second-order mutation operators that are able to generate subtle concurrency bugs not represented by the first-order mutation. These operators are used in addition to the synchronization-centric first-order mutation operators to form a small set of effective concurrency mutation operators for mutant generation. Our empirical study shows that our set of operators is effective in mutant generation with limited cost and demonstrates that this new approach is easy to implement.

1 Introduction

Mutation testing is a white-box fault-based software testing technique that uses mutants, slightly modified variants of the program source code or byte code, to characterize the effectiveness of a testing suite and locate weaknesses in the test data or program that are seldom or never exposed during normal execution [9]. Mutation testing is based on the *Competent Programmer Hypothesis* and the *Coupling Effect Hypothesis*. The *Competent Programmer Hypothesis* assumes that programmers are competent and normally write programs that are close to perfect; program faults are syntactically small and can be corrected with a few small code modifications [1, 9]. The *Coupling Effect Hypothesis* states that complex bugs in software are closely coupled to small, simple bugs. Thus, mutation testing can be effective in simulating complex real-world bugs [9, 23].

Mutation testing typically involves three stages: (1) Mutant generation, the goal of which is the generation of mutants of the program through inserting bugs. (2) Mutant execution, the execution of test cases against both the original program and the mutants. (3) Result analysis, to check the *mutation score*, *i.e.*, the percentage of nonequivalent mutants that are *killed* by the test suite [26, 23]. A mutant is *equivalent* to the original program if the mutant and the original program always produce the same output, hence no test case can distinguish between the two [8]. A mutant is considered *killed* by the test suite if the execution result of the mutant is different from the result of the original program [25]. A test data set is said to be *adequate* if its mutation score is 100% [8, 24].

For the first stage, a predefined set of mutation operators are used to generate mutants from program source code or byte code. A *mutation operator* is a rule that is applied to a program to create mutants [25]. Mutants containing one simple fault are called *first-order mutants* and mutants containing two simple faults are called *second-order mutants* [26]. Researchers have developed many sets of mutation operators [25, 17], targeting a variety of programming languages. For example, Delamaro *et al.* and Bradbury *et al.* have proposed different set of mutation operators for concurrent Java programs [7, 4]. Our empirical study and analysis shows that some subtle concurrency bugs are not generated by any of these proposed first-order mutation operators. Our study further shows that a large portion of these operators are not effective in mutant generation: the majority of the mutants are generated by a small subset of the mutation operators, generally those that are synchronization-centric, *i.e.*, directly relating to the synchronization of different processes or threads. Based on a general fault model for concurrent programs and our analysis of the limitations in prior work, we present our new mutation testing approach, which employs synchronization-centric second-order mutation operators that are able to generate subtle concurrency bugs not

represented by the first-order mutation. These operators are used in addition to the synchronization-centric first-order mutation operators to form a small set of effective concurrency mutation operators that can be used in mutant generation. Our empirical study shows that our small set of operators is effective in mutant generation with limited cost and demonstrates that this new approach is easy to implement. The initial analysis of the possible implications of our results has potential impact on the Coupling Effect Hypothesis, indicating that possibly the coupling effect is weaker in concurrent programs than in sequential programs.

The remainder of this paper is structured as follows. In Section 2, we describe our fault model for concurrent programs. In Section 3, we present the limitations of some previous work. In Section 4, we present our new approach. In Section 5, we present our empirical study. Lastly, we discuss the related work before we conclude.

2 Fault Model for Concurrent Programs

Testing concurrent programs is difficult. It is generally impossible or impractical to exhaustively test all combinations of input values or cover all possible control or data flow paths in sequential programs but even more so in concurrent programs; nevertheless, test suites can and must be constructed according to various criteria to attempt to find bugs. In order to develop a set of concurrency mutation operators that are able to model subtle concurrency bugs, we employ a general fault model that is based on the concurrency bug patterns and the synchronization mechanisms. Our definition of *fault* is a programming error that leads to an erroneous result in some programs during execution.

2.1 Concurrency Bug Patterns

Some prior research on concurrency bug patterns has been done. [13] and [20] described taxonomy of common concurrency bugs. [12] compiled a benchmark of concurrency bugs. [6] and [21] described some empirical studies on concurrency bugs. We consolidate the common concurrency faults from these prior research that we consider for mutation operators to model and present them below.

- Data Race: *Data race* condition happens when multiple threads read and write the same data, and the outcome of the execution depends on the particular order in which the accesses happen [6]. It is also called *thread interference*.
- Memory Inconsistency: *Memory inconsistency* errors occur when different threads have inconsistent views of the same variable.
- Atomicity Violation: *Atomicity violation* error is caused by concurrent execution of multiple threads violating the atomicity of a certain code region [21].
- Deadlock: *Deadlock* happens when multiple threads are blocked forever, waiting for each other.
- Livelock: *Livelock* happens when two threads are busy responding to each other and make no progress.
- Starvation: *Starvation* happens when a thread is unable to gain regular access to shared resources and is unable to make progress.
- Suspension: *Suspension* happens when a thread suspends or waits indefinitely.

2.2 Synchronization Mechanism

Concurrent programs rely on synchronization to ensure correct program execution. There are two main synchronization mechanisms: synchronization using shared memory and synchrnonization using message passing. For programming models that use shared memory synchronization (*e.g.*, Java and C#), the threads communicate primarily by sharing access to fields and the objects reference fields refer to. The synchronization aims to avoid thread interference and memory consistency errors. For programming models that use message passing (*e.g.*, Erlang [3] and Microsoft Asynchronous Agents Library [22]), the concurrent agents or actors in the programs communicate with each other through exchanging messages and use the synchronization to avoid problems in the message communications.

3 Limitations of First-Order Concurrency Mutation Operators

To measure the testability of concurrent Java programs, Ghosh described mutation based on two mutation operators that remove the keyword `synchronized` [14]. Long *et al.* tested mutation-based exploration for concurrent Java components [19]. Delamaro *et al.* proposed a set of 15 concurrency mutation operators for Java [7]. Later, Bradbury *et al.* proposed a new set of 24 concurrency mutation operators for Java [4]. The operators they proposed are all first-order mutation operators. We have investigated these mutation operators in our empirical study and identified some of their limitations.

3.1 Subtle Concurrency Bugs Are Not Generated

The first important limitation we found is that some subtle concurrency bugs are not generated by any of these proposed first-order mutation operators. This limitation could lead to loss of comprehensive representation of common concurrency bugs by the mutants, thus reducing the reliability of the mutation score that follows. We give two examples in the following subsections.

3.1.1 Data Race Example

The following code fragment from the *LinkedList* program, a Java program from the IBM concurrency benchmark programs repository [12], inserts an element to the end of a

list. Another process, not shown here, reads the list. The synchronized method first starts from the top of the list (line 4), then moves to the end of the list via a loop (line 5) before inserts the object `x` to the end (line 6). Suppose we only apply first-order mutation operators (*e.g.*, removing `synchronized` keyword from line 2 or deleting a statement from line 4 to 6), the mutant does not represent a feasible programming error that line 2 is not synchronized and line 5's error causes the node index `itr` does not move to the end of the list properly. The combined error in line 2 and line 5 would potentially cause data race because multiple threads would try to write to the header of the list without synchronization and other threads might read the list at the same time. The outcome of the execution would depend on the thread schedule and which thread made the last call to the method because different threads all try to update the header of the list. By definition, this is a data race condition. To apply either operator independently is not going to create the same fault because under single application of either first-order mutation operator, data race would less likely happen and their mutants would represent different kind of faults.

```
1   /* Inserts element to the end of list */
2   public synchronized void addLast( Object x )
3   {
4      MyListNode itr = this._header;
5      while( itr._next != null ) itr = itr._next;
6      insert(x,new MyLinkedListItr(itr));
7   }
```

3.1.2 Deadlock Example

Incorrect use of synchronization can result in two or more threads waiting for each other to release the locks on the synchronized objects, forming a deadlock circle. As shown in the following example code for money transfer between two accounts, the line 5 and 6 in the original code may be incorrectly programmed in a nested synchronized block, which makes the deadlock possible. For example, two threads with execution of line 9 and 10 simultaneously would lead to deadlock since each thread will be waiting in a circle for the other thread to release required lock. This kind of deadlocks that require changes in more than one place are not generated by any first-order operator.

```
1   void TransferMoney(Acct a, Acct b, int amount) {
2      synchronized(a) {
3         a.debit(amount);
4      }
5       synchronized(b) {
6         b.credit(amount);
7      }
8   }
1   void TransferMoney(Acct a, Acct b, int amount) {
2       synchronized(a) {
3         synchronized(b) {  //first change
4            a.debit(amount);
5            b.credit(amount);  //second change
6         }
7      }
8   }
9   Thread1.run() { TransferMoney(a,b,10); }
10  Thread2.run() { TransferMoney(b,a,20); }
```

3.2 A Large Portion of Mutation Operators Do Not Generate Any Mutant

Our empirical study show that a large portion of existing mutation operators are not effective in generating mutants. For example, several previously proposed mutation operators for concurrent Java, including MSF, MXC, MBR, RCXC, ELPA, EAN, RSTK, RFU, RXO and EELO [7, 4], did not generate any mutants in our experiments. Some others, including MXT, RNA, RJS, InsNegArg, and ReplTargObj [7, 4], generated very few mutants. In our assessment, over half of the total number of operators are non-performing mutation operators, *i.e.*, operators that do not generate any new mutant. Most of the performing ones are related to mutation of a synchronized method or block.

4 Approach

4.1 Synchronization-Centric Second-Order Concurrency Mutation Operators

Our new mutation testing approach is based on our fault model and our analysis of the limitations of some previous work. We use synchronization-centric second-order concurrency mutation operators to construct subtle concurrency bugs that are not represented by the first-order mutation. While, a random and brute-force approach without any reduction would lead to $n * n$ second-order mutation operators based on n first-order mutation operators. To reduce the number of second-order mutation operators and mutant execution cost, we employ two steps of reduction. We first choose one of the two first-order mutation operators to be a synchronized method or block related modification and the other first-order operator to perform code changes related to the same synchronized method or block. For example, in concurrent Java, there are five first-order mutation operators related to synchronized methods [7, 4], we choose two out of the five in the same category. Then we evaluate the chosen two first-order mutation operators to see if their combination can generate mutants that resembles some possible faults due to programming mistakes and only keep those meaningful combinations. This second reduction step through selection based on domain knowledge further reduces the amount of second-order mutation operators and leads to fewer unnecessary or redundant mutants.

After the set of synchronization-centric second-order concurrency mutation operators are chosen, they are combined with the synchronization-centric first-order mutation operators to form a smaller set of mutation operators for mutant generation.

4.2 Example Mutation Operators for Java

Table 1 lists the synchronization-centric second-order concurrency mutation operators for Java, as an example of synchronization using shared memory. We describe each operator with example code in the following subsections.

Table 1. Second-Order Concurrency Mutation Operators for Java

sync method	RKSN+RSSN	Remove `synchronized` Keyword and a Statement from Synchronized Method
	AKST+MASN	Add `static` Keyword and Modify Argument with Constant to Synchronized Method
	RKSN+MASN	Remove `synchronized` Keyword and Modify Argument with Constant
sync block	RSNB+RSSB	Remove `synchronized` Block and a Statement from Synchronized Block
	MOSB+RSSB	Modify `synchronized` Object and Remove a Statement from Synchronized Block
	MOSB+MVSB	Modify `synchronized` Object and Move Statement(s) Out of Synchronized Block

4.2.1 RKSN+RSSN

The RKSN+RSSN operator removes a `synchronized` keyword and a statement from a synchronized method. This operator simulates programming errors that can potentially lead to data race, memory inconsistency, and deadlock. The data race described in Section 3.1.1 can be constructed by this operator.

```
/* Original Code */
public synchronized void proc(Object A) {
   <statement 1>
   <statement 2>
}
/* RKSN+RSSN Mutant 1 */
public void proc(Object A) {  // sync removed
   ...// statement removed
   <statement 2>
}
/* RKSN+RSSN Mutant 2 */
public void proc(Object A) {  // sync removed
   <statement 1>
   ...// statement removed
}
```

4.2.2 AKST+MASN

The AKST+MASN operator adds a `static` keyword and modifies an argument with a constant to a synchronized method. This operator simulates programming errors that can potentially lead to data race and memory inconsistency.

```
/* Original Code */
public synchronized void send(String m) {...}
/* AKST+MASN Mutant */
public static synchronized void send(String n) {...}
```

4.2.3 RKSN+MASN

The RKSN+MASN operator removes a `synchronized` keyword and modifies an argument with a constant to a synchronized method. This operator simulates programming errors that can potentially lead to data race and memory inconsistency.

```
/* Original Code */
public synchronized void send(String m) {...}
/* AKST+MASN Mutant */
public void send(String n) {...}
```

4.2.4 RSNB+RSSB

The RSNB+RSSB operator removes the synchronized block and a statement from a synchronized block. This operator simulates programming errors that can potentially lead to data race, memory inconsistency, and deadlock.

```
/* Original Code */
synchronized (this) {
   <statement 1>
   <statement 2>
}
/* RSNB+RSSB Mutant 1 */
... // removed
    ... // removed
   <statement 2>
...
/* RSNB+RSSB Mutant 2 */
... // removed
   <statement 1>
    ... // removed
...
```

4.2.5 MOSB+RSSB

The MOSB+RSSB operator modifies a synchronized object and removes a statement from a synchronized block. This operator simulates programming errors that can potentially lead to data race and memory inconsistency.

```
/* Original Code */
synchronized(obj1) {
   <statement 1>
   <statement 2>
}
/* MOSB+RSSB Mutant */ // object modified
synchronized(newobj) {
   <statement 1>
    ... // removed
}
/* MOSB+RSSB Mutant */
synchronized(newobj) { // object modified
    ... // removed
   <statement 2>
}
```

4.2.6 MOSB+MVSB

The MOSB+MVSB operator modifies a synchronized object and moves statement(s) out of a synchronized block. This operator simulates programming errors that can potentially lead to data race, deadlock, memory inconsistency, and atomicity violation. The deadlock described in Section 3.1.2 can be constructed by this operator, *i.e.*, moving two lines of code including the synchronized block.

```
/* Original Code */
synchronized(obj1) {
   <statement 1>
   <statement 2>
    ...
}
/* MOSB+MVSB Mutant 1*/
<statement 1>  // moved
synchronized(newobj) { // object modified
   <statement 2>
    ...
}
/* MOSB+MVSB Mutant 2*/
<statement 1>  // moved
<statement 2>  // moved
synchronized(newobj) { // object modified
    ...
}
```

4.3 Example Mutation Operators for Erlang

Table 2 lists the synchronization-centric second-order concurrency mutation operators for Erlang, as an example of synchronization using message passing. The CRT (*i.e.*,

Change Reference Type) refers to changing a message reference from *Send by Ref* to *Send by Val*, and vice versa [15]. The CST (*i.e.*, Change Synchronization Type) refers to changing a message's synchronization method from *Sync Send* to *Async Send*, and vice versa. Since these mutation operators are self-explanatory, we do not give detailed example code here.

Table 2. Second-Order Concurrency Mutation Operators for Erlang

Messaging	CRT+MMP	Change reference type and modify message parameter
	CRT+RMP	Change reference type and reorder message parameter
	CRT+MMN	Change reference type and modify message name
	CRT+MMR	Change reference type and modify message recipient
	CST+MMP	Change sync type and modify message parameter
	CST+RMP	Change sync type and reorder message parameter
	CST+MMN	Change syn type and modify message name
	CST+MMR	Change syn type and modify message recipient
Constraint	CRT+RC	Change reference type and remove constraint
	CRT+MC	Change reference type and modify constraint
	CST+RC	Change syn type and remove constraint
	CST+MC	Change syn type and modify constraint

5 Empirical Study

5.1 Implementation

We developed an Eclipse Plug-in [11] named BUGGEN that is able to automate mutant generation after the specific mutation operator is selected. Eclipse is a popular integrated development environment (IDE) with an extensible plug-in system. Building BUGGEN as an Eclipse Plug-in leverages the functionalities of the Eclipse and simplifies software development. During our implemention, we found our set of mutation operators are easy to implement. In our empirical study, we focus on concurrent Java.

5.2 Example Programs

We use the following four example programs in our experiments to study mutant generation, as well as the cost and effectiveness of each proposed operator,

- *Webserver*, a Java web server program that supports concurrent client connections and synchronization [2].
- *Chat*, a Java chat program that supports multiple clients exchanging messages [10].
- *Miasma*, a graphical Java applet program from the NIH web-site [18]. It supports synchronization and uses `wait(t)` for prior pixels to be accepted before triggering another one.
- *LinkedList*, a modified Java program from the IBM concurrency benchmark programs repository [12]. The original program was developed to emulate the concurrency bug in using Java linked list, which is a non-synchronized collection.

We select the above example programs because they all employ different concurrency features and these programs are diversified in terms of type, size, coding style, applied field, and developer. These programs are representive in demonstrating common programming practices using concurrent Java. Table 3 lists some statistical information for each program's source code.

Table 3. Example programs

Program Name	LOC	classes	sync methods	sync blocks
Webserver	125	6	11	2
Chat	482	4	10	2
Miasma	360	1	0	2
LinkedList	421	5	1	1
Total	1,388	16	22	7

5.3 Mutant Generation Results and Analysis

In our experiments, we apply each of the mutation operators listed in Table 1, along with the synchronization-centric first-order mutation operators, on the example programs, count the number of mutants generated by each operator for each program, and then examine these mutants. From our experiments, we found that over half of the first-order mutation operators, especially those that are not related to synchronization, are not effective in generating mutants. The majority of the mutants are generated by synchronization-centric mutation operators. Our quantitative data and summations for each category are recorded in the histogram chart presented in Figure 1. Details for each operator and the example programs can be found in our technical report [27]. The vertical axis shows the number of mutants. Most synchronization-centric mutation operators, in particular the second-order ones, are effective in mutant generation.

Our empirical study demonstrates that subtle concurrency bugs not represented by the first-order mutation are generated by the second-order mutation operators; our mutant generation effort is limited; fewer percentage of equivalent mutants are generated. Second-order operators tend to decrease the percentage of equivalent mutants [26].

6 Related Work

Some prior studies have been done on mutation testing for concurrent programs [17]. Carver described deterministic execution mutation testing and debugging of concurrent programs using synchronization-sequence [5]. Researchers have developed many sets of mutation operators [25, 17], targeting a variety of programming languages. Other than the mutation operators for concurrent Java, Jagannath *et al.* have proposed a set of mutation operators for actor programming model [15]. Our synchronization-centric second-order mutation operators for message passing also apply to the actor programming model.

For higher-order mutation, Polo *et al.* studied mutation cost reduction using second-order mutants [26]. Jia *et al.* described some general cases of higher-order mutation and related algorithms [16]. In our approach, we used second-order mutation to construct some subtle concurrency faults.

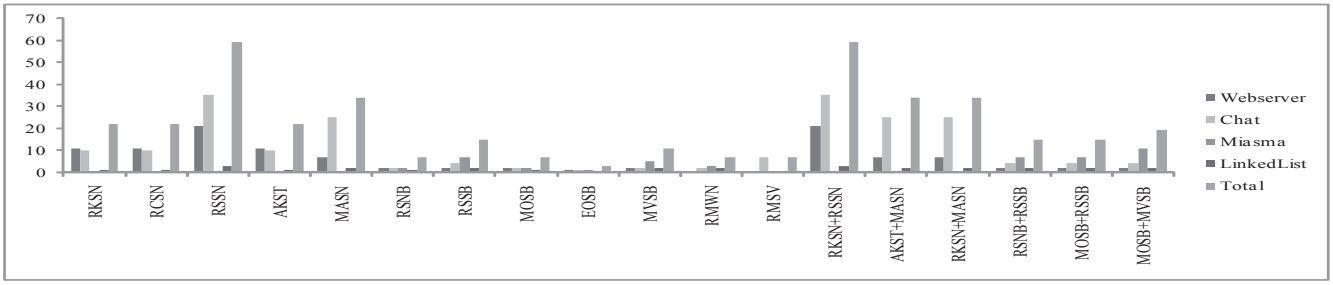


Figure 1. Number of mutants generated per operator

By keeping a small number of second-order mutation operators based on synchronization and reduction through domain analysis, we avoided the drastic growth of the number of mutants, thus avoiding higher computing cost in mutant execution. To the best of our knowledge, our work is the first study of the second-order mutation operators specifically for concurrent programs.

7 Conclusion

This paper first described a general fault model for concurrent programs and some limitations of previously developed sets of first-order concurrency mutation operators. We then presented our new mutation testing approach, which employs synchronization-centric second-order mutation operators that are able to generate subtle concurrency bugs not represented by the first-order mutation. These operators are used in addition to the synchronization-centric first-order mutation operators to form a small set of effective concurrency mutation operators that can be used in mutant generation. We developed an Eclipse Plug-in named BUGGEN to automate the mutant generation using these operators. Our empirical study showed that our set of mutation operators is effective in mutant generation with limited cost and this new approach is easy to implement. For future work, we plan to evalute some concurrency testing suites using the set of mutation operators.

8 Acknowledgments

Wu and Kaiser are members of the Programming Systems Laboratory, funded in part by NSF CNS-0717544, CNS-0627473 and CNS-0426623, and NIH 2 U54 CA121852-06.

References

[1] A. T. Acree, T. A. Budd, R. A. DeMillo, R. J. Lipton, and F. G. Sayward. Mutation analysis. Technical Report GIT-ICS-79/08, Georgia Institute of Technology, Atlanta, Georgia, 1979.

[2] J. Aldrich, E. G. Sirer, C. Chambers, and S. J. Eggers. Comprehensive synchronization elimination for java. *Science of Computer Programming*, 47(2-3):91–120, 2003.

[3] J. Armstrong, R. Virding, C. Wikström, and M. Williams. *Concurrent Programming in ERLANG*. Prentice Hall, 1993, Second Edition.

[4] J. S. Bradbury, J. R. Cordy, and J. Dingel. Mutation operators for concurrent java (j2se 5.0). In *Proceedings of the Second Workshop on Mutation Analysis (Mutation '06)*, pages 11–11. IEEE Computer Society, 2006.

[5] R. Carver. Mutation-based testing of concurrent programs. In *Proceedings of the International Test Conference*, pages 845–853, 1993.

[6] S.-E. Choi and E. C. Lewis. A study of common pitfalls in simple multi-threaded programs. In *Proceedings of the 31st SIGCSE Technical Symposium on Computer Science Education*. ACM, 2000.

[7] M. Delamaro, M. Pezzé, A. M. R. Vincenzi, and J. C. Maldonado. Mutant operators for testing concurrent java programs. In *XV Simpósio Brasileiro de Engenharia de Software*, pages 272 – 285, Rio de Janeiro, RJ, Brasil, 2001.

[8] R. A. DeMillo, D. S. Guindi, W. M. McCracken, A. J. Offutt, and K. N. King. An extended overview of the mothra software testing environment. In *Proceedings of the Second Workshop on Software Testing, Verification, and Analysis*, pages 142–151, 1988.

[9] R. A. DeMillo, R. J. Lipton, and F. G. Sayward. Hints on test data selection: Help for the practicing programmer. *Computer*, 11(4):34–41, 1978.

[10] D. J. Eck. Chat. available at `http://math.hws.edu/eck/cs124/s06/lab11/index.html`, 2006.

[11] Eclipse. Eclipse.org. available at `http://www.eclipse.org`, 2010.

[12] Y. Eytani and S. Ur. Compiling a benchmark of documented multi-threaded bugs. In *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS '04)*, page 266, 2004.

[13] E. Farchi, Y. Nir, and S. Ur. Concurrent bug patterns and how to test them. In *Proceedings of the 17th International Symposium on Parallel and Distributed Processing*. IEEE Computer Society, 2003.

[14] S. Ghosh. Towards measurement of testability of concurrent object-oriented programs using fault insertion: a preliminary investigation. In *Proceedings of the Second IEEE International Workshop on Source Code Analysis and Manipulation*, pages 17–25, 2002.kim

[15] V. Jagannath, M. Gligoric, S. Lauterburg, D. Marinov, and G. Agha. Mutation Operators for Actor Systems In *2010 Third International Conference on Software Testing, Verification, and Validation Workshops (ICSTW)*, pp. 157–162, 2010.

[16] Y. Jia and M. Harman. Constructing subtle faults using higher order mutation testing. In *Proceedings of the Eighth IEEE International Working Conference on Source Code Analysis and Manipulation*, pages 249–258, 2008.

[17] Y. Jia and M. Harman. An analysis and survey of the development of mutation testing. *IEEE Transactions of Software Engineering*, 2010.

[18] JRP. Miasma. available at `http://rsb.info.nih.gov/miasma/Miasma.java`, 2010.

[19] B. Long, R. Duke, D. Goldson, P. Strooper, and L. Wildman. Mutation-based exploration of a method for verifying concurrent java components. In *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS '04)*, page 265, 2004.

[20] B. Long and P. Strooper. A classification of concurrency failures in java components. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS '03)*, pp. 8, 2003.

[21] S. Lu, S. Park, E. Seo, and Y. Zhou. Learning from mistakes: a comprehensive study on real world concurrency bug characteristics. In *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '08)*. ACM, 2008.

[22] Microsoft Asynchronous Agents Library http://msdn.microsoft.com/en-us/library/dd492627(VS.100).aspx

[23] A. J. Offutt. Investigations of the software testing coupling effect. *ACM Transactions on Software Engineering and Methodology*, 1(1):5–20, 1992.

[24] A. J. Offutt, A. Lee, G. Rothermel, R. H. Untch, and C. Zapf. An experimental determination of sufficient mutant operators. *ACM Transactions on Software Engineering and Methodology*, 5(2):99–118, 1996.

[25] A. J. Offutt and R. H. Untch. Mutation 2000: uniting the orthogonal. *Mutation Testing for the New Century*, pages 34–44, 2001.

[26] M. Polo, M. Piattini, and I. García-Rodríguez. Decreasing the cost of mutation testing with second-order mutants. *Software Testing, Verification and Reliability*, 19(2):111–131, 2009.

[27] L. Wu and G. Kaiser. Empirical study of concurrency mutation operators for java. Technical Report CUCS-041-10, Department of Computer Science, Columbia University, 2010.

The ucsCNL: A Controlled Natural Language for Use Case Specifications

Flávia A. Barros, Laís Neves
Center of Informatics
Federal University of Pernambuco
Recife (PE), Brazil
Email: {fab, lmn3}@cin.ufpe.br

Érica Hori
MV Sistemas
Apply Solutions
Recife (PE), Brazil
Email: ericahori@gmail.com

Dante Torres
Other Ocean Ltd
Newfoundland, Canada
Email: dantegt@gmail.com

***Abstract*— In general, test generation tools receive as input either requirements or use case specifications. However, in most companies these specifications are written in free natural language (NL), and the lack of standardization may become a problem for the generation tools and testers. A promising solution is to use a Controlled NL (CNL) to write software specifications. We present here the ucsCNL for the authoring of Use Case specifications. The ucsCNL is implemented as a plug-in of TaRGeT, a tool for the automatic generation of feature test cases based on use case scenarios written in English. The ucsCNL is already in use, and has achieved satisfactory results.**

I. Introduction

Software Testing has grown in importance in recent years, as a way to increase quality and reliability of the final product. Several tools to automate software testing tasks have been proposed, from test generation to its execution. In particular, automatic test generation has been the focus of several works [1], due to the drawbacks of manual test design, which is time-consuming, and not always systematic and precise.

Usually, test generation tools receive as input either requirements or use case (UC) specifications, from which the test cases are derived. This input should be unambiguous, to preserve the quality of the testing process. However, in most companies requirements and use cases are written using free natural language, and the absence of standardization may become a problem both for the generation tool, and for the professionals who manually execute the test suits.

A promising solution is to use a Controlled Natural Language (CNL) to restrict the creation of requirements and/or use cases. A CNL is a subset of natural language that uses a restricted set of grammar rules and a predefined vocabulary, in order to avoid textual complexity and ambiguity [2].

We present here the ucsCNL, a subset of English language designed for the authoring of UC specifications. The ucsCNL counts on two knowledge Bases: a Lexicon, with the domain-specific vocabulary; and a Grammar, used to restrict the sentence constructions allowed for writing the UC specifications.

The ucsCNL is implemented as a plug-in of TaRGeT [3], a tool for the automatic generation of feature (black box) test cases based on use case scenarios. The ucsCNL is already in use, and has achieved satisfactory results. Initial experiments showed a better performance of testers when manually executing test generated from UCs written using the ucsCNL *versus* UCs written in free English. This is an original work which combines techniques from Software Engineering, Artificial Intelligence and Natural Language Processing.

Section II discusses research in the CNL field. Section III presents the ucsCNL in detail, section IV shows implementation details, and section V brings conclusions and future work.

II. Controlled Natural Languages and Software Specifications

In order to safeguard the quality of the input specifications in the software development process, some companies use a Controlled Natural Language specially designed to address their particular communicative needs. A CNL [2] is a subset of some NL which uses:

(1) a domain-specific vocabulary (Lexicon), in order to avoid *synonymy* (i.e., two different terms referring to the same entity) and *lexical ambiguity* (i.e., the same term referring to two or more entities in the application domain); and

(2) a restricted set of grammar rules, which can be general (e.g., 'write short and simple sentences'), or more formal, counting on grammar rules to constrain the accepted syntactic structures (in order to avoid *structural ambiguity* - a sentence being mapped into two or more different syntactic structures).

It is possible to identify two major groups of CNLs, aimed at different purposes. The most basic goal of a CNL is to define a standard to be followed throughout an organization, in order to provide for unambiguous and clear technical documentation (e.g., ASD-STE100 Simplified Technical English [4]). These CNLs are also known as 'simplified', or 'technical languages', since their Grammar is basically a set of general writing rules (e.g., 'write short and grammatically simple sentences', 'use active instead of passive voice' [5]).

The second group of CNLs is at the more formal side. In case a mapping from CNL specifications to a formal representation is desired, the CNL must use a precise syntax and semantics. Based on a formal grammar and a controlled lexicon, it is then possible to define a mapping from CNL sentences into a more formal representation (such as First-Order Logic - FOL) - e.g., PENG [6]. These formal representations

can be used for further processing, such as model checking [7], automatic test generation [8], and so on.

We highlight here the Attempto project[1] and works derived from the Attempto Controlled English (ACE) [9], such as the work of [10], which focuses on Requirement specifications.

Finally, [11] presents a tool for modal-based test generation from 'natural-language-like' functional specifications. Requirements are represented using a Template Based Natural Language Specification (TBNLS), where each requirement corresponds to a TBNLS representation. These TBNLSs are mapped into a Formal Requirement Language in which requirements are represented as tuples: (start-condition (if), consequence (then), end-condition (until)). Clearly, this is a too restricted grammar for our purposes.

III. A Controlled Natural Language for Use Case Authoring

This section presents the ucsCNL developed for the authoring of use cases specifications. Our CNL is not merely a technical language, since we are interested in the automation of the whole testing process. This way, we designed a more formal CNL, counting on a Lexicon with pre-defined word types and terms, and a Grammar, used to restrict the sentence constructions allowed for writing the UC specifications.

The ucsCNL version presented in this paper was developed for the mobile phone domain. However, our CNL can be adapted to different application domains. In this work, UC specifications consist of three basic fields:

(1) *Initial Condition*, stating the system's state before the test takes place (e.g., 'The phone is in data connection screen');

(2) *User Action*, stating the test steps to be executed by the tester (e.g., 'Cancel the operation pressing END key'); and

(3) *System Response*, stating the result of an action or the system state after the test was completed (e.g., 'The operation was successfully canceled').

The ucsCNL Grammar provides rules for the three UC fields (section III-B). Section III-A presents the ucsCNL Lexicon.

A. The ucsCNL Lexicon

The Lexicon contains the ucsCNL vocabulary, whose terms are classified into 7 different lexical classes (also known as 'parts of speech') [12]: noun (representing the domain entities), verb (representing an action or an event that may occur in the domain), determiner (a noun modifier - detailed below), adjective, adverb, preposition, and conjunction.

Determiners include articles (a, the), quantifiers (every, all, some), and numerals (cardinals and ordinals). Pronouns such as 'this', 'that' or 'it' are not allowed, in order to avoid ambiguity caused by pronominal reference (i.e., anaphora).

Determiners, prepositions and conjunctions are closed word classes, that is, they contain a stable (fixed) set of words which do not depend on the application domain. Terms belonging to these categories are already provided by the ucsCNL initial vocabulary, and cannot be added or erased by the user. Only the system's administrator is allowed to edit these classes. The user is free only to add nouns, verbs, adjectives and adverbs.

Verb entries contain the infinitive form plus inflections: present; past simple; past participle (also used as adjective); and gerund form, also used as a noun. Modal verbs (e.g., may, can, could) are not allowed, since they express the idea of 'possibility', and we need precise statements regarding Use and Test Cases, in order to avoid misinterpretations.

B. The ucsCNL Grammar

The ucsCNL grammar consists of a set of rewriting rules used to restrict the sentence constructions allowed for writing UC specifications. We work within the Immediate Constituents Grammar approach [12], according to which a sentence can be analyzed as a combination of units - immediate constituents- (e.g., sentence = noun phrase + verb phrase + prepositional phrase). These constituents can, in turn, be analyzed as smaller constituents (e.g., noun phrase = article + noun), until reaching irreducible constituents (such as nouns, verbs, etc).

Sentences in the different UC fields are based on different syntactic structures. To specify the *User Action* field, imperative sentences are used (e.g., 'Press the END key'). On the other hand, *Initial Condition* and *System Response* sentences convey, respectively, the system state before and after an action is taken. Here, active and passive sentences are used.

In order to simplify the parsing process, two separate grammars were defined (sections III-B.2 and III-B.3). This way, the parsing of each UC field is guided by a different set of rules. The ucsCNL also counts on a Base Grammar, which defines rules for basic constituents (section III-B.1).

The ucsCNL *parser* receives an input sentence and returns its grammatical structure, according to the Lexicon and Grammar under use (section IV-A). The process starts by attributing the input sentence to the token 'Sentence', which is the root of the parsing tree. Thus, the UA and IC-SR grammars start with a rewriting rule for this token (see tables III and IV).

1) Base Grammar: The Base Grammar defines rules for the basic constituents, namely: noun phrase, determiner, qualifier, prepositional phrase, relative clause, verb phrase and gerund phrase. These basic constituents are used by all other ucsCNL grammars, which actually extend the Base Grammar with field-specific rules built upon these general constructions.

A noun phrase (NP) is "a group of words in a sentence which together behave as a noun" [13]. Qualifiers are defined as "a word or phrase which limits the meaning of another word or phrase, or makes it less general, such as an adjective or adverb" [13]. An NP may end with a Prepositional Phrase or a Relative Clause, to add more information about the preceding noun. A Prepositional Phrase consists of a preposition followed by an NP.

Table I presents some rules for NP, Determiner and other constituents. The comma sign represents the 'AND' connector, and the vertical bar represents the 'OR' connector. Constructions ending with a question mark are optional, and brackets indicate that only one of the constructions between

[1] http://attempto.ifi.uzh.ch/site/

TABLE I
BASE GRAMMAR REWRITING RULES (NOUN PHRASE)

NounPhrase = NP, [‘and’, NP]?
NP = Determiner?, QualifierList?, Noun, (PrepositionalPhrase | RelativeClause)?
Determiner = DT, OD?, CardinalList? Determiner = CardinalList CardinalList = CD, CardinalList?
Qualifier = ADV?, (ADJ | VBN) QualifierList = Qualifier, QualifierList?, [’and’, Qualifier]?
PrepositionalPhraseList = PrepositionalPhrase, PrepositionalPhraseList?, [‘and’, PrepositionalPhrase]?
RelativeClause = ‘that’, RCVerbPhrase

TABLE II
BASE GRAMMAR REWRITING RULES (VERB RULES)

VerbPhrase = Verb, VerbComplement?
Verb = VB | ActiveVerbPresent | ActiveVerbPast | VerbToBe
ActiveVerbPresent = VBP | VBZ
ActiveVerbPast = VBD
VerbToBe = (VTBP | VTBZ | VTBDP | VTBDZ), ‘not’?
VerbComplement = SimpleVC | ComplexVC
SimpleVC = (NounPhrase, PrepositionalPhraseList?) | PrepositionalPhraseList
ComplexVC = SimpleVP, ‘by’, GerundPhrase
GerundPhrase = VBG, SimpleVC

TABLE III
USER ACTION GRAMMAR MAIN RULES

Sentence = UASentenceList
UASentenceList = UASentence, UASentenceList?, [‘and’, UASentence]?
UASentence = ImperativeSentence | NegativeImperativeSentence | SubordinatedImperativeSentence
ImperativeSentence = VB, VerbComplement, ToComplement?, ADV? ToComplement = ‘to’, VB, NounPhrase?
NegativeImperativeSentence = ‘Do not’, ImperativeSentence, ADV?
SubordinatedImperativeSentence = CJ, GerundPhrase, (ImperativeSentence | NegativeImperativeSentence)

TABLE IV
INITIAL CONDITION AND SYSTEM RESPONSE GRAMMAR RULES.

Sentence = SRSentence, [‘and’, SRSentence]?
SRSentence = SRNucleus, SRComplementList?
SRNucleus = ActiveVoiceSentence | PassiveVoiceSentence | ThereBeSentence | VerbToBeSentence
ActiveVoiceSentence = NounPhrase, Neg?, ADV?, ActiveVerbPresent, ADV?, SimpleVC?, ADV?
PassiveVoiceSentence = NounPhrase, VerbToBe, ADV?, VBN, ADV?
VerbToBeSentence = NounPhrase, VerbToBePresent, (NounPhrase | PrepositionalPhrase | QualifierList)
SRComplement = ToComplement | [PrepositionalPhraseList, ADV?] | [CJ, SRNucleus]

the signs can be used. Finally, ‘[]’ indicates that all constructions between the signs must occur together. Some sentence formations require a particular word occurring in a given position (rather than any word from a particular class). Words between quotation marks must be literally interpreted by the rules (e.g., ‘and’ in Table I).

We introduce now the acronyms (tokens) used to represent the Lexicon entries: noun = Noun, verb = VB and other tokens (see below), determiner = DT, adjective = ADJ, adverb = ADV, preposition = PP, and conjunction = CJ. Numerals have their own entries: ordinals = OD and cardinals = CD. The main Verb entries are: VB (infinitive form) and VTB (verb to be). The verb conjugations are represented by adding suffixes to these acronyms: D - past tense; N - past participle; P - non-3rd person singular present; Z - 3rd person singular present. Finally, the gerund form is represented by G suffix.

A verb phrase (VP) is a group headed by a verb and may consist of a single verb, or auxiliary and main verbs, with optional complements and adjuncts (e.g., ‘The agenda *has three entries*’). Table II brings some rules for verbs.

2) User Action Grammar: User Action (UA) sentences are *imperative sentences* that convey an action to be executed by the tester (for manual test execution). UA sentences may also state how the action should be executed (e.g., ‘Send a message *by pressing the SEND key*’). This grammar allows imperative sentences in the *negative form*, as well as *subordinate clauses* to an imperative sentence (e.g., ‘*After creating a message with 100 characters*, go to the drafts folder’).

Note that imperative sentences do not have an explicit subject, since UC actions are commands to the tester. This way, the UA rules do not accept an NP in the beginning of the sentence (see Table III).

3) Initial Condition and System Response Grammar: Initial Condition (IC) sentences are statements defining the system state before a test is started. In turn, System Response (SR) sentences convey messages stating how the system is expected to behave after some action has been taken by the tester. We observed that IC and SR sentences have similar grammar structures. This way, the ucsCNL uses the same grammar rules for the parsing of both IC and SR fields (Table IV).

The IC and SR fields convey four different kinds of sentences: active and passive voice sentences, existential sentences, and the so-called ‘verb To Be’ sentences. The ‘verb to be’ sentences are simple formations that denote the state of a domain entity, or better describe it (e.g., ‘The imported media file is a music file’).

IV. Implementation Details

The ucsCNL is implemented as plug-in of a larger system for feature (black box) test cases generation, the TaRGeT (Test and Requirements Generation Tool) [3]. TaRGeT was developed as a software product line, and its main purpose is to automate a systematic approach to generate feature test suites from use case specifications written in the ucsCNL.

TaRGeT receives as input use cases scenarios written in CNL, thus aiding the test designers/engineers, who are more familiar with NL sentences than with formal specification languages. The system model and test suites are automatically derived from these UC scenarios.

A. The ucsCNL Advisor

The ucsCNL plug-in is implemented as the TaRGeT *CNL Advisor* view at the workbench. This advisor gives support to the process of UC authoring, obeying the ucsCNL Grammar and Lexicon (in order to minimize possible mistakes in the testing phase). In order to conform with TaRGeT's software product line approach, the ucsCNL Lexicon and Grammar bases are represented as XML files.

The CNL Advisor counts on 3 modules: a pre-processing module, a POS-tagger and a context free parser. Initially, the pre-processing module verifies whether all words in the sentence are included in the Lexicon. When a new word is detected, the user receives an error message. If the word belongs to an open class (noun, verb), the user will able to add it to the Lexicon via a simple user interface. When the word belongs to a closed class, the user receives a message advising him/her to modify the input sentence, or to contact the system's administrator, who has special priority to edit the Lexicon and the Grammar.

Following, the POS-tagger tags all words in the input sentence with their lexical class. Words that belong to more than one class are tagged with all possibilities, and the parsing process will disambiguate the double categorization based on the available grammar rules. The parser receives a list of words with their lexical class, and builds one tree for every possible syntactic structure. Sentences that could not be correctly parsed are returned to the user. In the case of syntactic error, the Advisor indicates the word categories that were expected instead, and gives examples of correct sentences. The Advisor also allows the user to filter the errors, displaying only the user action, initial condition or system response sentences.

B. Empirical tests

Empirical tests were performed to evaluate the impact and the benefits of the ucsCNL for text standardization. Although this is not the unique aim of our CNL, this preliminary test already indicated the benefits of text standardization. This case study was executed in the context of a mobile device testing company. The aim was to compare the gains of manually executing TC suites created using the ucsCNL versus suites written in free natural language. We considered here: (1) the execution time (the ucsCNL performed better in 35% of the TCs); and (2) number of detected defects/bugs (the ucsCNL detected one extra bug, and avoided a false defect accused by the free LN TCs). The results demonstrate a slight advantage of the use of ucsCNL in relation to free natural language. New tests will be performed in order to confirm this advantage, as well as to identify other gains with the use of the ucsCNL.

V. Conclusion and Future Work

This paper presented the ucsCNL for the authoring of Use Case specifications, from which test cases can be automatically derived by TaRGeT. The main aim of this work was to aid users (in particular, test designers) who are more familiar with natural language descriptions than with formal specifications.

We are currently developing a mechanism for the automatic mapping from ucsCNL sentences into a formal language representation, to automatically generate test scripts from these formal specifications. Another future work is the inclusion of ontology-based semantic analysis to check the coherence of the UC input specifications according to the specific domain application being tackled.

Acknowledgments: This work was partially supported by the National Institute of Science and Technology for Software Engineering (INES www.ines.org.br), funded by CNPq and FACEPE, grants 573964/2008-4 and APQ-1037-1.03/08.

References

[1] A. Belinfante, L. Frantzen and C. Schallhart, "Tools for Test Case Generation," Book Chapter, in [7], 2005, pp. 391–438. Available at: http://www.cs.ru.nl/ lf/publications/BFS05.pdf, access date: 10 March 2011.

[2] R. Schwitter, "Controlled natural language," Available at: http://sites.google.com/site/controllednaturallanguage/, access date: 10 March 2011.

[3] P. Borba, D. Torres, R. Marques, and L. Wetzel, "TaRGeT - Test and requirements generation tool," in Motorola's 2007 Innovation Conference (IC'2007), 2007.

[4] "ASD simplified technical english," in Specification ASD-STE100, International specification for the preparation of maintenance documentation in a controlled language. Available at: http://www.asd-ste100.org/, access date: 10 March 2011.

[5] U. Muegge, "Controlled language - the next big thing in translation?" ClientSide News Magazine, vol. 7, 2001, pp. 21-24.

[6] R. Schwitter, "English as a formal specification language," in 13th International Workshop on Database and Expert Systems Applications (DEXA 2002). IEEE Computer Society, USA, 2002, pp. 228–232.

[7] M. Broy, B. Jonsson, J.-P. Katoen, M. Leucker, and A. Pretschner (Eds.), "Model-based testing of reactive systems: Advanced lectures," in Lecture Notes in Computer Science vol. 3472, 2005, 659 p.

[8] D. Leitao, D. Torres, and F. A. Barros, "NLForSpec - Translating natural language descriptions into formal test case specifications," in Proc. of the Nineteenth International Conference on Software Engineering and Knowledge Engineering (SEKE 2007), 2007, pp. 129–134.

[9] N. E. Fuchs, K. Kaljurand, and T. Kuhn, "Attempto Controlled English for Knowledge Representation," in Lecture Notes in Computer Science, vol. 5224, pp. 104–124, 2008.

[10] N. E. Fuchs and R. Schwitter, "Attempto - controlled natural language for requirements specifications," in Proc. of Seventh International Logic Programming Symp. - Workshop on Logic Programming Environments, 1995, pp. 25–32.

[11] M. W. Esser and P. Struss, "Obtaining models for test generation from natural-language-like functional specifications," in Proc. of 18th International Workshop on Principles of Diagnosis, 2007, pp. 75–82.

[12] D. Crystal, A dictionary of linguistics and phonetics, 6th ed. Oxford: Blackwell, 2008.

[13] Cambridge Dictionaries Online, Available at: http://dictionary.cambridge.org/, access date: 10 March 2011.

A Brief Survey on Automatic Integration Test Order Generation

Zhengshan Wang[1,2,3], Bixin Li[1,2], Lulu Wang[1,2], Qiao Li[1,2]
[1]School of Computer Science and Engineering, Southeast University, Nanjing, China
[2]Key Lab of Computer Network & Information Integration (Southeast University), Ministry of Education
[3]Department of Computer Science and Technology, Chuzhou University, Chuzhou, China
Email: {zhshwang,bx.li,wanglulu,qiao}@seu.edu.cn

Abstract

A common problem in object-oriented software integration testing is to determine the order in which classes are integrated and tested. In this paper, we first overview some related work based on their objectives in current literature and then provide some analysis and evaluation.

***Index Terms*—object-oriented; integration test order; coupling measurement; graph-based algorithm; search-based algorithm**

I. Introduction

During object-oriented software integration testing, an important problem is to determine the order in which classes are integrated and tested, which is called class integration test order (*CITO*) problem. When integrating and testing a class that depends on other classes that have not been developed or tested, some stubs must be developed to simulate these other classes. There are two types of stubs: specific stub and realistic stub [16] (or generic stub [9]). It is very error-prone and costly to construct stubs. All related papers tried to find a (optimal) test order using minimal stubbing cost which is usually modeled as the number of stubs to be constructed or modeled as the overall stub complexity.

Many approaches proposed to solve this problem can be divided into two categories based on their objectives: minimizing the number of stubs [2], [5], [7], [11], [12], [13], [14], [15], [16] and minimizing the overall stub complexity [1], [3], [4], [6], [17]. If the objective is to minimize the number of stubs, there are two key steps which are modeling and breaking cycles. A dependency model which can be created from *UML* diagrams or source code is used to represent classes and inter-class dependency relationships using nodes and edges, respectively. In current literature, there are two kinds of dependency models which are object relationship diagram (*ORD*) [12] and test dependency graph (*TDG*) [16]. If a dependency model has no cycles, a test order can be generated by using a simple reverse topological sorting algorithm based on inter-class dependency relationships, otherwise all existing approaches tried to design an effective and efficient algorithm to break cycles. In current literature, there are two types of typical algorithms for breaking cycles which are graph-based algorithm (*GBA*) and search-based algorithm (*SBA*). If the objective is to minimize the overall stub complexity, one extra step must be introduced to measure each stub complexity based on inter-class coupling information before breaking cycles.

The rest of this paper is organized as follows: Section II overviews some existing approaches for minimizing the number of stubs. Section III overviews some existing approaches for minimizing the overall stub complexity. Section IV provides some analysis and evaluation. Section V draws some conclusions.

II. Minimizing the Number of Stubs

A. Minimizing the Number of Specific Stubs

Kung et al. [12] used an *ORD* as a dependency model which was created from source code and had three types of dependency relationships: inheritance, aggregation and association. They used a *GBA* to break cycles by selecting and removing association edges. Their algorithm did not use any heuristic information to select association edges

Supported partially by the National Natural Science Foundation of China under Grant No. 60773105 and no. 60973149, and partially Supported by Doctoral Fund of Ministry of Education of China under Grant No. 20100092110022, partially by National High Technology Research and Development Program under Grant No.2008AA01Z113, and partially by the Natural Science Foundation of Chuzhou University under Grant No.2010KJ019B.

Correspondence to: bx.li@seu.edu.cn

to break cycles. The time complexity of their algorithm is O(n^4).

Tai et al. [15] used an *ORD* as a dependency model which was created from *UML* class diagrams and had three kinds of dependency relationships: inheritance, aggregation and association. They determined major level numbers of classes based on inheritance edges and aggregation edges and determined minor level numbers of classes based on association edges. If the classes with the same major level number contained cycles, they used a *GBA* to break cycles. Their algorithm preferred to select and remove an association edge (u,v) with the highest weight to break cycles. The weight was defined as the sum of in-degree of u and out-degree of v. The time complexity of their algorithm is O(n^4).

Briand et al. [5] used an *ORD* as a dependency model which was created from *UML* class diagrams and had five kinds of dependency relationships: inheritance, aggregation, association, composition and usage. They used a *GBA* to break cycles. Their algorithm preferred to select and remove an association edge (u,v) with the highest weight to break cycles. The weight was defined as the product of in-degree of u and out-degree of v. The time complexity of their algorithm is O(n^4).

Malloy et al. [13] and Kraft et al. [11] used an *ORD* as a dependency model which was created from source code and had six types of dependency edges: inheritance, composition, association, dependency, polymorphism, and ownedElement. They used trial-and-error method to assign different weights to different edges based on their types and then used a *GBA* to break cycles. Their algorithm preferred to select and remove an edge with the lowest weight to break cycles. The time complexity of their algorithm is O(n^4).

Mao et al. [14] used a weighted *ORD* (*WORD*) as a dependency model which was created from *UML* class diagrams and had three types of dependency relationships: inheritance, aggregation and association. They used a triple (cycle weight, direction factors of edges, association intensity) to represent weight for each association edge. They defined the cycle whose length is two as a cycle pair. They first selected and removed association edges to break all cycle pairs, then used a *GBA* to break other types of cycles. Their algorithm preferred to select and remove an association edge with the highest cycle weight to break cycles. The time complexity of their algorithm is O(n^4).

Hashim et al. [8] used an *ORD* as a dependency model which was created from source code and had five types of dependency edges: inheritance, composition, aggregation, association and dependency. They used coupling between objects as a new cost function instead of Malloy et al's parameterized cost function to assign weights to all types of edges with a deterministic way. They used a *GBA* to break cycles. Their algorithm preferred to select and remove a node with the lowest weight to break cycles. The time complexity of their algorithm is O(n^3).

Bansal et al. [2] used an *ORD* as a dependency model which was created from source code and had eight types of dependency edges: inheritance, composition, association, dependency, polymorphism, ownedElement, friend and exception. They used Malloy et al.'s idea [13] to assign weights to all types of edges and used Abdurazik et al.'s method [1] (discussed in Section III) to break cycles.

B. Minimizing the Number of Generic Stubs

Traon et al. [16] used a *TDG* as a dependency model which was created from *UML* diagrams. They used a *GBA* to break cycles. Their algorithm preferred to select and remove a node with the highest weight to break cycles. The node weight was defined as the sum of the number of incoming back edges and the number of outgoing back edges. The time complexity of their algorithm is O(n^3). The number of generic stubs depended on the starting node of deep first search (*DFS*).

Hanh et al. [7] used a *TDG* as a dependency model which was created from *UML* diagrams. They provided two different types of strategies to break cycles. The first called *Triskell* used a *GBA* to break cycles. They preferred to select and remove a node with the highest weight to break cycles. The weight was defined as the number of cycles the node appeared in. The second used a *GA* to break cycles. The time complexity of *Triskell* strategy is non-polynomial because it takes non-polynomial time to enumerate all cycles before assigning weights to nodes.

Hewett et al. used a *TDG* as a dependency model which was created from *UML* class diagrams. In [10], they used a fast algorithm to find a (optimal) test order with minimal generic stubs. In [9], they improved their previous algorithm by adding an additional heuristic information. Two papers all used *GBAs* to break cycles and their time complexity is O(n^2).

III. Minimizing the Overall Stub Complexity

Briand et al. [4] used an *ORD* as a dependency model which was created from *UML* class diagrams and had four types of dependency relationships: inheritance, aggregation, association and usage. They proposed a coupling measurement technique to estimate stub complexity based on inter-class coupling information and used a *GA* to break cycles. They disallowed to remove inheritance edges and composition edges to break cycles.

Abdurazik et al. [1] used a *WORD* as a dependency model which was created from source code and had nine

types of dependency relationships: inheritance, implementation, composition, aggregation, association, dependency, etc. They provided a coupling measurement technique to estimate stub complexity using more fine-grained information and presented three different algorithms to break cycles. Each of them used a *GBA* to break cycles, and preferred to select and remove an edge (node) with the highest cycle-weight ratio (*CWR*). Their coupling measurement technique only simply assigned fixed values to inheritance edges and composition edges. The time complexity of their algorithms is non-polynomial because their ideas are similar to *Triskell*'s proposed by Hanh et al.

Borner et al. [3] used an *ORD* as a dependency model which was created from source code and had three kinds of dependency relationships: inheritance, association and dependency. They believed that test focus should be considered when performing integration testing. They used a simulated annealing algorithm and a genetic algorithm to find a (optimal) test order that considered not only the simulation effort but also the test focus.

Our previous work [17] used an extended *WORD* (*EWORD*) as a dependency model which was created from source code and had six types of dependency relationships: inheritance, implementation, composition, aggregation, association and usage. We proposed a coupling measurement technique to estimate stub complexity for all types of edges and used a random iterative algorithm (*RIA*) to break cycles. Our *RIA* algorithm used some properties of minimal feedback arc set and the idea of simulated annealing, which made it more effective.

Cabral et al. [6] used an *ORD* as a dependency model which was created from source code and had four kinds of dependency relationships: inheritance, composition, association and dependency. They used a multi-objective ant colony algorithm (*MOCA*) to generate a set of *Pareto* optimal solutions that achieved a balance between attribute complexity and method complexity.

IV. Analysis And Evaluation

Table I is a summary table, where the time complexity of all *SBAs* is not provided because their time complexity is related to specific implementation information. From this table, We can obtain some analysis results as follows.

• Published year: Three papers are published from 1995 to 2000, six papers are published from 2001 to 2005 and eight papers are published from 2006 to 2010, which indicates that more and more papers give attention to *CITO* problem.

• Objective: Though only five papers try to minimize the overall stub complexity, four of them are published from 2009 to 2010, which indicates that finding a (optimal) test order with minimal overall stub complexity is current hot topic. The reason is that minimizing the overall stub complexity is more reasonable than minimizing the number of stubs.

• Model: Seventeen papers are listed. Ten of them create their dependency models from source code and six of them are published from 2006 to 2010, which indicates that *CITO* problem is given more attention in reverse engineering research field. The reason is that source code can provide more fine-grained information.

• Algorithm type: Though five papers use search-based algorithms to break cycles, three of them are published in 2009 and 2010, which indicates that based-search algorithms have more utilization potentiality than based-graph algorithms. The reason is that based-search algorithms can escape from a local optimal solution and can be used to perform multi-objective optimization.

• Time complexity: Two papers are published in 2008 and 2009 whose time complexity is O(n^2), which indicates that designing a fast algorithm to break cycle is given more attention in recent years. The reason is that some software systems have a large number of cycles to be broken.

• Algorithm constraint condition: From 2006 to 2010, seven papers allow to remove inheritance edges and composition edges to break cycles and only one paper does not, which indicates that it is acceptable to remove inheritance edges and composition edges to break cycles in recent years.

Some challenging problems encountered are identified and explained as follows.

• The models proposed for representing inter-class dependencies lack precision. An *ORD* and a *TDG* can represent polymorphism dependency relationships, but applying class hierarchy structure to construct these dependency relationships is too simply, because some dynamic dependency edges may be not exist.

• The techniques proposed for measuring inter-class coupling information lack precision. Different stubs may need different test effort to be constructed, so it is more reasonable to find a (optimal) test order with minimal overall stub complexity than minimal number of stubs. The number of accessed attributes and the number of called methods are usually used as inter-class coupling information to calculate stub complexity, but it is not sufficient to use these information to calculate stub complexity.

• The algorithms proposed for breaking cycles lack effectiveness and efficiency. For a large-scale application case, it is very difficult to design an effective and efficient algorithm to break cycles. Graph-based algorithms are usually very fast but they have no chance to escape from a local optimal solution. On the contrary, search-based algorithms are usually very slow and require to adjust a lot of parameters to improve their performance, but they have chances to escape from a local optimal solution.

TABLE I. Analysis Results of Related Works

No.	Reference	Year	Objective	Model			Algorithm		
				Type	Edge Types	Origin	Type	Time Comp.	Constraint
1	Kung [12]	1995	MNSS	ORD	In,Ag,As	Code	GBA	$O(n^4)$	Yes
2	Tai [15]	1997	MNSS	ORD	In,Ag,As	UML	GBA	$O(n^4)$	Yes
3	Traon [16]	2000	MNGS	TDG	-	UML	GBA	$O(n^3)$	No
4	Hanh [7]	2001	MNGS	TDG	-	UML	GBA,GA	$\Omega(2^n)$, -	No
5	Briand [4]	2002	MOSC	ORD	In,Ag,As,De	Code	GA	-	Yes
6	Briand [5]	2003	MNSS	ORD	In,Co,Ag,As,De	Code	GBA	$O(n^4)$	Yes
7	Malloy [13]	2003	MNSS	ORD	In,Co,As,De,Po,Ow	Code	GBA	$O(n^4)$	No
8	Mao [14]	2005	MNSS	WORD	In,Ag,As	UML	GBA	$O(n^4)$	Yes
9	Hashim [8]	2005	MNSS	ORD	In,Co,Ag,As,De	UML	GBA	$O(n^3)$	No
10	Kraft [11]	2006	MNSS	ORD	In,Co,As,De,Po,Ow	Code	GBA	$O(n^4)$	No
11	Hewett [10]	2008	MNGS	TDG	-	UML	GBA	$O(n^2)$	No
12	Abdur. [1]	2009	MOSC	WORD	In,Im,Co,Ag,As,De,...	Code	GBA	$\Omega(2^n)$	No
13	Hewett [9]	2009	MNGS	TDG	-	UML	GBA	$O(n^2)$	No
14	Bansal [2]	2009	MNSS	ORD	In,Co,As,De,Po,Ow,...	Code	GBA	$\Omega(2^n)$	No
15	Borner [3]	2009	MOSC	ORD	In,As,De	Code	SA,GA	- , -	No
16	Wang [17]	2010	MOSC	EWORD	In,Im,Co,Ag,As,De	Code	RIA	-	No
17	Cabral [6]	2010	MOSC	ORD	In,Co,As,De	Code	ACO	-	Yes

MNS(G)S: minimizing the number of specific (generic) stubs; MOSC: minimizing the overall stub complexity.
In: inheritance; Im: implementation; Co: composition; Ag: aggregation;
As: association; De: dependency; Po: polymorphism; Ow: owenElement.
GBA: graph-based algorithm; SA: simulated annealing algorithm; GA: genetic algorithm;
RIA: random iterative algorithm; ACO: ant colony optimization algorithm.
Constraint represents whether or not inheritance and composition edges are allowed to be removed to break cycles.

V. Conclusions

In this paper, we first review some related work based on their objectives and then provide some analysis and evaluation. By analysis, it is found to be a good strategy for automatic integration test order generation to construct a more precise dependency model from source code, to calculate stub complexity using a more precise coupling measure technique, and to design an effective and efficient algorithm for breaking cycles.

References

[1] A. Abdurazik and J. Offutt. Using coupling-based weights for the class integration and test order problem. *The Computer Journal*, 52(5):557–570, 2009.

[2] P. Bansal, S. Sabharwal, and P. Sidhu. An investigation of strategies for finding test order during integration testing of object oriented applications. In *Proceeding of International Conference on Methods and Models in Computer Science*, pages 1–8, 2009.

[3] L. Borner and B. Paech. Integration test order strategies to consider test focus and simulation effort. In *Proceeding of International ConferencFe on Advances in System Testing and Validation Lifecycle*, pages 80–85, 2009.

[4] L. C. Briand, J. Feng, and Y. Labiche. Using genetic algorithms and coupling measures to devise optimal integration test orders. In *Proceedings of 14th International Conference in Software Engineering and Knowledge Engineering*, pages 43–50, 2002.

[5] L. C. Briand, Y. Labiche, and Y. Wang. An investigation of graph-based class integration test order strategies. *IEEE Transaction on Software Engineering*, 29(7):594–607, 2003.

[6] R. da Veiga Cabral, A. Pozo, and S. R. Vergilio. A pareto ant colony algorithm applied to the class integration and test order problem. In *Proceedings of the 22nd IFIP WG 6.1 international conference on Testing software and systems*, pages 16–29, 2010.

[7] V. L. Hanh, K. Akif, Y. L. Traon, and J.-M. Jézéquel. Selecting an efficient OO integration testing strategy: An experimental comparison of actual strategies. In *Proceedings of the 15th European Conference on Object-Oriented Programming*, pages 381–401, 2001.

[8] N. L. Hashim, H. W. Schmidt, and S. Ramakrishnan. Test order for class-based integration testing of java applications. In *Proceedings of the Fifth International Conference on Quality Software*, pages 11–18, 2005.

[9] R. Hewett and P. Kijsanayothin. Automated test order generation for software component integration testing. In *Proceedings of ASE 2009*, pages 211–220, 2009.

[10] R. Hewett, P. Kijsanayothin, and D. Smavatkul. Test order generation for efficient object-oriented class integration testing. In *Proceedings of SEKE2008*, pages 703–708, 2008.

[11] N. A. Kraft, E. L. Lloyd, B. A. Malloy, and P. J. Clarke. The implementation of an extensible system for comparison and visualization of class ordering methodologies. *Journal of System Software*, 79:1092–1109, 2006.

[12] D. C. Kung, J. Gao, and P. Hsia. Class firewall test order and regression testing of object oriented programs. *Journal of Object-Oriented Programming*, 8(2):51–65, 1995.

[13] B. A. Malloy, P. J. Clarke, and E. L. Lloyd. A parameterized cost model to order classes for class-based testing of c++ applications. In *Proceedings of 14th International Symposium Software Reliability Engineering*, pages 353–364, 2003.

[14] C. Mao and Y. Lu. Aicto: An improved algorithm for planning inter-class test order. In *Proceedings of the Fifth International Conference on Computer and Information Technology*, pages 927–931, 2005.

[15] K.-C. Tai and F. J.Daniels. Interclass test order for object-oriented software. In *Proceedings of the 21st International Computer Software and Applications Conference*, pages 602–607, 1997.

[16] Y. L. Traon, T. Jéron, J.-M. Jézéquel, and P. Morel. Efficient object-oriented integration and regression testing. *IEEE Transaction on Reliability*, 49(1):12–25, 2000.

[17] Z. Wang, B. Li, L. Wang, M. Wang, and X. Gong. Using coupling measure technique and random iterative algorithm for inter-class integration test order problem. In *Proceedings of 34th International Computer Software and Applications Conference Workshop*, pages 329–334, 2010.

Generation of Scripts for Performance Testing Based on UML Models

Maicon B. da Silveira, Elder M. Rodrigues, Avelino F. Zorzo,
Leandro T. Costa, Hugo V. Vieira and Flávio M. de Oliveira
Faculty of Informatics (FACIN) – Pontifical Catholic University of Rio Grande do Sul (PUCRS)
Porto Alegre – RS, Brazil
bernardino@acm.org, elder.rodrigues@acad.pucrs.br, avelino.zorzo@pucrs.br,
leandro.teodoro@acad.pucrs.br, hugovares@gmail.com and flavio.oliveira@pucrs.br

***Abstract*—Software testing process has a high cost when compared to the other stages of software development. Automation of software testing through reuse of software artifacts (e.g. models) is a good alternative for mitigating these costs and making the process much more efficient and efficacious. Model-Based Testing (MBT) is a technique to automatic generation of testing artifacts based on software models. For software development, the most spread modeling language in either the industrial or academic environments is UML. In such environments, it is desirable to reuse UML models also for MBT, avoiding re-building a different model exclusively for testing automation. These are the main reasons that make these semi-formal models an alternative to implementing MBT. Even though there are a lot of testing tools available commercially, to the best of our knowledge, none of them fully uses MBT. Therefore, this paper describes a case study showing how to implement the MBT process to automate test scripts generation and execution in a real-world, context. Furthermore, our solution is generated automatically by a Software Product Line (SPL).** [1]

***Keywords* - Model-Based Testing; Software Product Line; Performance Testing.**

I. Introduction

Currently, a great number of people and companies use computer programs to automate their activities, delegating to systems the execution of complex tasks. This widespread use of computer systems has also increased the number of residual software or hardware faults that generate failures to users [1] [2]. Therefore, it is important that during the development of a system, different techniques are applied to guarantee that the system provides a service that can be trusted. The ability to deliver a service that can justifiably be trusted is known as dependability [1]. The main attributes that integrate the dependability are reliability, availability, security, confidentiality, integrity and maintainability. According to the taxonomy presented in [1], system dependability can be achieved by four techniques: 1) Fault Prevention - *prevent the occurrence or introduction of faults*; 2) Fault Tolerance - *avoid service failures in the presence of faults*; 3) Fault Removal - *reduce the number and severity of faults*, and; 4) Fault Forecasting - *to estimate the present number, the future incidence, and the likely consequences of faults*. Several works that provide system dependability through fault tolerance, fault prevention and fault forecasting are present in the literature [3] [4] [5].

Although all techniques are used to achieve software dependability, the most used technique in all areas of software development in industry is fault removal, through software testing. Software testing is a process that focus on finding program failures[2] during runtime [6], or that has activities to validate the requirements of a program, determining if the expected results are met [7]. Performance is a key component of reliability and availability; therefore, performance testing is a major activity in system fault removal. However, due to the systems evolution and their amount of features, systems are becoming so complex that testing them is a difficult task. Therefore, it is necessary to implement a testing process to mitigate testing execution on the final product. This process should aim to reduce the costs impact, improving the quality of the software product [2].

One of the techniques that improves the software testing process is Model-Based Testing (MBT) [8]. This technique consists in the generation of test cases and/or test scripts based on the application model. Besides, it also includes the specification of the features that will be tested [9].

In previous work [10] [11], we have used MBT to build testing tools for security and functional testing. These tools were derived products from a Software Product Line [12] called PL*e*T*s* [11]. The work presented in this paper expands our previous work to apply MBT in the generation of a new product to execute performance testing. This new product generates test scripts for a commercial tool called LoadRunner [13]. Basically, we include stereotypes in the system UML models to express performance information that will be used during the execution of test scripts.

This paper is organized as follows. Section II presents a short description of MBT, SPL and the PL*e*T*s* architecture. In Section III we show how we have used stereotypes to include information on the UML models and a brief description of the LoadRunner tool. In Section IV we apply our strategy to an actual case study that uses the LoadRunner tool to execute performance testing. This case study is a tool used in a major

[1]Study developed by the Research Group of the PDTI 001/2011, financed by Dell Computers of Brazil Ltd. with resources of Law 8.248/91.

[2]We use fault, error, and failure definition from [1].

IT company. Finally, Section V summarizes the contributions of our work.

II. Background

Software modeling is an important technique that is used in software development because it allows to capture and to share knowledge about a system. During the development process, information about the system is described in many different documents. To include this information, through, for example, UML stereotypes, enriches the specification documents. This increases the quality of the specification and the use of models developed as the system evolves [14]. Thus, this information is used to model the incremental creation of new artifacts, or even allows the automation of other processes to improve the quality of different developed artifacts. This section describes two approaches to automate software artifacts development based on features that are included in the system model: Model-Based Testing and Software Product Line.

A. Model-Based Testing

Usually, system behavior and requirements are described using a formal or semi-formal model, allowing team members to share and to use these models during the life-cycle of the software development. In spite of that, test engineers are still producing test cases or test scripts in an informal way based on a mental representation that they create. A better alternative would be to derive test artifacts from the system models [15]. This strategy is commonly known as Model-Based Testing (MBT) [8]. MBT is based on the idea of the automation of test case/scripts generation. Albeit, the use of existing models can increase productivity during the process of software testing [14].

The cost of software testing is related to the number of interactions and test cases that are executed during the development process. As it is one of the most costly and expensive phases of software development [6], MBT is a good approach to mitigate this problem by automating the process of generating test cases or scripts [16].

Several works on MBT have been produced in the past years. A systematic review is presented in [17] to evaluate quantitatively and qualitatively some features of MBT. In this study, 78 articles were evaluated and the following items were reviewed: type of model (formal or semi-formal) used for test generation; test types in which the approach can be applied to; level of automation; support tools; criteria for test coverage, etc. The study presents test domains (system, integration, unit/component, regression) in which each of the works were applied to. The survey identifies that most of the works use MBT in system testing domain (66%). System testing includes Performance testing, which is the focus of our work.

Some of the works mentioned in [17] use the same MBT process, for example [8] [18]. This process requires specific activities in addition to the usual activities of software testing. This will require that the test engineers adjust their testing process, and invest in the use and training of new tools. The main activities that define the MBT process are (see

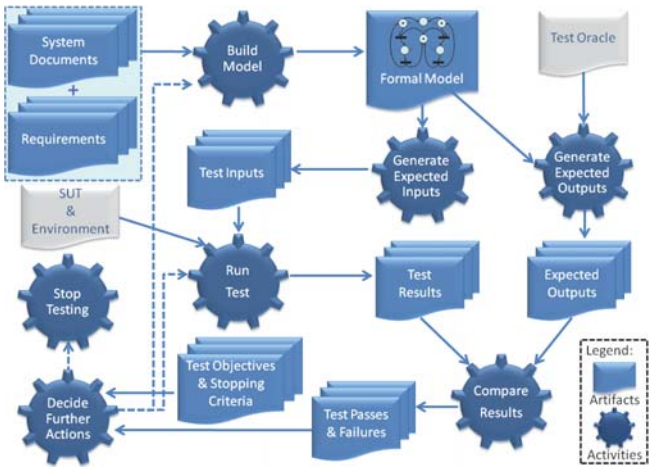


Fig. 1. Activities for MBT [8]

Figure 1) [8]: *Build Model*, *Generate Expected Inputs*, *Generate Expected Outputs*, *Run Tests*, *Compare Results*, *Decide Further Actions* and *Stop Testing*. 1) *Build Model*: constructs a model based on the specification of system requirements. This step defines the choice of the model, according to the application being developed; 2) *Generate Expected Inputs*: uses the developed model to generate test inputs (test cases, test scripts, application input); 3) *Generate Expected Outputs*: generates some mechanism that determines whether the results of a test execution are correct or not. This mechanism is the test oracle and it is used to determine the correctness of the output; 4) *Run Tests*: executes test scripts and stores the processing results of each test case. This execution can be performed in the system under test (SUT) and/or system's environment; 5) *Compare Results*: compares the test results with expected outputs (test oracle), generating reports to alert the test team about failures; 6) *Decide Further Actions*: based on the results, it is possible to estimate the software quality. Depending on the quality achieved, it is possible to stop testing (quality achieved), to modify the model to include further information to generate new inputs/outputs, to modify the system under test (to remove remaining faults), or to run more tests; 7) *Stop Testing*: concludes the system testing. The activities from MBT process can bring several new advantages to the test team [8], for example: shorter schedules, lower cost, and better quality.

A good possibility to reduce the problems mentioned at the beginning of the previous paragraph would be to have a single tool that would cover all the phases of the MBT process, i.e. a tool in which it would be possible to describe the system model, that would generate test cases/scripts, that would execute the test scripts and also compare the results. Even better if the test team could have a tool that could generate the testing tool for each different application or different type of test the test team wants to execute over the same application. Furthermore, it is desirable that the test team reuse implemented artifacts (e.g.: models, software components, scripts). Thus, in this context, it becomes interesting

to design a set of MBT tools based on a Software Product Line (SPL). A SPL ensures the variability, reusability of test artifacts, thus decreasing costs and time to market. Several evidence of the benefits of the use of SPL, in different areas, can be found in [19] [20]. The next section introduces the concept of SPL.

B. Software Product Line

A Software Product Line (SPL) seeks to exploit the commonalities among systems from a given domain, and at the same time to manage the variability among them [12]. According to [21], SPL engineering has three main concepts: core assets development, product development, and management. The core assets are the main part of an SPL, and its components aim to represent, in a clear way, the common and variable aspects of the future products. Thus, following the SPL concepts, new products variants can be quickly created based on a common architecture, models, software components, etc. Because of that, SPL allows for rapid entry of a product on the market as well as makes it easier for mass customization of products of a company. Companies are finding that the practice of building sets of related systems from common assets can, in fact, produce quantitative improvements in product quality and consumer satisfaction, efficiently meeting the current demand for mass customization.

However, given the large number of products that can be present in a product line (PL), it is necessary to control the variability and commonalities among them. The variability management is used to control the variables aspects present in the products of the PL.

Feature Models is an important concept to modeling variability. Originally, Feature Modeling was developed as part of Feature-Oriented Domain Analysis (FODA) [22]. However, nowadays it is applied in many areas such as embedded systems [23] or networks protocols [24].

When Feature Models are applied to represent variability, they are analyzed and categorized as common, optional or alternative [25]. *Common features* represent features that must be present in every product of the SPL. There are also called mandatory, necessary, or kernel features. *Optional features* represent features that are supported by some products in the SPL, and; the *alternative features* represent features that are mutually exclusive, i.e. only one of the features can be provided in each product of the SPL (see Figure 2).

C. PLeTs Tool

The PL*e*T*s* tool [11] aims to automate the generation, execution and results collection of MBT process. The tool is able to manage the whole MBT process and is based on the concepts of SPL. Its goal is not only the reuse of artifacts to make it easier and faster to develop a new tool of the family, but also to improve the creation, run and gathering of test results. It was developed with the intent to be used by software engineers, developers and test engineers, assisting the process of defining and executing test cases and test scripts. Figure 2 shows the current PL*e*T*s* Feature Model that represents some of the features that could be present in a software variant. The first level of the model has four main features:

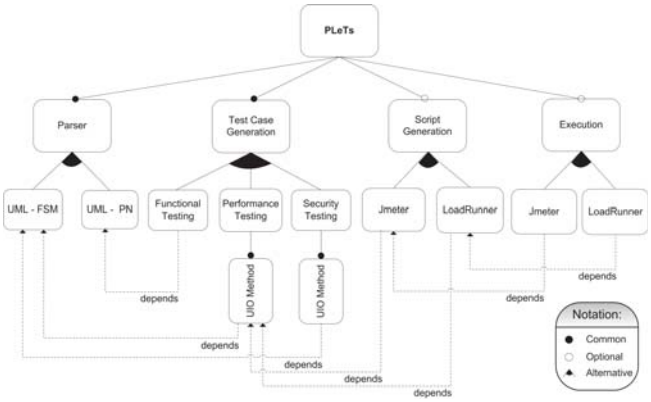


Fig. 2. Feature Model for PL*e*T*s* Tool [11]

1) *Parser*: represents the *Build Model* step in the MBT main activities (see Section II-A). It is a mandatory feature and has two child features: UML - FSM and UML - PN. Each one of these parsers is used to extract the information from the UML models to generate a formal model (Finite State Machine (FSM) or Petri Nets (PN)); 2) *Test Case Generation*: represents part of the *Generate Expected Inputs* step in the MBT main activities. It is a mandatory feature and has three child features: Functional Testing, Performance Testing and Security Testing (one of them should be selected in each software variant). Both Performance Testing and Security Testing have a mandatory child feature: UIO Method [26]; 3) *Script Generation*: represents another part of the *Generate Expected Inputs* step in the MBT main activities. It is an optional feature, because, for example, for security testing there could be no tool to execute the generated test cases. This feature has two child features: Jmeter [27] and LoadRunner [13]; 4) *Execution*: represent the *Run Tests and Compare Results* step in the MBT main activities. This feature also has two child features: Jmeter and LoadRunner.

It is important to highlight that there are dependencies between some features (see the dotted lines in Figure 2). For example, if some software variant selects the feature *Execution* and the child feature *LoadRunner*, it must select the feature *Script Generation* and the child feature *LoadRunner*, because the tool is not able to execute the tests without a test script.

Another important point is that the Feature Model can be extended to support new testing techniques or tools, adding new child features to the main four features. For example, if someone wants to add new features to work with the SilkPerformer tool [28], he should include new child features for the *Script Generation* and *Execution* main features.

To develop the tool in a way to represent the feature model flexibility, the PL*e*T*s* architecture is based on plug-ins that allows extensibility and flexibility. Based on that, the tool allows to select, in runtime, each plug-in (represented by a feature) that is necessary to perform a MBT activity and to automatically generate/execute the test scripts.

III. UML Models for Performance Testing

In previous works [10] [11], we described some components of PLeTs developed for security and functional testing. Here we apply our approach in a different application domain, reusing, or expanding, previous components automatically through SPL.

In our approach, the starting point for test script generation is the construction of an UML model activity diagram with performance stereotypes[3], represented in an XMI file. This XMI file is parsed and converted into a formal model, e.g. Finite State Machine (FSM). Then, performing the UIO Method [26], the sequences of activities that have to be executed are obtained. These sequences could be transformed in a description that is equivalent to test cases in natural language. This approach was used in our previous works, i.e. all these features have already been developed. To expand our work, we propose here a set of new features to support the generation of a product that can perform performance testing for the LoadRunner tool [13].

As mentioned above, we use stereotypes, which are the way we describe performance information necessary to generate our test cases and test scripts. We include stereotypes in two UML diagrams: use case and activity. Our approach uses four stereotypes from our previous works and a new one that was missing. The five performance stereotypes are the following: 1) <<PApopulation>>: this stereotype has two tags: the first one represents the number of users that are running the application, while the second one represents the host where the application is executed (defined in all actors of the use cases diagram); 2) <<PAprob>>: defines the probability of execution for each existing activity; 3) <<PAtime>>: expected time to perform a given use case; 4) <<PAthinkTime>>: denotes the time between the moment the activity becomes available to the user and the moment the user decides to execute it, for example, the time for filling a form before its submission; 5) <<PAparameters>>: defines the tags for the input data that will be provided to the application when running the test scripts (this is a new stereotype that our previous works did not include).

A. LoadRunner

HP LoadRunner [13] is a product to analyze the behavior and the performance of systems. This tool can emulate hundreds, or thousands, of users, known as Virtual Users (VUsers), simultaneously.

In the LoadRunner architecture the main configuration part that has to be changed for each application that is tested is stored in the script folder. Basically, our PLeTs plug-in generates a new *script* file, that are scripts written in C language. The test description, which includes application transactions and parameters, is included in that file. Our PLeTs plug-in also generates the configuration scenarios, which are included in *scenarios* file. This file contains performance counters that will be used in the reports that are generated by the LoadRunner.

[3]We use UML 2.0 SPT (Schedulability, Performance and Time) Profile [29] [30].

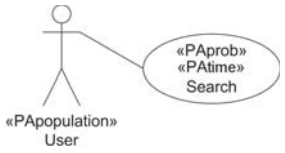


Fig. 3. Skill Management Use Case Diagram

Another important LoadRunner feature is a library with predefined functions. Each set of functions that are included in this library have a specific task in each of the testing protocols that LoadRunner implements. For example, functions starting with `web` are used to represent HTTP requests, while the ones starting with `lr` are general and can be used in all protocols. Some of the existing functions, which will be used in the work described in this paper, are: 1) lr_think_time: determines the idle time between user interactions and the system; 2) web_submit_data: submits web form data without a previous operation context; 3) web_url: is responsible for accessing a URL via a web browser; 4) web_image: represents a click on an image on a page (tag HTML <IMG>); 5) web_link: represents a click on a text link on a page (tag HTML <A href=...> e </A>); 6) web_submit_form: submits web form data but considering the context of the previous operation.

Based on these functions, it is possible to simulate the test scenarios in order to verify the performance behavior of a given application. Moreover, the concepts discussed in this section (tool architecture, script features and functionalities interpreted by LoadRunner), were important to implement the automatic scripts generation based on information extracted from the UML model.

IV. Case Study: Skills Management Tool

In this section, we apply our strategy and the PLeTs tool to an application that manage skills, certifications and experience of employees of a given organization. This tool is called Skills and was developed in collaboration between a research group of our institution and an IT company. Skills was developed in the Java programming language, using the MySQL database for data persistence and Tomcat as web application server.

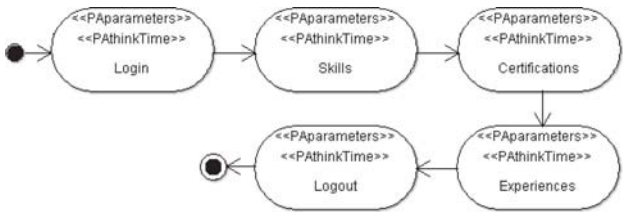


Fig. 4. Skill Management Activity Diagram

One example of our use of UML with stereotypes is the "Search" case. Figure 3 shows part of the user interaction behavior with the application. Furthermore, the necessary steps

to implement this case study are detailed in the activity diagram shown in Figure 4. This diagram represents five sequential activities, starting with "Login" to access the system, "Skills" to consult the user's abilities, "Certifications" to view the technical certification assigned to the user; "Experiences" a list the user's professional experience; and "Logout" to exit the system.

Once all the UML diagrams (e.g. see Figures 3 and 4) have been constructed, we use PLeTs to generate the test scripts for the Skill Management Application. The scripts were generated to run on LoadRunner, but we could have changed the plug-in that generates scripts and apply the same set of test using a different testing tool, for example the IBM Rational Performance Tester [31]. Figure 5 shows an extract from the script that was generated. This extract shows the actions of a given user, for example the think time through function `lr_think_time(10)` and the submission of data filled in the step "Login" (from the activity diagram - see Figure 4) through function `web_submit_form`.

```
Action()
{
web_url("Search",
"URL=http://192.168.1.26/skillsApp/mainIE.jsp",
"Resource=0",
"RecContentType=text/html",
"Referer=",
"Mode=HTML",
LAST);

lr_think_time(10);
web_submit_form("Loging.jsp",
ITEMDATA,
"Name=username", "Value=admin", ENDITEM,
"Name=password", "Value=123456", ENDITEM,
LAST);
```

Fig. 5. Script Generated for LoadRunner

As described in Section III, we can include five stereotypes in our UML diagrams, with one or more tags. As can be seen in Figure 3, the "Search" use case diagram has three of those stereotypes, and they are generated with the following values:

- $PApopulation$
  - $TDpath$ = http://192.168.1.26/skillsApp/mainIE.jsp
  - $TDpopulation$ = 30
- $PAprob$
  - $TDprob$ = 1.0
- $PAtime$
  - $TDtime$ = 300

The other stereotypes, in this case study, are included in the activity diagram as shown in Figure 4. For example, the "Login" activity has the following values:

- $PAthinkTime$
  - $TDthinkTime$ = 00:00:10
- $PAparameters$
  - $TDaction$ = login.jsp
  - $TDparameters$ = username@@admin
  - $TDparameters$ = password@@123456

Notice that the `TDparameters` tag is the concatenation of two pieces of information: name and value, separated by the delimiter @@. This information could be generated automatically for different scenarios that the test engineer wants to test.

All tags are extracted from the UML diagram and processed by the PLeTs plug-in that generates scripts for LoadRunner. Although all necessary information is included in the UML diagrams, LoadRunner also needs a description for the test scenarios. The configuration of the test scenarios is included in the *scenarios* file. The PLeTs plug-in uses a template to automatically generate the *scenarios* file for LoadRunner. This template has some markings that are replace by the tags from the UML diagrams. For example, tag `<<Vusers>>` represents the number of virtual users that will be simulated during the test.

Besides the `<<Vusers>>` marking, the template file for scenarios has other markings that are used when generating test scripts: 1) `<<Path>>`: contains the application host address; 2) `<<Result_file>>`: specifies the results file name; 3) `<<HostGenerator>>`: defines the server that generates the load; 4) `<<TestChief>>`: contains the paths for the script tests that will be run for the scenario; 5) `<<GroupChief>>`: stores information on each VUsers group that will be simulated by the test script; 6) `<<GroupInfo>>`: defines the performance counters that will be used during the test.

Once the test script generation is completed, PLeTs calls the LoadRunner tool passing as a parameter the test script and scenario produced. In the test we have performed, we used the LoadRunner standard performance counters that are already set in the default user interface. We could have redefined the screens and counters we wanted to verify, but for the Skill Management application this was not necessary.

V. Conclusion

The use of formal models is a good way for modeling the behavior and structure of a system. This allows for a precise understanding of the system behaviour by software developers or test engineers and, therefore, allows for a better reuse of the system components as the system evolves.

Although automatization of software testing is desirable, test engineers usually perform testing manually using, of course, a set of tools. Capture-replay tools, for example, are widely used, but they require a full, running build of the application in order to create the test scripts for the first time, thus delaying script development; moreover, they require the tester to execute (manually!) all the script al least once. Besides, test engineers have to build a mental model of the whole testing process or strategy. As shown in this paper, MBT can help test engineers to build a formal model for their testing process as well. It is important to mention that despite all the advantages of using MBT described in this paper, MBT requires a professional with skills in programming languages, software testing and modeling, and also with theoretical background

on mathematics, automata theory, graph theory and formal languages [8].

Furthermore, the use of MBT could require a significant investment if not supported by a set of tools to plan, monitor and produce test artifacts. This paper has shown how we have applied MBT to an application without having to start the whole process from scratch, hence helping test engineers that do not have the above mentioned skills or background. Our approach is based on an SPL tool that generates testing tools based on MBT. In some previous works we had already applied our strategy to different testing domains and we were able to reuse some already developed testing artifacts in the work presented here. This has reduced our costs to test an actual application that is used in an IT company.

Another important contribution of this paper is the development of a tool to allow the use of MBT in commercial tools, such as the LoadRunner. Currently there are none, to the best of our knowledge, commercial tool that uses MBT. The LoadRunner plug-in developed for the PL*e*T*s* tool shows that the use of MBT is viable in industry. Besides, the team of the IT company can now use the same plug-in for different applications they want to test.

Despite the advantages of the implementation of the new plug-in presented here, some improvements in this new plug-in could be adopted, for example, in the current plug-in implementation several data that are needed for the generation of the LoadRunner scripts are inserted in the model stereotypes (See Section IV). A simple modification could be the insertion, in the `TDparameters` tag, of a name of a file, or even a database, that would contain these data. The plug-in, then, would generate scripts based on the data stored in that file, or database. These improvements will allow us to use parts of the plug-in for different commercial tools and to further explore our strategy.

ACKNOWLEDGMENT

We also thank CNPq/Brazil, CAPES/Brazil and INCT-SEC for the support in the development of this work. We also thank the reviewers for their comments, which helped to improve our paper.

REFERENCES

[1] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic Concepts and Taxonomy of Dependable and Secure Computing," *IEEE Transaction Dependable Secure Computing*, vol. 1, no. 1, pp. 11–33, 2004.

[2] M. Young and M. Pezzè, *Software Testing and Analysis: Process, Principles and Techniques*. John Wiley & Sons, 2005.

[3] J.-C. Laprie, "Dependability Evaluation of Software Systems in Operation," *IEEE Transactions on Software Engineering*, vol. SE-10, no. 6, pp. 701–714, 2009.

[4] D. Powell, "Failure mode assumptions and assumption coverage," *22nd International Symposium on Fault-Tolerant Computing. Digest of Papers.*, pp. 386–395, 2002.

[5] A. Romanovsky and A. F. Zorzo, "Coordinated atomic actions as a technique for implementing distributed gamma computation," *Journal System Architecture*, vol. 45, no. 15, pp. 1357–1374, 1999.

[6] G. J. Myers and C. Sandler, *The Art of Software Testing*. New York: John Wiley & Sons, 2004.

[7] B. Beizer, *Software System Testing and Quality Assurance*. New York: Van Nostrand Reinhold, 1984.

[8] I. K. El-Far and J. A. Whittaker, *Model-based Software Testing*. New York: Wiley, 2001.

[9] M. Popovic and I. Velikic, "A Generic Model-Based Test Case Generator," *12th IEEE International Conference and Workshops on Engineering of Computer-Based Systems*, pp. 221–228, 2005.

[10] K. P. Peralta, A. M. Orozco, A. F. Zorzo, and F. M. Oliveira, "Specifying Security Aspects in UML Models," *1st International Workshop on Modeling Security In ACM/IEEE 11th International Conference on Model-Driven Engineering Languages and Systems*, pp. 1–10, 2008.

[11] E. de M. Rodrigues, L. D. Viccari, A. F. Zorzo, and I. M. Gimenes, "PLeTs-Test Automation using Software Product Lines and Model Based Testing," *22th International Conference on Software Engineering and Knowledge Engineering*, pp. 483–488, jul. 2010.

[12] P. Clements, L. Northrop, and L. M. Northrop, *Software product lines: practices and patterns*. Addison-Wesley Longman Publishing, 2001.

[13] Hewlett Packard - HP, "*Software* HP LoadRunner," Available in: https://h10078.www1.hp.com/cda/hpms/display/main/hpms_content.jsp?zn= bto&cp=1-11-126-178̂_4000_100, sep. 2010.

[14] L. Apfelbaum and J. Doyle, "Model Based Testing," *Software Quality Week Conference*, 1997.

[15] M. Broy, B. Jonsson, J.-P. Katoen, M. Leucker, and A. Pretschner, *Model-Based Testing of Reactive Systems: Advanced Lectures*. Secaucus: Springer, 2005.

[16] M. Veanes, C. Campbell, W. Grieskamp, W. Schulte, N. Tillmann, and L. Nachmanson, "Model-Based Testing of Object-Oriented Reactive Systems with Spec Explorer," *Formal Methods and Testing*, 2008.

[17] A. C. Dias Neto, R. Subramanyan, M. Vieira, and G. H. Travassos, "A Survey on Model-Based Testing Approaches: A Systematic Review," *1st ACM International Workshop on Empirical Assessment of Software Engineering Languages and Technologies*, pp. 31–36, 2007.

[18] M. Utting, A. Pretschner, and B. Legeard, "A taxonomy of model-based testing," Working Paper, The University of Waikato, Hamilton, New Zealand, Tech. Rep., apr. 2006.

[19] M. Steger, C. Tischer, B. Boss, A. Müller, O. Pertler, W. Stolz, and S. Ferber, "Introducing PLA at Bosch Gasoline Systems: Experiences and Practices," *3rd International Conference on Software Product Lines*, vol. 3154, pp. 34–50, 2004.

[20] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 2nd ed. Addison-Wesley Longman, 2003.

[21] Software Engineering Institute (SEI), "Software Product Lines (SPL)," Available in: http://www.sei.cmu.edu/productlines/, sep. 2010.

[22] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature-Oriented Domain Analysis (FODA) Feasibility Study," Carnegie-Mellon University, SEI, Tech. Rep., nov. 1990.

[23] K. Czarnecki, T. Bednasch, P. Unger, and U. W. Eisenecker, "Generative Programming for Embedded Software: An Industrial Experience Report," *1st ACM SIGPLAN/SIGSOFT Conference on Generative Programming and Component Engineering*, pp. 156–172, 2002.

[24] M. Barbeau and F. Bordeleau, "A Protocol Stack Development Tool Using Generative Programming," *1st ACM SIGPLAN/SIGSOFT Conference on Generative Programming and Component Engineering*, pp. 93–109, 2002.

[25] H. Gomaa, *Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures*. Addison Wesley Longman Publishing, 2004.

[26] R. Anido and A. Cavalli, "Guaranteeing Full Fault Coverage for UIO-Based Testing Methods," *8th International Workshop for Protocol Test Systems*, pp. 221–236, 1995.

[27] Y. Jing, Z. Lan, W. Hongyuan, S. Yuqiang, and C. Guizhen, "JMeter-based aging simulation of computing system," *International Conference on Computer, Mechatronics, Control and Electronic Engineering*, vol. 5, pp. 282–285, aug. 2010.

[28] G. hun Kim, H. choun Moon, G.-P. Song, and S.-K. Shin, "Software Performance Testing Scheme Using Virtualization Technology," *4th International Conference on Ubiquitous Information Technologies Applications*, pp. 1–5, dec. 2009.

[29] OMG, "UML Profile for Schedulability, Performance, and Time Specification - OMG Adopted Specification Version 1.1," Available in: http://www.omg.org/spec/SPTP/1.1/, 2005.

[30] C. Woodside and D. Petriu, "Capabilities of the UML Profile for Schedulability Performance and Time (SPT)," in *Workshop SIVOES-SPT RTAS'2004*, 2004.

[31] D. C. et al., *Using Rational Performance Tester Version 7*. IBM Redbooks, 2008.

How IT Professionals Face Negotiations

Sergio Assis Rodrigues
COPPE/UFRJ - Computer Science Department, Graduate School of Engineering, Federal University of Rio de Janeiro, Rio de Janeiro, Brazil
sergio@cos.ufrj.br

Jano Moreira de Souza
COPPE/UFRJ - Computer Science Department, Graduate School of Engineering, Federal University of Rio de Janeiro, Rio de Janeiro, Brazil
jano@cos.ufrj.br

***Abstract* — Despite the fact that people frequently see software development as a commodity, decision-making processes in Information Technology (IT) projects usually require great effort in solving several types of conflicts. The main tool to solve such conflicts is negotiation, which requires strategies and personal skills. Preparation step is the cornerstone to lead successful deals, but one of the major issues faced by negotiators, especially beginners, is to find out when they are emotionally prepared. Regarding behavioral aspects, there are particularities in IT people's conduction, once they are more reasoning than other professionals. Thus, this work aims at presenting a research about how IT professionals face IT project's negotiations. The research was managed in Brazilian professionals and evaluated IT people's performance through psychological tests, behavioral simulations and negotiation games to set negotiator skills profile. In addition, some case studies and simulation results are presented. This exploration is important as a way to show possible characteristics of style and dominant skills used by IT people during software development deals. This study works as a fundamental starting point to realize the requirements to develop specific negotiation support systems to IT context.**

Keywords - Negotiation, Behavioral Simulations, Psychological Tests, Negotiation Support Systems

I. INTRODUCTION

In order to realize which strategy you should keep during negotiations, it might be helpful to identify some counterparty characteristics. In theoretical and practical researches, a set of specific types have been formed according to a number of characteristics. This work considers four types of negotiator's behavior: i) Logically Reasoning people, ii) Creative people, iii) Communicative people, and iv) Organized people.

A. Logically Reasoning people

There are some key features that a logically reasoning negotiator should possess [1]: a) set the rules of negotiation; b) develop an agenda; c) argue logically; d) adapt the position to meet changing situation; e) be likely to see the process as being more important that content or outcome.

Though the logic behavior of a negotiator may seem easy to be deal with there is a problem which can deter the resolution of negotiations and mislead the opponents. Logical reasoning implies that it is peculiar for the person who is using it, i.e. for one negotiator it may seem obvious to react in a particular way while for his counterpart his behavior is odd or incomprehensible [2]. In the framework of his researches, the author highlights a problem that appears when a logically operating negotiator confusing his opponent with "reasons" that do not address to his logic, which is a situation referred as "argument dilution". Usually, while confirming a viewpoint, people tend to use the strongest arguments in the beginning and the weakest ones are the last to be enumerated. But the listener deliberately marks the last arguments and bases his further opinion upon them. In order to avoid misleading the negotiation to an unsuccessful result, a logical reasoning negotiator should single out only a couple of reasons that give sense and approve his position.

B. Creative people

Another negotiation style should be applied while negotiating with creative people. This type has been given a close examination to by Michael Spangle and Myra Warren Isenhart [3]. They define creative thinkers as those who percept the given information in a unique way and look at the problems from various angles. These people are distinguished from other negotiators by their ability to arrange conventional solutions into more beneficial negotiation`s outcomes. DeBono [4] introduces a concept of "lateral thinking". Those who keep to this principle when managing negotiations tend to open new facets of the situations hidden to others. According to [4] lateral thinking can be expressed in the following actions: a) Looking again at things that are taken for granted (re-examining assumptions); b) Reducing division and polarization on issues, which are natural tendencies of the mind; c)Engaging in deliberate generation of ideas; d) repackaging information in different ways.

Authors compare the problem solving process executed by creative negotiators with completing the "blank spaces" with the missing parts. He argues that being creative negotiators cannot focus on already approved solutions, hence they start being inventive [3].

C. Communicative people

When negotiating, we confront people who are communicative and more reserved counterparts. Communicative people are prone to starting conversations first, responding with eagerness and interest. When it comes to negotiations it is not always easy to say which behavior is more beneficial. According to [5] every person must react in coherence with his personal peculiarities. For instance, if a person is reserved by nature he can turn it to an instrument of his negotiating strategy. As he listens more than speaks he gets more information, therefore he can make more solid decisions. Otherwise, if you are a talkative negotiator and your opponent turns to be the same, when you start asking him questions he will feel more at ease, the atmosphere will become less tense

and your chances of coming to an agreement (mutually beneficial) will increase.

The author adds up that disclosing information has also too sides. It can hurt you or your negotiating partner. Or it can reveal some information about you that will show your interests from the other side and the other negotiator will be able to count with it [5].

D. Organized people

One may assume that the most efficient (and probably easy) negotiations are held with organized (or disciplined) people. This estimation holds true for a variety of reasons. First of all it is important to clarify the terms "disciplined" and "organized". Ambler [6] gives an insight to discipline and shows that this type of people: i) is capable of completing the tasks they have started; ii) possesses unique skill sets; iii) is able to bear ordeals and face "brutal facts"; iv) is willing to adhere to the organization`s systems for getting work done.

If we look at the definition of the word "organized" in a business dictionary we can get the following explanation: "State of being efficient or methodical. For example, the employee was highly organized and knew immediately where to find the general ledger."

E. Other considerations

It is generally known that emotions convey information about the individual who by any way expresses them. The process of the transmission can be denominated as intrapersonal and interpersonal effects on the individuals [7][8]. Intrapersonal effect is the one which is directed exactly on the person who experiences the emotions which are later transforming his behavior during negotiations. Interpersonal effect incites one negotiator who interacts with the second one to react differently due to the conduct of the first. The emotions are given much attention to, since they are used as an indicator of the preferences and priorities of an individual.

It is interesting to compare the theories of the behavior`s impact on the negotiations. Let us regard the first one conducted by Lerner [9] and the second by Pietroni [10]. The first, as it has already been described earlier, presented the negative outcomes for a negotiator that are induced by his (her) bad mood. Here, a person can face a problem of being confused by his (her) inner state and is willing to opt for the delay of the negotiations.

On the other hand, Pietroni argues that a negotiator undergoing emotional difficulties (such as being angry with something) wins from this state of affairs since his opponent tends to concede him in this case.

The two contrary viewpoints show two possible outcomes of a negotiation and leave field for more exploration and research. Often, dealing with the field of investigation we face all kinds of research methods by which we can identify the peculiarities of their originators.

For this purpose, this work will speak about the investigating methods showed above and will try to understand the profile of IT professionals through the prism of logical reasoning, creative, communicative and organization aspects. The article aims at presenting analyses about Brazilian IT negotiator's profile and dominant skills. The follow sections show studies and results about how IT professionals face negotiations. The evaluations were carried out during the last 4 years and a group of specialists were select for the experiments depicted in this work.

II. IT PROFESSIONALS IN BRAZIL

In 2008 there were more than 80 thousand IT regular professionals working specifically in IT companies in Brazil, as show in Table 1. Considering other companies (not only IT corporations), this number grows to 285 thousand people [11].

TABLE I. IT REGULAR PROFESSIONALS WORKING IN IT COMPANIES [11]

Occupancy	IT Companies	All Companies
IT Director	160	1.138
IT Manager	1.730	10.118
Computer Engineer	622	1.103
System Analyst	37.435	81.836
Software Development Technician	11.616	26.588
Network and DB Administrators	2.073	6.017
Computer Operators and Monitors	12.285	44.621
Network Operators	357	5.417
Data Transmission Operators	10.134	72.329
Telecommunication Staff	4.051	35.777
Total	**80.463**	**284.944**

The group of IT Brazilian Industry concerns people who work in followed activities: Hardware consulting, Software development and maintenance, Software development consulting, Data processing, Data base, on line distribution and electronic contents, Maintenance of IT equipments, among other IT activities.

On the other hand, there are people who work not only in IT Brazilian Industry but also in others [11]. This background reveals the relevance of IT area in Brazilian Industry.

Therefore, this work analyses how these professionals behave as negotiators. The deal, in this case, may occur in several circumstances in software development projects, such as in sales prospecting, purchase of services, changing requirements, resources allocation and other conditions.

III. EXPERIMENTS AND RESULTS

One goal of the current work is to verify if information technology professional, especially in the software development area, has dominant skills and specific behaviors styles in their negotiations.

Notably, Brazil is a huge country, with the fifth largest territory of the world and about 184 million people, according to the final report from the IBGE population accounts, made in 2007 in 5,435 municipalities [13]. In the Information Technology subject, Softex points out that in 2008, about 285.000 professionals worked in the Brazilian industry, whether or not for computer companies. This reports shows the expectative for 2009 were 340.000 IT people [11]. This work used this value to base the experiments.

From these data, some criteria that reflect the limitations and boundaries of this research were used. The confidence level, the margin of error and the population frequency were allowable for the desired ends.

- Target audience: 340.000
- Desired confidence level: 95%
- Estimated margin of error: 15%
- Estimated populational frequency: 50%
- Sample size: 43

In order to find out the sample size, Cochran studies [14] and the table available on the work of [15], whole presented in Figure 1. , were used.

PARAMETERS	
P(pop)	Hypothesized true proportion for population
N	Fixed population size
n(f)	Sample size for finite population
n(i)	Sample size for infinite population
Error	Desired margin of error or precision
C.L.	Desired confidence level
Test	One- or two-sided test
Z(tab)	Statistical parameter used to test significance
S.E.	Standard error

User Defined (Enter Your Information in These Cells)
Calculated by Program (These Cells are Locked)
Additional Definitions Found in Embedded Notes (i.e., cell with red triangles)

One-Sided Test: "Greater than Test"	One-Sided Test: "Less than Test"	Two-Sided Test:
H_0: P ≤ P0	H_0: P ≥ P0	H_0: P = P0
H_A: P > P0	H_A: P < P0	H_A: P ≠ P0

	P(pop)	N	Error (%)	C.L. (%)	Test	Z(tab)	S.E.
Enter Data:	0,50	340.000	15,0%	95%	2	1,96	0,001

MINIMUM NUMBER OF SAMPLES REQUIRED	
n(f) Finite Population	n(i) Infinite Population
43	43

Figure 1. Sampling calculation used in this research (based on [15])

This work uses questionnaires found in [12][16] and some interesting results have been achieved. The questionnaire was applied to 74 professionals aiming to verify the types of 3.prevailing skills. 53 people were from IT area and 21 were from other areas.

Although outnumbered, the individuals classified as non-IT were kept as a benchmark. Thus, some analysis could be observed, as shown in Figure 2, which suggests that IT professionals are concentrated in the Logical Reasoning profile.

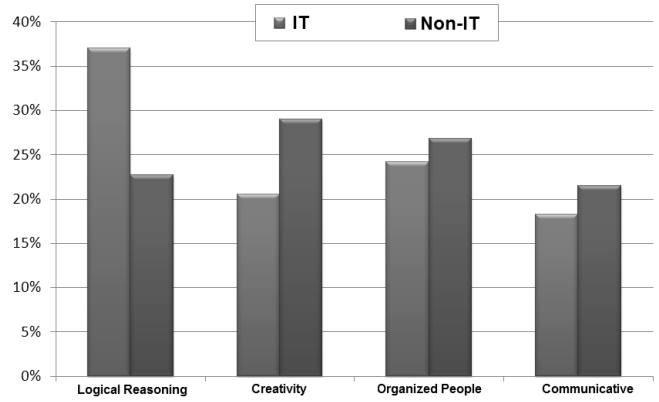


Figure 2. Profile types comparative

The arrangement presented in Figure 2 and Figure 3 suggests, generally, some characteristics of IT professionals that can be remarkable during negotiations, such as:

- Easily identify problems and rationally deal with them;
- Work with facts instead of aspirations;
- Remain prepared to justify the done deal, since they are comfortable at explaining their position in a logical and rational;
- Seem to be inefficient to handle multiple creative solutions;
- They have difficulty coping with emotions;
- They can take too long to close the deal on behalf of an attempt to find a unique and rational solution.

Figure 3 shows that the Intellectual and Technical / Organizational profiles are dominant in the interviewed IT professionals. This layout reflects the instinct to seek solutions through methodical and well-defined processes. Besides, the Figure 3 indicates that the individual IT has the greatest difficulties in dealing with interpersonal and creative aspects and is also slightly less inclined to routine, since it does not involve major innovations and challenges.

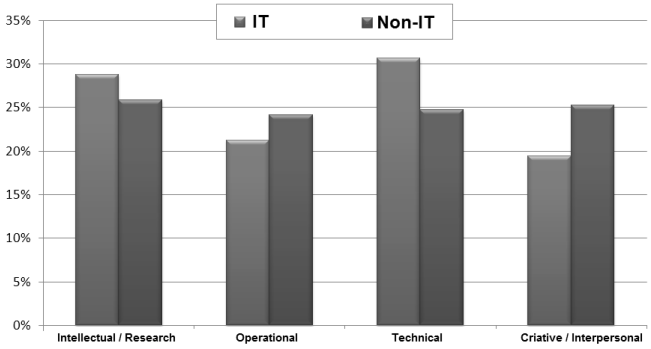


Figure 3. Prevailing aptitudes

Figure 4 and 5 respectively show the profiles and dominant aptitudes of IT professionals, referred as Technical and the ones, known as Management. The technical individuals do not lead any person, while the managers have leadership role.

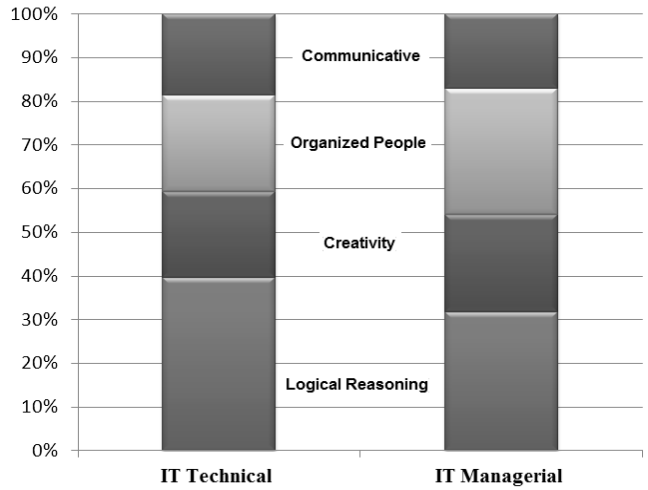


Figure 4. Profiles: IT professional

Figure 4 shows that the professional IT technician tends to be more logical and analytical while management has a more balanced structure, highlighting the organization as dominant aptitude.

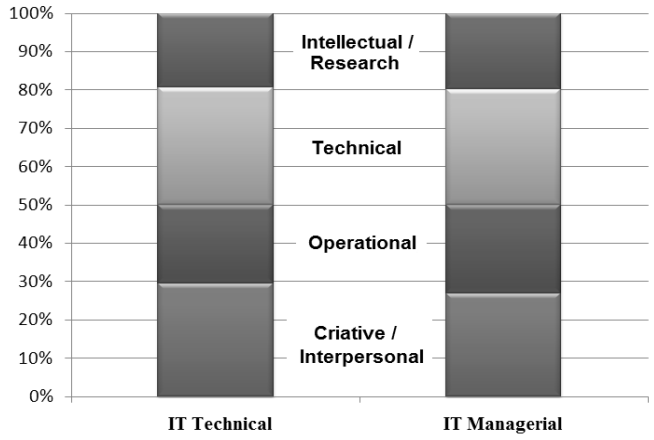


Figure 5. Prevailing aptitudes: IT professional

In the skills assessment, the results did not show discrepancies between the technical and managerial profiles, as illustrated in Figure 5.

From this research, some aspects (methods and tools) were developed or customized to deal with the points of better understanding and the possible difficulties faced by professionals at the time of negotiation:

- The setup method should consider that the individual will be rational in their assessment and will have difficulties to transcribe large volumes of information.
- Models that comprise sequential steps may have synergy with the methodical professional profile.
- Because it is a more synesthetic profile, the preparation tool must provide mechanisms that indicate that the provided data was really processed and instead of just be stored, for example, if the tool request textual data, it is important to generate graphs or tabulated mining mechanisms text indicating the major concerns (words or phrases, for example) in the current negotiation.
- Also from the synesthetic profile perspective, teaching methodologies that rely on computers can benefit IT professional negotiation skills learning. In this case, as it is a learning environment, a negotiation games approach can be associated with a context that involves settlement options and multiple outcomes in order to supply the lack of creativity intrinsic to the professional.
- About the difficulty of handling interpersonal work, the tool must contain mechanisms to assemble a relationship tree, so that the IT negotiator always remember the others involved, even when they are not at the negotiating table.

CONCLUSIONS

We addressed that a great difficulty during negotiations is to distinguish if the negotiator is well prepared or not. This work provides an analysis of how to understand a negotiators' psychological profile, their tendencies during conflict resolution and attempts to show their experience in the involved negotiation context.

Experiments were carried out through questionnaires and psychological tests that respondents could not realize they were being analyzed. This approach was important to guarantee impartiality during the answers.

The results depict the relevance of behavioral studies, especially considering how difficult is to analyze tendencies during negotiations among IT professionals. The rate of use of this mechanism is growing as well as the interest of new IT professionals and also students.

REFERENCES

[1] Ezendu, E. (2009), "Building Negotiation Skills", http://www.slideshare.net/ezendu/building-negotiation-skills (last accessed on 12 March 2011)

[2] Newall, I. (2009), Research in Organizational Behavior, 22: 1–50. Logical Persuasion In Negotiation, (last accessed on 04 November 2009)

[3] Spangle, M. & Isenhart, M. W. (2003). Negotiation: communication for diverse settings, Ed. SAGE, 435 pages

[4] DeBono (1967), New Think: The Use of Lateral Thinking, 1967

[5] Stitt, A. (2010). Negotiation Tip of the Month by Allan Stitt, http://www.sfhgroup.com/ca/training/negotiation/negotiation-tip/, tips from March 2010 and July 2010, (last accessed on 10 March 2011).

[6] Ambler, T. E. (2009). A Culture of Discipline - Building Toward Great, Center for Simplified Strategic Planning, Inc. Ann Arbor, http://www.strategyletter.com/CD0207a/featured_article.php, (last accessed on 12 March 2011)

[7] Keltner, D., & Haidt, J. (1999). The social functions of emotions at multiple levels of analysis. Cognition and Emotion, 13: 505–522.

[8] Morris, M. W., & Keltner, D. (2000). How emotions work: The social functions of emotional expressions in negotiations.

[9] Lerner, J.S., D.A.Small and G.Loewenstein (2004) "Heart strings and purse strings: carryover effects of emotions on economic decisions", Psychological Science, 15: 337-41

[10] Pietroni, D., Van Kleef, G. A., De Dreu, C. K. W., & Pagliaro, S. (2008). Emotions as strategic information: Effects of other's emotions on fixed-pie perception, demands and integrative behavior in negotiation. Journal of Experimental Social Psychology, 44, 1444-1454.Spangle, M & Isenhart, M.W. (2003). Negotiation: Communication for Diverse Settings, p 134

[11] Softex, (2009) Software e Serviços de TI: A indústria Brasileira em Perspectiva. Available at: http://publicacao.observatorio.softex.br/_publicacoes/arquivos/apresentacoes/Apresentacao_Software_e_servicos_DEZEMBRO_ITS.pdf (last accessed on 13 February 2010).

[12] Rodrigues et al (2010). An approach to understand IT professionals' behavior during negotiations, Negocia, negocia.fr, Paris (France)

[13] IBGE, (2007). Contagem da População 2007, Ministério do Planejamento, Orçamento e Gestão. Available at: http://www.ibge.gov.br/home/estatistica/populacao/contagem2007/contagem.pdf. (last accessed on 13 February 2010).

[14] Cochran, W., (1977). Sampling techniques 3 ed., New York: Wiley.

[15] Snedecor, G. & Cochran, W.G., (1992). Statistical methods 8 ed., Ames Iowa: Iowa State Univ. Press.

[16] Miranda, R., (1997). Além da inteligência emocional : uso integral das aptidões cerebrais no aprendizado, no trabalho e na vida, Rio de Janeiro: Campus.

Designing a Distributed Systems Architecture Testbed for Real-Time Power Grid Systems

Yan Liu, Ian Gorton
Fundamental & Computational Sciences Directorate
Pacific Northwest National Laboratory
Richland, WA
yan.liu@pnl.gov, ian.gorton@pnl.gov

Yousu Chen, Shuangshuang Jin
Advanced Power and Energy Systems
Pacific Northwest National Laboratory
Richland, WA
yousu.chen@pnl.gov, Shuangshuang.jin@pnl.gov

Abstract— **Power engineers who are striving to improve real-time attribute of power grid applications are ill equipped with software engineering methods and tools that allow them to rigorously evaluate their designs, taken into account data communication, geographic locations, and high performance computing capacity. This paper presents a technical approach to designing a testbed for embedding real-time monitoring and computation functionalities into the power grid. The approach focuses on integrating the parallel computational models with the data management infrastructure for near-real time state estimation. We study and summarize various forces and requirements that drive the design decisions in the distributed systems architecture. Given the continental scale of the power grid, it is important for the testbed to be extensible and scalable within a complex topology of physical entities, controlled by an overlaid network of power utilities and regulatory balancing authorities. This paper outlines the technical steps, and software toolkits to develop this testbed.**

Keywords-distributed architecture, power grid, real-time, testbed

I. INTRODUCTION

The power grid is a continental scale distributed generation and distribution system with a complex topology of physical entities, controlled by an overlaid network of power utilities and regulatory balancing authorities. For example, the eastern interconnection has approximately a dozen reliability coordinators and over a hundred balancing authorities. Each reliability coordinator and balancing authority has its own control center that uses a state estimator to maintain situation awareness of their internal system; the state estimator at the reliability coordinator is often a hierarchical state estimator that covers several balancing authorities.

To improve the reliability of the power grid, near real-time situational awareness is recognized as one key enabling functionality for better planning and reliable grid operations. Underpinning this vision, emerging and future deployments of high speed sensors such as PMUs (Phasor Measurement Units) give direct access to the current state of the grid at various points within the power grid. Synchronized phasor measurements are commonly referred to as *synchrophasors*. PMUs produce real-time synchrophasor data that capture the dynamic characteristics of the power system, and hence facilitate time-critical control. PMUs typically generate measurements 30 samples/second with precise time synchronization. This is in comparison with traditional Supervisory Control And Data Acquisition (SCADA) measurements, which generate a sample every five seconds or longer and are not time synchronized.

PMUs are becoming increasingly attractive in various power system applications such as system monitoring, protection, control, and stability assessment. The number of PMUs connected to the power system is expected to increase by 3-5 orders of magnitude by 2020. Inevitably, the substantial growth of these high quality sensors in the near future creates the need for new models such as dynamic state estimation to enable real-time predictions[1]. It is envisioned the time to solution of such models needs to be radically reduced to the 10 milliseconds to 1 second range, compared to current delay of 2-4 minutes to obtain results. In addition, the growth of model size is expected to be 2 orders of magnitude in the next 10 years.

Consequently, the solution is envisaged that High Performance Computing (HPC) capabilities accommodate the computing demand of these models at the regional scale [4,5]; and state estimation results are communicated at the wider continental scale. Connecting HPC-enabled distributed power models across regional locations becomes a systems architecture design challenge, involving many intricate factors

- Transmission latencies for data from sensors to distant monitoring applications;
- Requirements from applications for hard-real delivery of monitoring data in order to provide highly accurate situational awareness;
- Tolerance of applications for delayed or missing data;
- The hierarchical nature of data collection functions and associated monitoring applications such as state estimation.

At the regional scale, evaluating the performance of running power models on the local computing center is vital to meet the required time constraints. At the continental scale, the geographical location of various computing centers impacts the communication delays of the status exchanged between regional computing centers [9]. As a result, it is essential to design a distributed systems architecture well equipped with a rigorous evaluation method and software toolkit to collect empirical evidence for a design option.

In this paper, we propose technical approaches to designing a testbed for exploring HPC-enabled power models in

distributed systems architecture. In section II, the testbed design is motivated by the requirements and issues arising from the large scale and distributed nature of the power grid. The testbed approach is presented in section III, followed by the conclusion.

II. System Requirements and architecture challenges

At the infrastructure level, one mission of North American SynchroPhasor Initiative (NASPI) is to create robust, widely available and secured data management infrastructure, called *NASPInet* [2]. NASPInet explored a tiered architecture for distributed continental scale network. This conceptual architecture outlines the guidance at the network level, connecting sensors, monitoring entities, and power grid applications through NASPInet phasor gateways and data bus middleware. However, it still remains an open research question - what is the optimal architecture design to leverage the data management infrastructure at the programming level.

To address this issue, we envision the solution is threefold. First, power models in parallel have practical needs for connecting to the data management infrastructure[1]. These models usually implement the algorithms using specific parallel computing programming models such as MPI (Message Passing Interface), hence their communication with the infrastructure is more complicated than device-based equipments such as sensors in the power grid, which have limited computing tasks and mostly communicate data directly through network protocols. For an instance of the distributed state estimation, communication across the local MPI boundary is necessary to exchange power grid monitoring data or analysis results.

The interface between MPI code and the infrastructure not only needs mechanisms wrapping the middleware APIs, but also software utilities pre-processing and post-processing accessed data. For example, the monitoring data from remote regions of the power grid represent bus line status in addition to the local regional bus line model. These data need to be assembled into one bus line model according to an overall network topology. Since data may arrive at different time, they need to be buffered first and then partitioned to the processors for parallel computing.

Second, the effectiveness of the parallel power models includes near real-time requirements solving the models and the accuracy of the state estimation. The evaluation of a model must take into accounts the communication delays and failures. In fact, power grid engineers are ill equipped with simulation tools that allow them to construct representative distributed architecture scenario and do what-if analysis. Although network simulators are available, they are mostly used to analyze the end-to-end data flow between entities [8] rather than being integrated with the top level applications.

The simulation of the distributed architecture entails customizing the network simulators in a way that allows power engineers to configure the network topology by assigning values to networking parameters for each bus line within the communication scope. In addition, sampling data need to be collected according to the network topology configured. In reality, these sampling data should be obtained using the interface through the infrastructure. For the testing purpose, sampling data are synthetically generated and sent to the parallel power models through the interface.

Third, applying the parallel computational model to improve the real-time feature of the power models is an active research topic. The power grid algorithms and tools are typically run on personal computers and at most, moderate-sized clusters. As we envision the size, complexity, and interconnectedness of the power grid are dramatically growing, forecasting the vulnerabilities of the future grid using these outdated capabilities hinders the ability to predict, react to and/or mitigate failures such as the 2003 blackout on the East Coast in the US.

New development of the power models underpinned by HPC capability or re-engineering the existing ones are emerging. The ability to estimating the end performance at the design level can provide early feedback to improve the model as well as reduce the costs of faulty deployment. Predicting performance at the design level entails a systematic performance analysis method and tools to collect profile data.

Overall, we define the testbed in this paper that encompasses the simulation toolkit for running experiments and predicting performance of the power models in distributed systems architecture. By experiments we mean exploring scenarios of deploying the HPC centers at different geographic locations, observing the data communication effects on power model performance, and partitioning the network topology to do what-if analysis.

III. The technical approach

The testbed is designed to explore the distributed systems architecture depicted in Figure 1. In this architecture, a site (such as site A) accommodates the computing center deployed at a local balancing authority that controls the system. Such a computing center is equipped with parallel computing software and hardware. A site also encapsulates the computing center of a regional reliability coordinator (such as site B) that oversees reliable operation of the systems. A reliability coordinator connects to multiple sites of balancing authorities or other reliability coordinators, exchanges lower level state estimation results, and performs hierarchical state estimation. We envision both the balancing authority and reliability coordinator employ high performance computing centers to run power models such as state estimation.

The sensor reading data from transition substations are sent via SCADA systems within each site. With PMU data available from several Phasor Data Concentrators (PDC)s deployed within the site, PMU data can be combined with SCADA data to improve the accuracy of state estimation.

[1] Later in this paper, we refer data management infrastructure as infrastructure for short.

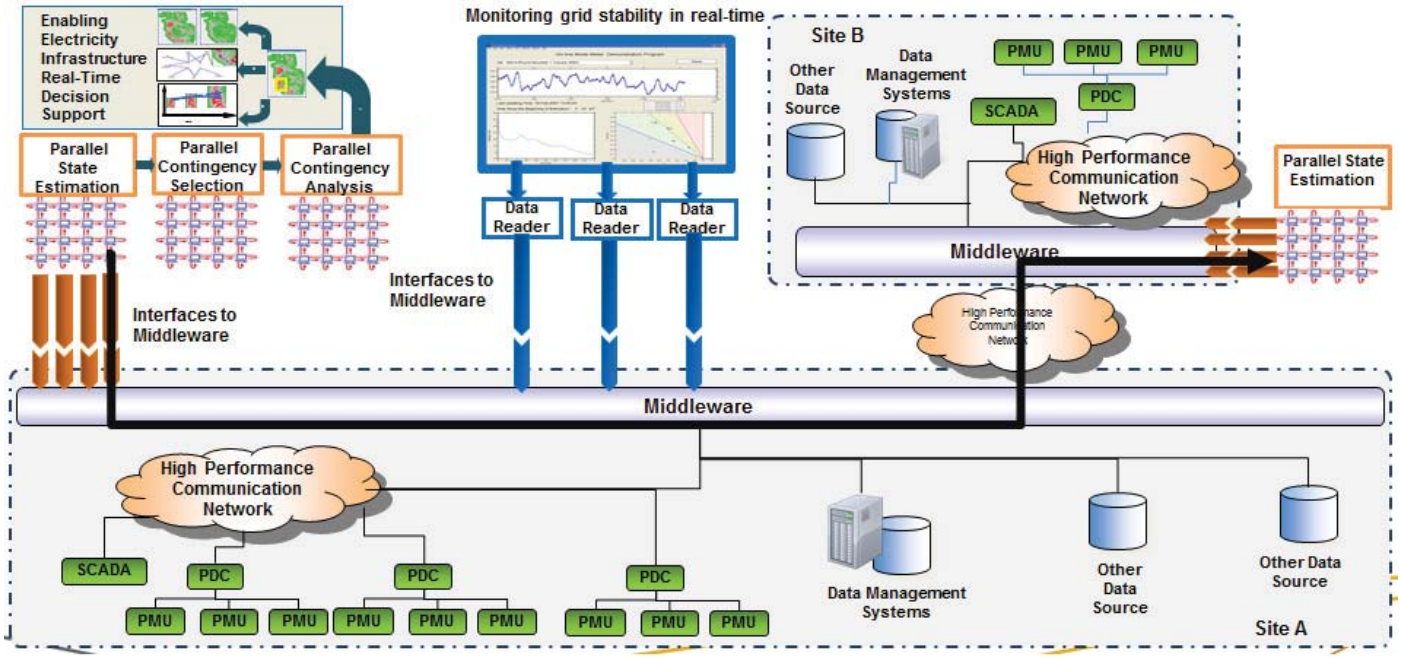


Figure 1: A distributed systems architecture for real-time power grid system

The data communication across sites is through dedicated middleware that can handle high throughput and large amount of sensor reading data in the power grid. In this work, we assume the middleware provides APIs and supports the IEEE standard protocols of disseminating power grid data. On each site, historical sensor reading data from the local region are stored in its data management system. The middleware is also responsible for retrieving the datasets of interest to the power grid applications.

Given the middleware and data management infrastructure in Figure 1, the testbed is devised in three steps as follows.

A. *Step one*

An interface layer is developed to wrap the communication between the MPI processor and the outside as shown in Figure 1. Using MPI, the parallel power model usually has a master processor to fetch the data for computing, partition the data and dispatch a piece of data to a number of worker processors running the computing tasks in parallel. Hence only the master node needs to explicitly invoke the interface as a client to the remote site. We refer the master node as the communicator node.

The interface layer devises key components of data processor and data buffer at the client side. The data processor resolves the location of data demanded (either from the local region or from remote sites) from a centralized data registration service. The data processor then uses the location information to connect to the corresponding server and fetch the data through the middleware. Depending on the protocol that the middleware supports, the data processor needs to extract the required fields of data (such as bus voltage, phase angle) and assemble them as inputs to the parallel power models. To illustrate the assembly of the data from disparate sites, the IEEE 118 bus system is grouped into three areas as the experiment devised in [3]. Each area can be considered as a separate site in the architecture of Figure 1 that delivers the SCADA and PMU data along the bus lines involved. Consequently data retrieved from individual areas need to be assembled into a single bus model before they are input to the state estimation model.

B. *Step two*

The interface layer is ported to a network simulation tool as shown in Figure 2. The network simulation tool is configured to estimate the delay incurred by data communication from remote sites.

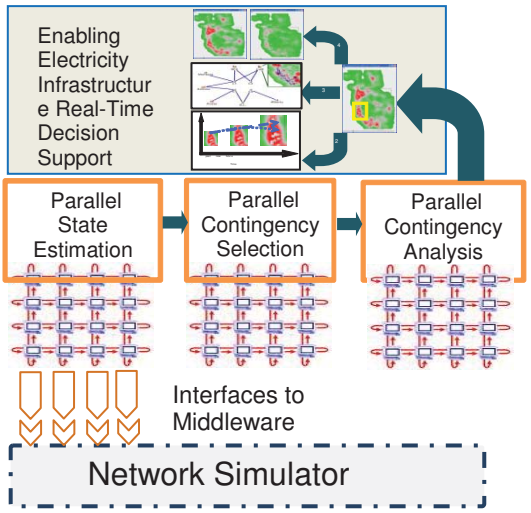


Figure 2 Simulating data communication delays

Since individual site in the architecture represents the power grid utility centers and balancing authorities, the network simulation needs to have a precise network topology reflecting the real topology of physical entities. The goal of the distributed systems architecture design is to embed HPC-enabled power grid models at the optimal site, given the geographic locations of the physical entities. Using the network simulator has obvious benefit in the testbed. The network simulator allows flexible designation of the HPC sites, and estimates the effects through simulation.

C. *Step three*

The ultimate goal of the testbed is to predict the performance of new parallel models and effects of any modifications made to existing models. Without complete source code of the model, predicting the performance at the design level entails a systematic approach. Our previous work [6] develops a benchmark based performance predication method for Java enterprise applications. The core of this method is applied to the testbed as shown in Figure 5. Benchmarks are deployed to observe the performance characteristics. They are useful vehicle to obtain performance profile by empirical measurements. The performance profile is then used to calibrate a performance model characterizing the parallel models. Performance modeling of parallel applications is well established research [7]. Existing modeling techniques can be applied to develop the model.

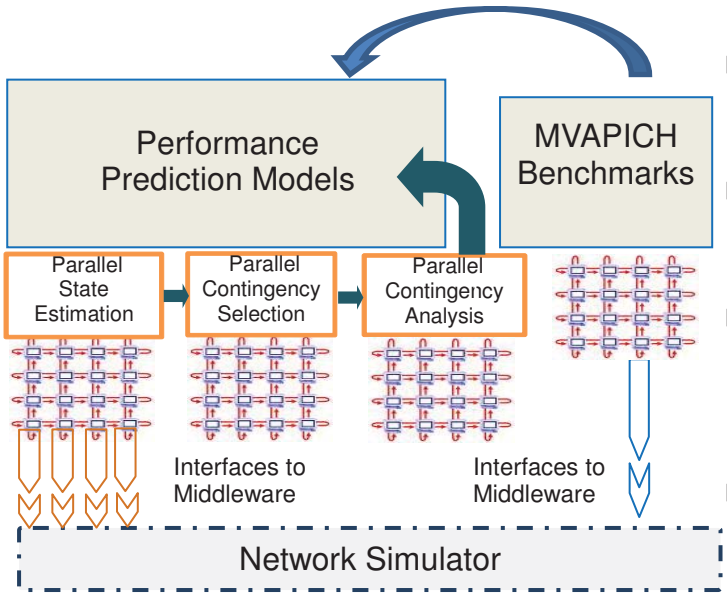


Figure 3 Performance prediction using the testbed

The testbed packs the interface layer, the customized network simulation toolkit, the benchmark suite, and the performance analysis model in a single environment. The testbed aims to facilitate engineers who design future HPC power grid models to evaluate their design, given the infrastructure capacity, networking condition, and geographic location.

I. CONCLUSION

The evolution of the power grid advocates HPC capabilities for improving the power models and delivering near real-time operations in power systems. Computing centers equipped with HPC software and hardware need to be designated in the overlaid network of physical entities. This raises architecture challenge to integrate the HPC applications with the data management infrastructure and the networking environment. Power engineers are in need of software engineering tools that facilitate them to evaluate their parallel power models in distributed systems architecture. From our research experience, we prioritize the requirements and discuss the challenges to address them. We propose a testbed design to make the integration between HPC applications and the infrastructure more transparent to the power engineers. The ultimate goal of the testbed is a useful tool for engineers to estimate the performance of their models under certain conditions of data communication. Our future work aims to validate the testbed design with hieratical state estimation and dynamic analysis of power systems.

REFERENCES

[1] Bakken, D.E., Bose, A., Hauser, C.H., Schweitzer. E.O., Whitehead, D. E., Zweigle, G.C., Smart Generation and Transmission with Coherent, Real-time Data, Technical Report TR-GS-015, August 2010. http://gridstat.net/publications/TR-GS-015.pdf [last accessed on Mar 20, 2011]

[2] Bobba, R. , Heine, E., Khurana, H., and Yardley, T., "Exploring a Tiered Architecture for NASPInet," Proceedings of the IEEE PES Conference on Innovative Smart Grid Technologies (ISGT), Gaithersburg, MD, Jan. 19-21, 2010.

[3] Bose, A., Poon, K., Emami, R., Implementation Issues for Hierarchical State Estimators, Final Project Report, September 2010. http://www.pserc.wisc.edu/documents/publications/reports/2010_reports/Bose_State_Estimation_S-33_Final_Report_8-2010_ExecSum.pdf [last accessed Mar 20, 2011]

[4] Chen Y, Z Huang, and D Chavarría-Miranda. 2009. "Performance Evaluation of Counter-Based Dynamic Load Balancing Schemes for Massive Contingency Analysis with Different Computing Environments " In IEEE PES General Meeting. PNNL-SA-69878, Pacific Northwest National Laboratory, Richland, WA.

[5] Gorton, I., Huang, Z., Chen, Y., Kalahar, B., Jin, S., Chavarría-Miranda, D., Baxter, D., and Feo, J. 2009. A High-Performance Hybrid Computing Approach to Massive Contingency Analysis in the Power Grid. In Proceedings of the 2009 Fifth IEEE international Conference on E-Science (December 09 - 11, 2009). E-SCIENCE. IEEE Computer Society, Washington, DC, 277-283. DOI= http://dx.doi.org/10.1109/e-Science.2009.46.

[6] Liu, Y., Gorton, I., Liu, A., Jiang, N., and Chen, S., 2002. Designing a test suite for empirically-based middleware performance prediction. In *Proceedings of the Fortieth International Conference on Tools Pacific: Objects for internet, mobile and embedded applications* (CRPIT '02). Australian Computer Society, Inc., Darlinghurst, Australia, Australia, 123-130.

[7] Martinez, D.R.; Blanco, V.;Boullon, M.;Cabaleiro, J.C.; Rodriguez, C.; Rivera, F.F.; Software Tools for Performance Modeling of Parallel Programs, Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International, e: 26-30 March 2007, 1 - 8

[8] Ragib Hasan, Rakesh Bobba and Himanshu Khurana, "Analyzing NASPInet Data Flows", IEEE Power Systems Conference and Exposition (PSCE '09), Seattle, Washington, March 2009.

[9] Tomsovic, K.; Bakken, D.E.; Venkatasubramanian, V.; Bose, A.; , "Designing the Next Generation of Real-Time Control, Communication, and Computations for Large Power Systems," *Proceedings of the IEEE* , vol.93, no.5, pp.965-979, May 2005

Supporting Software Engineering Education through a Learning Objects and Experience Reports Repository

Rodrigo Santos, Cláudia Werner
COPPE/University of Rio de Janeiro
Rio de Janeiro, RJ, Brazil
{rps, werner}@cos.ufrj.br

Heitor Costa
Federal University of Lavras
Lavras, MG, Brazil
heitor@dcc.ufla.br

Simone Vasconcelos
IFF – Fluminense Federal Institute
Campos, RJ, Brazil
simonevs@iff.edu.br

***Abstract* – The formation of human resources in Software Engineering (SE), as well as in other areas of Computer Science (CS), depends on the education research. It is possible to identify some efforts performed by SE community. However, they represent isolated and localized initiatives, reducing their diffusion and use in large scale in global and regional scenarios. This work presents a strategy to support the educational research in SE focused on empirical research. The goal is to contribute to a body of knowledge in SE education thought a learning objects and experience reports repository.**

Keywords – Software Engineering Education;Learning Objects; Experience Reports; Empirical Software Engineering; Reuse.

I. INTRODUCTION

The teaching and learning process in Software Engineering (SE) has been discussed in the last years [14] to reflect the current demand for complex and large systems of systems and software-intensive systems. Usually, professors teach SE concepts through theoretical courses, contemplating practical activities in a small and short time project. Thus, the new software engineers meet a scenario where learned techniques and methods are not applicable, and they use *ad hoc* practices based on their experiences, focusing on coding. This can negatively impact the establishment of engineering principles and good practices in software development processes. The university should not take all the responsibility (since it is not a company) but need to prepare the students for real challenges in SE industry [15].

Motivated by this fact, some efforts consisted in developing reference documents such as SWEBoK [7], SE2004 [1] and CSDP [10], as well as the establishment of many undergraduate and graduate programs around the world. However, all these efforts tend to dissipate when treated as a set of isolated strategies, contributing to a divergent and localized scenario in SE education, no evidence-based and no focused on use of the best educational resources [9]. So, it is important to identify mechanisms that allow the organization of this knowledge, exposing it to the SE community.

In order to contribute to SE education, the empirical paradigm involves the collection and analysis of data and evidences that can be used to characterize, evaluate and show relations among technologies, practices, and experiences on SE teaching and learning processes. Therefore, empirical results can compose a body of knowledge over time [5], providing a base to accepted and well-formed theories about SE education. These theories can be materialized by a learning objects and experience reports repository, where researchers and professors can communicate, act as producers and consumers of educational resources, and also drive SE education to a more mature field of Computer Science (CS).

In this sense, this paper aims to present EduSE Research Strategy, an approach to explore empirical studies as a mean to support large scale SE education. From cooperatively developing a scientific investigation protocol with SE community in four phases, a unified body of knowledge can be organized, taking into account SE global aspects and regional peculiarities (i.e., problems, solutions and challenges in education in SE areas). Also, this process allows the existence of a dynamic and evidence-based repository of SE learning objects and experience reports in its fifth phase. The paper follows this structure: Section II discusses the concepts of learning objects and experience reports, contextualizing SE education; Section III presents EduSE Research Strategy, its phases and infrastructure; and Section IV points out final considerations and future work.

II. BACKGROUND

A. Learning Objects and Software Engineering Education

A learning object is the smallest independent structural experience that contains an *objective*, a *learning activity* and *assessment*. It represents a set of reusable and self-contained digital resources with an educational objective structured by three internal components: *contents*, *learning activities* and *contextual elements* [25]. A learning object must have an external information structure (metadata) that facilitates its storage, search and retrieval.

For the Learning Technology Standards Committee (LTSC) of the Institute of Electrical and Electronics Engineers (IEEE), a learning object is any digital or non-digital entity that can be used in the technological support to learning, education or training, e.g., texts, images, graphs, tables, presentations, diagrams, videos, games or any digital educational material, used by the professors assisting students in teaching a subject [11]. In its essence, a learning object is reusable.

Some types of information may be included in a learning object and its metadata [11]: general course descriptive data, life cycle, instructional content, glossary of terms, quizzes and

assessments, rights, relations to other courses, and educational level. Some works consider a process model for learning object development, such as ADDIE [17]. Besides, the creation of the learning object standards makes the definition of important characteristics possible, such as reuse, durability, and accessibility, highlighted in standards (e.g., SCORM [2]).

In SE education, the use of learning objects aims at minimizing the mentioned problems since it provides mechanisms to facilitate the teaching and learning processes, beyond promoting relations between concepts, practices and results in SE. Some learning environments are developed to assist the SE learning process and simulate many real situations related to software development that stimulates and motivates students. This kind of learning objects can be games, scenes, graphs, and others. Some games used in SE learning considering *Project Management* (e.g., *The Incredible Manager* [8]) and *Requirements Engineering* (e.g., *Guess what we want* [3]) areas.

B. *Experience Reports and Software Engineering Education*

An important element in CS educational techniques is the *experience report*, i.e., the exposition of a tradeoff between educational proposition and realization made by professors, which evidences success and failure strategies and tactics in teaching and learning processes over time [9]. This element represents a channel among professors and researchers to treat educational issues and communicate new solutions among them. In this sense, the efforts to generate an experience report database use empirical techniques [5]. The empirical studies allow theories to be formulated, tested, and validated, evolving an experience report to a status of evidence (or not), i.e., evidences are generated from characterizing, assessing, predicting, controlling, and improving products, processes, and theories. Thus, experiences in education can explore these studies towards the continuous improvement.

The execution of educational research is supported by studies which aim to discover something unknown or test something known. They can be classified into **primary** and **secondary studies**. Primary studies are directed by hypotheses to be verified or inferred. These studies are conducted when it is necessary to characterize a particular learning object in use within a specific context (e.g., game, educational technique or software). The result analysis of a primary study can be quantitative, semi-quantitative or qualitative, and it can represent (or is associated to) an experience report. Three types can be identified [26]: (i) *case study* is executed to observe a behavior or phenomenon shown by an entity within a limited time; (ii) *quasi-experiment* is executed once a greater control of the situation is needed, aiming at manipulating one or more variables and control the value of others; and (iii) *survey* is executed to collect information from a sample of the population through a set of questions.

Although primary studies characterize a specified learning object, they are not sufficient [26]. Thus, secondary studies aim to integrate results from several correlated primary studies. Secondary studies are useful in revealing evidences and constructing bodies of knowledge that can be mapped to real and everyday educational experiences. These studies happen through *systematic reviews* and *meta-analysis* [6]: the first one is a methodology focused on a literature search protocol; and the other one is a study applied after a systematic review to statistically treat the quantitative data from analyzed papers.

Initiatives of empirical studies plan and execution in SE education can be found in the literature, some of them reporting experiences of using different and interesting strategies to have the students' attention. Papers of two conferences, Brazilian Forum on SE Education (FEES) and International Conference on Software Engineering Education & Training (CSEET) were examined, identifying some educational discussions, such as (i) SE as an undergraduate course and SE in CS courses, (ii) teaching areas of knowledge from SWEBoK, (iii) interdisciplinarity, and (iv) status of SE in university and industry. This analysis is presented in FIGURE I.

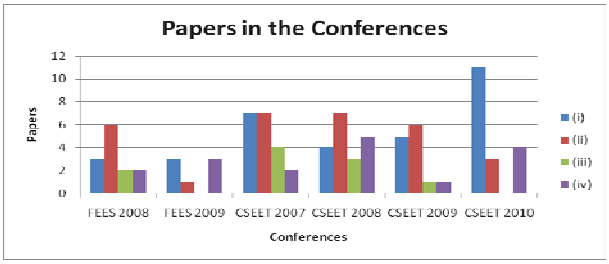


FIGURE I. Educational aspects identified on papers

III. EDUSE RESEARCH STRATEGY

Based on Shull *et al.* [22] and Spínola *et al.* [23] proposals, EduSE Project was started in order to generate a collaborative and large scale research strategy to support SE education, called EduSE Research Strategy [19]. It proposes secondary (i.e., systematic review) and primary (i.e., survey) studies to establish, base and integrate SE education researches in a *collaborative* (i.e., among different SE researchers), *distributed* (i.e., among different institutions) and *specialized* (i.e., dividing experts by SE areas work groups, such as requirements, design, test, reuse etc.) way. Hence, two communities are highlighted as direct EduSE Research Strategy stakeholders: *researchers* (who research in SE and wish to improve the education of their SE areas) and *professors* (who teach SE classes).

Using the GQM (Goal/Question/Metric) approach [5], the project goal *is* to analyze SE teaching and learning processes *for the purpose of* characterizing *with respect to* identifying problems, solutions and challenges, beyond regional peculiarities from the SE researchers' *point of view*, *in the context of* SE and CS courses. In this sense, the research question could be established as: *What are the main problems, existent solutions and pointed challenges in SE teaching and learning process related to dimension <SE area>?*.

From the research question, four phases were developed to compose EduSE Research Strategy's kernel (FIGURE II.) by deriving work focused on the technology definition research strategy proposals mentioned above. These phases are connected and refined by activities and tasks: (1) *Ad hoc Literature Review Phase* aims at identifying basic concepts about SE education, and allowing the definition of a systematic review protocol for supporting each SE area work group; (2)

Systematic Review Phase aims at elaborating and executing the systematic review protocol in each SE area work group – based on the results extracted from papers analysis, the work group decides if the study needs to be refined, or if the set of mapped knowledge should be evaluated through a survey; (3) *Survey Phase* aims at planning and executing studies to evaluate the knowledge acquired in the previous phase, considering SE professors community perspective in regional scenarios; and (4) *Body of Knowledge Phase* aims at joining all knowledge obtained from the last three phases in order to organize a SE education body of knowledge (i.e., studies reports; tracking among researches questions and problems, solutions, challenges and regional peculiarities; search, retrieval and communication mechanisms to find and improve teaching practices and SE techniques etc.). Thus, this phase requires a *repository* to share information related to SE education.

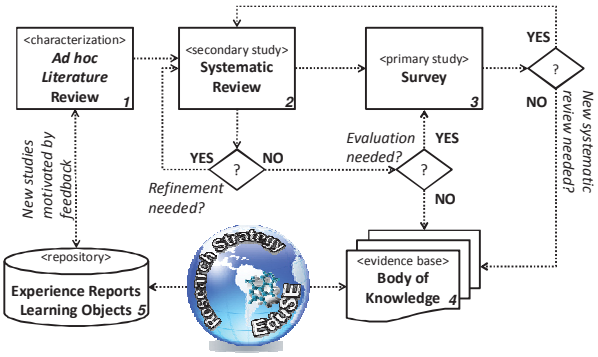


FIGURE II. EduSE Research Strategy Phases

When the set of research questions mentioned before are treated through the empirical paradigm, a body of knowledge can support well-formed theories in this field and directly impact the process of transferring research products to software industry, since SE researchers and professors can interfere in SE human resources formation. It means that SE courses and classes should be treated as laboratories that use leaning objects and are based on experience reports to explore regional SE industry issues, using and improving SE tools, techniques, and processes, as well as disseminating SE good practices. All stakeholders win: *regional and global software industry* hires well-formed software engineers, and *universities and research centers* have a chance to evaluate their SE scientific researches through educational researches on their SE areas using the empirical paradigm in different scenarios and subjects.

The results of preliminary survey with professors discussed in [21] provided knowledge for two research activities in EduSE Project: the first one is the definition of the *EduSE Research Strategy fifth phase*, and the last one is the modeling of a *web environment architecture* to support EduSE Research Strategy. Initially, the *Management of Learning Objects and Experience Reports Phase* aims to support the learning objects and experience reports development and life cycle, as shown in FIGURE III. So, a mechanism integrated to the repository should allow search, retrieval, storage, documentation, and publishing of the mentioned educational resources.

Aiming to validate their researches through *in vivo* primary studies, SE researchers create and publish educational resources as reusable components, each of them composed by a *learning object*, a *manual* (instructions), and an *XML scientific protocol* which will generate a questionnaire with data of interest to be collected when a resource is used *in vivo*. On the other hand, aiming to improve their classes and save time, SE professors freely search and retrieve these educational resources, and download them after agreeing with feedback terms. That is, after using and collecting required data in classes, SE professors should provide feedback, fulfilling a form with the questionnaire prepared by the resource producer. Hence, SE researchers can aggregate and summarize data, extracting information about their educational resources and, consequently, improving educational resources and related SE research products. Also, they can publish papers related to their researches based on real cases, creating a positive cycle.

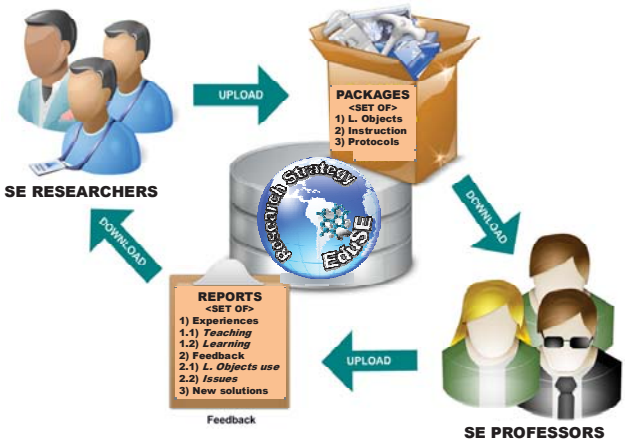


FIGURE III. Management of Learning Objects and Experience Reports

In order to share teaching and learning experiences as well as learning objects evaluations, SE professors can report opinions and describe situations, particularities, scenarios, and contexts faced during SE classes in any SE area or course. This information is also a reusable component and should be stored and classified to be searched and retrieved. SE researchers can access them to develop or evolve learning objects as solutions. In this sense, it is important to track learning objects and experience reports to problems, solutions, challenges, and regional peculiarities previously identified in EduSE Research Strategy phases 2-3 that compose the SE body of knowledge.

Based on the five phases, a web environment architecture to support EduSE Research Strategy was developed and a infrastructure is under development, the *EduSE Portal*, as shown in FIGURE IV. EduSE Portal is a JEE web information system to Computer Supported Cooperative Work developed on the JBoss Seam framework [4]. It combines communication, collaboration, cooperation, and coordination mechanisms to empirical studies processes, activities and tasks aiming to semi automatize EduSE Research Strategy phases 1-4. Currently, EduES Project is focusing in integrating EduSE Portal and the components and services library *Brechó-EcoSys* [20]. This step aims at generating a SE education ecosystem[1] for learning

[1] *Software Ecosystem* consists of a set of businesses functioning as a unit and interacting with a shared market for software and services, together with the relationships among them, frequently underpinned by a common technological

objects and experience reports development and management, based on software reuse processes [12].

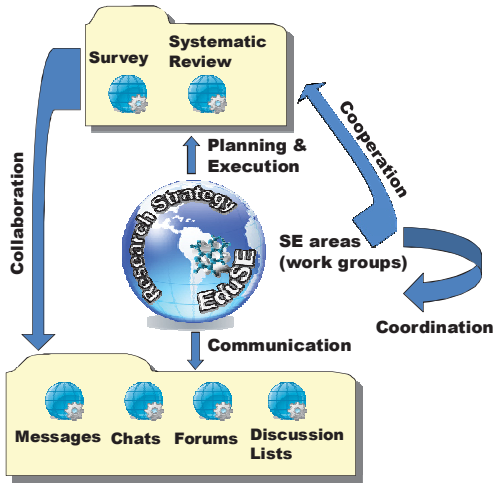


FIGURE IV. EduSE Portal's Architecture

IV. CONCLUSION

Since SE teaching and learning processes consist in an important concern for the next years, this paper presented EduSE Research Strategy, an approach to explore empirical studies as a mean to support large scale SE education. Related work consists of (i) SAKAI [18]: a tool to create an environment to Computer-Supported Collaborative Learning, (ii) Moodle [16]: an open source tool to Learning Management Systems, and (iii) SWEnet [24]: guided by SWEBoK and IEEE, this network community presents a framework to categorize knowledge areas in SE. In any case, the empirical paradigm basis is missing, when compared to EduSE Strategy Approach and EduSE Portal, as presented in Section III. SAKAI and Moodle are used in local or situated scenarios (course/faculty), and SWEnet presents no scientific protocol or methodology to expose and evaluate educational resources, i.e., leaning objects and experience reports. In this way, EduSE Approach aims to generate a business model to support continuous improvement in SE education, linking and motivating the involved stakeholders.

Future work consists of: (i) planning and executing a case study to verify EduSE Portal mechanisms to support EduSE Research Strategy phases 1-3; (ii) creating a framework for developing and managing learning objects and experience reports based on software reuse processes; (iii) finishing the infrastructure to support EduSE Research Strategy phases 4-5 and integration to Brechó-EcoSys. Thus, the researchers hope to contribute, improve and evolve SE education through the empirical paradigm and generate an educational ecosystem.

ACKNOWLEDGMENT

The authors thank CNPq for their financial support in Tec3ES Project – Technologies and Strategies in SE Education.

platform or market, and operating through the exchange of information, resources and artifacts [13].

REFERENCES

[1] ACM & IEEE. 2004. *The Joint Task Force on Computing Curricula – Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering*. The Computing Curricula Series.

[2] ADL. 2011. "Frequently Asked Questions about SCORM". At: <http://www.adlnet.gov/Documents/SCORM%20FAQ.aspx>.

[3] Alexander, M. & Beatty, J. 2008. "Effective Design and Use of Requirements Engineering Training Games". In: *Requirements Engineering Education and Training*, Barcelona, Spain, 18-21.

[4] Allen, D. 200. *Seam in Action*, 1ª ed. Manning, 625p.

[5] Basili, V.R., Shull, F., & Lanubile, F. 1999. "Building Knowledge through Families of Experiments". *IEEE Transactions on Software Engineering* 25, 4 (July), 456-473.

[6] Biolchini, J.C.A. *et al.* 2007. "Scientific Research Ontology to Support Systematic Review in Software Engineering". *Advances Engineering Informatics* 21, 2 (April), 133-151.

[7] Bourque, P. & Dupuis, R., eds. 2004. *Guide to the Software Engineering Body of Knowledge: 2004 Version*. IEEE Computer Society.

[8] Dantas, A.R., Barros, M.O. & Werner, C.M.L. 2004. "A Simulation-Based Game for Project Management Experiential Learning". In: 16th *SEKE International Conference*, Banff, Alberta, Canada, 19-24.

[9] Hiebert, J., Gallimore, R. & Stigler, J.W. 2002. "A Knowledge Base for the Teaching Profession: What Would It Look Like and How Can We Get One?". *Educational Research* 31, 5 (June/July), 3-15.

[10] IEEE CS. 2008. *Certified Software Development Professional*.

[11] IEEE P1484.12.2/D1. 2002. *Draft Standard for Learning Object Metadata*. IEEE LTSC WG-12.

[12] ISO/IEC. 2008. *ISO/IEC 12207 Systems and Software Engineering – Software Life Cycle Processes*. ISO, Geneve.

[13] Jansen, S., Finkelstein, A., & Brinkkemper, S. 2009. "A Sense of Community: A Research Agenda for Software Ecosystems". In: *31st ICSE – New and Emerging Research*, Vancouver, Canada, 187-190.

[14] Lethbridge, T.C. *et al.* 2007. "Improving Software Practice through Education: Challenges and Future Trends". In: 29th *ICSE, The Future of SE*, Minneapolis, MN, USA, 12-28.

[15] Meyer, B. 2001. "Software Engineering in the Academy". *IEEE Computer* 34, 5 (May), 28-35.

[16] Moodle. 2011. "Moodle: Open Source Community Based Tools for Learning". At: <http://moodle.org/>.

[17] Mustaro, P.N., *et al.* 2007. "Structure of Storyboard for Interactive Learning Objects Development". In: Koohang, A. & Harman, K. (eds.) *Learning Objects and Instructional Design*, Informing Science, 253-279.

[18] Sakai. 2011. "Sakai Project". At: <http://sakaiproject.org/portal>.

[19] Santos, R., Santos, P., Werner, C. & Travassos, G. 2008. "An Strategy to Support the Brazilian SE Educational Research", In: *5th Experimental SE LatinAmerican Workshop*, Salvador, BA, Brazil, 1-10. *In Portuguese*.

[20] Santos, R.P. & Werner, C.M.L. 2011. "Brechó-EcoSys: From a Component Library to a Software Ecosystems Platform". In: 12th *ICSR*, Demos Session, Pohang, Korea. To Appear.

[21] Schots, M., Santos, R., Mendonça, A. & Werner, C. A Survey to Characterize Brazilian SE Education Scenario". In: *II Brazilian Forum on SE Education*, Fortaleza, CE, Brazil, 57-60. *In Portuguese.*

[22] Shull, F., Carver, J. & Travassos, G. 2001. "An Empirical Methodology for Introducing Software Processes", In: *Joint 8th Experiences and Case Studies and 9th ACM SIGSOFT FSE-9*, 288-296.

[23] Spínola, R., Dias-Neto, A. & Travassos, G. "Developing Software Technologies through Experimentation: Experiences from the Battlefield", In: *XIII Iberoamerican Conference on Software Engineering*, Cuenca, Ecuador.

[24] SWEnet. 2011. "The Network Community for Software Engineering Education". At: <http://www.swenet.org/>.

[25] Todorova, M. & Petrova, V. "Learning Objects". In: International Conference on Computer Systems and Technologies: e-Learning, Sofia, Bulgaria, 697-702.

[26] Wöhlin, C. *et al.* 2000. *Experimentation on Software Engineering: An Introduction*. The Kluwer International Series in Software Engineering.

Structuring Software Engineering Case Studies to Cover Multiple Perspectives

Emil Börjesson and Robert Feldt
Software Engineering and Technology
Chalmers University
Gothenburg, Sweden
Emil.Borjesson@Chalmers.se

***Abstract*—Case studies are used in software engineering (SE) research for detailed study of phenomena in their real-world context. There are guidelines listing important factors to consider when designing case studies, but there is a lack of advice on how to structure the collected information and ensure its breadth. Without considering multiple perspectives, such as business and organization, there is a risk that too few perspectives are covered.**

The objective of this paper is to develop a framework to give structure and ensure breadth of a SE case study.

For an analysis of the verification and validation practices of a Swedish software company we developed an analytical framework based on two dimensions. The matrix spanned by the dimensions (perspective and time) helped structure data collection and connect different findings. A six-step process was defined to adapt and execute the framework at the company and we exemplify its use and describe its perceived advantages and disadvantages.

The framework simplified the analysis and gave a broader understanding of the studied practices but there is a trade-off with the depth of the results, making the framework more suitable for explorative, open-ended studies.

***Index Terms*—Case Study, Multi-perspective, Framework, Empirical**

Version 1, May 5, 2011.

I. Introduction

The case study is an observational research method used in many different fields of research due to its flexibility and its ability to investigate a phenomenon in its context [1]. Case studies are also applicable when there is no clear distinction between the phenomena and its context. This is particularly true in empirical software engineering research where there are many factors that impact the phenomenon, such as the type and organization of the company, the development processes used etc. To understand a contemporary phenomenon we also need to understand its history and how the different factors have evolved over time.

Existing advice for empirical research in software engineering focus on experiments and systematic reviews while guidelines for case studies was only recently published by Runeson and Höst [1]. There, a high quality case study is defined as a study that produces valid information of academic or industrial significance, either generic or practitioner oriented [1]. A key criteria for achieving this is that the data is collected in a planned and consistent manner and that conclusions are based on a clear chain of evidence. This can be a challenge in practice since case studies are typically flexible research designs with multiple sources of evidence; it is not clear-cut how to find the right balance between a flexible research design that allows multiple factors and causes to be taken into account while providing enough structure and support for planning and analysis.

Software engineering is different from computer science in that it takes more perspectives than only the technical into account. For example the personality or motivation of the engineers [2], [3] can affect the quality of their work and organizational and career considerations can affect important activities such as effort estimation [4], [5]. In general, the characteristics not only of the people but of the organization, business and processes used in software development are all important.

In recent empirical research in industry we were faced with designing a case study to describe and understand the verification and validation practices at a Swedish company developing safety-critical software systems. From initial talks at the company we understood that the practices could not be studied in isolation; they were heavily tied to the whole context of the company as well as how they had evolved over time. To structure our data collection and understanding we based our case study on an analytical matrix combining these two main aspects. On one dimension we wanted to cover at least the main perspectives of the BAPO framework with its four aspects [6]: Business, Architecture (technical aspects), Processes and Organization. The other dimension was time detailed in three steps as Past, Current and Future. Together this created a matrix of 12 different sub areas to be considered during the case study. We also defined a general six-step process to design, collect and analyze the findings of the study and adapted it to the company. This paper describes this analytical framework, called the BAPO/PCF framework for software engineering case studies, covering both the matrix and the process to adapt and use it. We exemplify and evaluate the framework based on our application of it at the studied company. In particular this paper addresses the following research questions:

1) What are the perceived advantages of a multi-perspective approach to conducting a case study?

2) What are the perceived disadvantages of a multi-perspective approach to conducting a case study?
3) How does the multi-perspective structure of the design affect research validity?

The definition of a research design is in this paper a methodology used to conduct a research project, such as a case study, an experiment, a longitudinal research project, etc. A research method is defined as a way of eliciting information within a research project; hence a research design can include the use of one or several research methods.

To help the reader differentiate between what was done in the state of practice analysis and what are general case study practices in the paper, we refer to the latter as the 'case study' and the former as either 'company study' or 'study at the company'.

The paper is divided as follows, section II will present the research context in which the research design was developed and executed. Section III will present how the framework was developed and executed. In Section IV the advantages and disadvantages of the design will be discussed, and finally some conclusions will be presented in Section V.

II. Research Context and the studied Company

The project, for which the research design was developed, was a state-of-practice analysis of a small company, less than 50 employees, which develop safety critical software applications. The state-of-practice analysis constituted the first part of a larger project, with the goal of improving the company's verification and validation practices. The analysis was conducted in order to understand the company's needs, regarding the company's processes, organization, etc, to narrow the scope of the process improvement effort.

The company conducts software development in bespoke projects to single customers, but the end applications are developed from a set of core products. These products are maintained according to market demand, making the company's business strategy a mix of bespoke and market driven engineering. Because of the nature of the software the company is developing, the company's business is governed by different quality standards and frameworks. These standards affect the development, but primarily the company's verification and validation practices. Standards and frameworks are also imposed on the company by the company's customers, who have different needs in their own domains. These demands require the company to be flexible in their development, which has resulted in the adoption of iterative as well as an agile software development based on Scrum [7]. The architectural granularity of the company's systems is on a sub-system level, hence coarse grained, but efforts are made to increase the granularity by refactoring the systems with reusable components based on services, so called service oriented architectures (SOA). The decision to change the system architectures was taken internally at the company but driven by external business factors related to a large European project that will change much of the company's market domain. The migration from sub-systems to SOA has required the company to acquire both new knowledge as well as new practices. Organizational changes have also been made, including changing the responsibilities of different roles and how communication is handled internally at the company. The company study was conducted over approximately 6 months, during which 7 structured interviews, 2 surveys, 1 structured observation, and a considerable amount of hours were spent on document analysis and watercooler discussion.

III. The BAPO/PCF Framework

The term research process is in this paper defined as the six-step methodology that was used to develop the case study design used at the company as well as the execution of the design. Figure 1 presents the process steps and how they incrementally relate to one another.

Get Knowledge about the Domain. Very little was known about the company when the project started and therefore the first step of the process was to gain a deeper understanding about the company and the company context. This information was acquired through a combination of interviews with people at the company and through literature review of internal documentation as well as documentation about the company's domain. The context in which the company operates was important to understand the goals and needs of the company, in order to align the improvement effort with these goals, such as process changes or introduction of new tools, etc. The initial interviews also served as a way of finding new sources by using snowballing, sources that could be used later during the company study. Snowballing is conducted by asking questions at the end of interviews and looking at document references to locate further artefacts to study [8]. The majority of the domain documentation that was studied was market quality standards and frameworks. Development and quality standards were important to study because the results of the company study would be used as a base for a larger project that focused on the company's verification and validation (V&V) practices, hence changes done to the V&V practices would, other than to comply with company needs, also have to comply with such standards. Domain specific documentation is often hard for someone from outside the domain to understand, so therefore all interpretations and question marks related to the analyzed domain documentation had to be verified with people at the company.

The collected data from step 1 showed that a broader view would be required to capture the state-of-practice at the company in its context. To meet this requirement and structure the research a multi-perspective framework was developed based around an analytical matrix with two dimensions. The first dimension was chosen to represent the perspectives of the company that would be analyzed, defined as company Business, Software Architectures, Processes and Organization (BAPO). These perspectives were chosen because they are used within academic and industrial frameworks such as the Family evaluation framework, which is used to evaluate Software Product Lines [6]. The second dimension of the matrix was chosen to represent time, defined as the Past, Current and

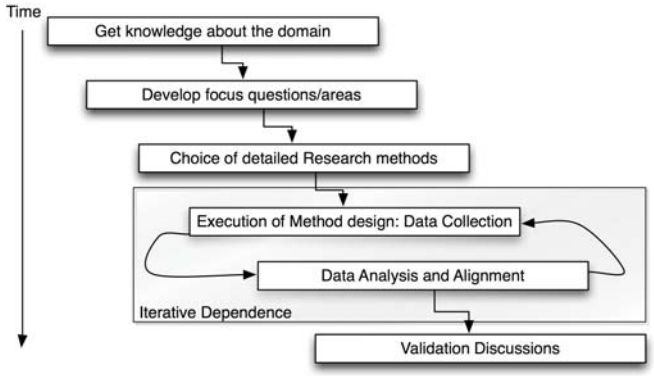


Fig. 1. Research process

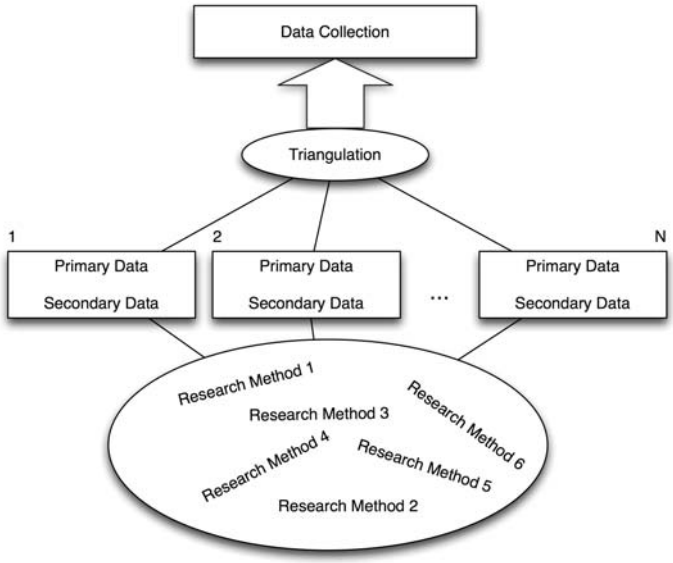


Fig. 2. General triangulation method

Future (PCF), since a chronological dimension of the elicited information was perceived to enable deeper and more precise conclusions to be drawn from the research findings. Hence the matrix starts with past Business, moving on to current business and so on, until finally reaching future organization. The BAPO/PCF framework was also chosen because it was perceived to be a good fit in this particular study, which might not be the case in the context of another company.

Develop focus Questions/Areas. The second step of the process was to use the information that was gathered in the first step, combined with the BAPO/PCF framework, to narrow down the focus of the company study by developing concrete research questions. The research questions were split up among the 12 sub areas/cells of the analytical matrix, which gave an initial understanding of where within the company to elicit information to answer the research questions. For instance it gave a coarse-grained view of which roles within the company that should be interviewed and what documentation to analyze. Information about existing company roles and documentation was provided by the first step of the process. The research questions were split among the 12 matrix cells based on the researchers own opinion on where they fit best, with the company context taken into consideration. For instance questions regarding the company's agile development processes were mostly constricted to the process row of the matrix, only sorted in time, whilst questions about the organization were split up between business, processes and organization in this case. The reason for the spread of organizational questions came from the need to understand how the organization had been affected by the other company aspects, for instance how the introduction of agile processes had affected the organization, or what affect business changes had resulted in, how roles had changed, etc. Some questions could not be confined to just one cell, such questions were mostly of higher level, but they helped to find logical connections between cells within the matrix. The options in these cases were to add the question in several of the cells, or to redefine them to make each of the questions unique and only fit in a specific cell.

Choice of Detailed Research Methods. The third step of the process was to choose what research methods that would be used in the company study. Primarily qualitative methods were chosen, such as interviews, visual inspection, structured observation and water-cooler discussions. Water-cooler discussions are conducted during coffee and lunch-breaks and make use of the fact that most information sharing within development companies are conducted during breaks [9]. This allows the researcher to get qualitative data that was current and truthful, but since the information often included the persons own opinions the information had to be verified to remove bias. Surveys were also used to provide quantitative results that would give depth to the research results. The surveys were deductive, hence based on previously elicited information, to verify the collected results and therefore conducted later in the company study. Different research methods are more appropriate to use depending on what sources are available, and therefore it is good to have a set of methods to choose from [1]. Creating a plan of what methods to use in what cells is suitable, but it should be considered a guideline that can be changed depending on the conditions of the study rather than a strict plan.

Data collection and Data analysis and Alignment. The first three steps of the process constitute the development of the design, whilst the following three steps of the process discuss how to execute the design. The fourth and fifth steps are defined as data collection and data analysis and alignment, and were executed incrementally, meaning that collected data was continuously analysed and aligned with previously collected information within the BAPO/PCF matrix, before new information was elicited and analyzed. Such analysis and alignment became essential since the data collection governed how the research would proceed, meaning that the collected information was used to evaluate if a certain research question had been answered or not. If the question had been answered

the research could move on to a new cell, otherwise further time was spent to find more information.

Case studies require flexibility to be able to respond to different events that may occur during the course of the study, such as for instance sudden research opportunities. For instance during the company study an opportunity arose to do observation of the company's system testing practices that would not arise again during the course of the planned study period. Because events, such as opportunities, have to be considered the research has to be flexible and can not follow a strict research plan. The BAPO/PCF framework does however support such flexibility, since the research questions connected to the BAPO/PCF matrix cells can be answered out of order, which allows the researcher to jump between cells and work with the question that is the most urgent at the moment. The progress of the research can also be measured through what research questions within the matrix have been answered. Each cell that gets filled with information, and its research question answered, constitutes a measurable part of the study progress, and once all questions in all cells have been answered the study is complete. The time spent with a certain matrix cell can either be planned according to a fixed time budget, or in a more ad hoc manner where the collected results drive the time spent working with a cell as well as the order which the matrix cells are traversed. There are academic papers that state that research using the BAPO perspectives should start with the business aspect because this has the largest impact on the company, followed by software architectures that has the second largest impact and so on until the organizational aspect has finally been analyzed [6]. No evidence could be found to support this claim during the company study, which used an approach where intermediate results decided what cell to jump to next in the study. The study at the company did not follow the matrix row by row, but instead company processes were first investigated, followed by software architectures, followed by company business and finally company organization. The BAPO/PCF matrix traversal was not chosen at random, it was chosen based on information that became available in the initial part of the forth process step and proved throughout the study to be valuable asset to keep the research focused. Even though several deviations were made from the research plan, because opportunities arose, it proved to be valuable to have a plan to fall back on, which is also supported by Runeson and Höst that state that a plan is crucial for a case study to succeed [1].

To strengthen the analysis of the collected information, in the company study, cause-effect chains (CEC) were combined with the BAPO/PCF matrix. These event chains link the research results together to provide a broader contextual view and also help to find the causes behind a result. Two different types of CEC's were used during the company study, where the first type focuses on one path through the matrix, the simplest being from a certain perspective's past to the same perspective's current or from the current to the future, hence a chronological view of to the result. The first type of a CEC can also span between different company perspectives

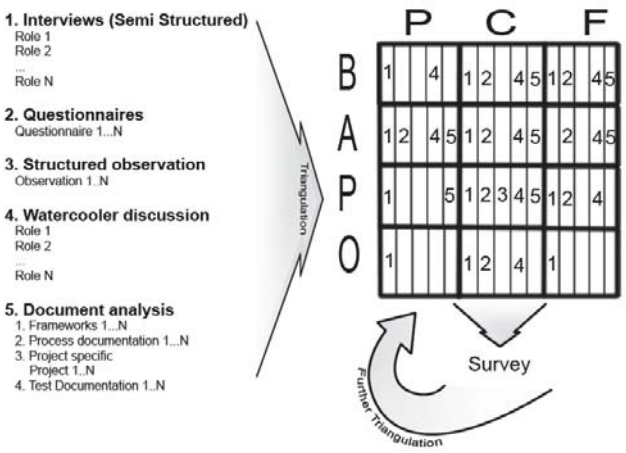


Fig. 3. Research design triangulation

to provide a cross-perspective view, but always form a single path through the BAPO/PCF matrix that shows how an event in one perspective in time has affected another perspective in time and so on. The second CEC type differs from the first type in the sense that it starts in one matrix cell and spans out to several cells both in time and between perspectives, which shows the broader impact an event has had within the company. In the company study the CEC's were developed post-elicitation of the research information by using inspection to find the internal connections between the results, which made it possible to draw clearer conclusions, develop predictions about the future of each company perspective, and also raise the result validity. Predictions for the future perspective of the matrix were developed using induction, with qualitative research data as input, based on the concept that if a specific action A in context B has outcome C in the past, which is repeated in the current in the same context B with result C, it would be likely that the same pattern would reoccur also in the future. A visual representation of the CEC's was also developed that uses the research results, described in bullet-point lists in the matrix cells, which are connected by drawing arrows from the origin matrix cell to the cell(s) that have been impacted by the event in the origin cell. It should be noted that a CEC can never find a connection between the current to the past, or from the future to the current, but if there seems to be a connection backwards in time it might be possible to create another CEC that was previously overlooked. An example of how to visualize the second type of a CEC is presented in Figure 4, which shows how the introduction of unit-testing impacted the studied company in a larger context, which could have been overlooked if the scope of the research design had been narrower. The general reasoning behind CEC's has also been visualized in Figure 5.

The BAPO/PCF matrix can also be used as a tool to visualize the research result quality. Several techniques were used to ensure data validity during the company study, but the primary technique was triangulation [1], [10], which states

that in order for information to be valid it must be verified by at least three sources. These sources can be either artefacts or roles, for instance documentation, different roles at the company, etc, or results gathered by using several research methods, such as interviews, literature review, etc. The matrix can be used to show how triangulation was performed during a study by visualizing which sources were used to answer what research question within the BAPO/PCF matrix. This provides a graphical overview of the data validity that is easy to understand, also by non-researchers like for instance the managers of the company under study. The general triangulation technique is presented in Figure 2, and the visualization of what sources that were used for triangulation, in the company study, is presented in Figure 3. Another technique that was used to ensure data validity during this step of the process, in the company study, was to send transcripts of interviews, preliminary documentation analysis reports, etc, to different roles within the studied company for validation. This technique provides the researcher with fast feedback if the elicited information has been correctly interpreted or not, which is valuable in cases where the information includes contextual jargon unknown to the researcher.

Validation Discussion. The final step of the process was to validate the research results with the studied company. This was partially done in step four and five since those steps deal with information validation through the means of company review, but those reviews were on a information level, whilst the final validation was on a conclusion level to ensure that the conclusions drawn from the research information were reasonable and correct. The final validation can be done in different ways, like for instance writing a report that the company can review, or the results can be presented orally. In the company study the results were presented orally during a power-point presentation where the analytical matrix was once more used. All significant results from the study were broken down into bullet-points in the matrix cells and added to a power-point slide. The graphical representation of the results gave the audience a clear overview of the results, which could afterwards be discussed further in more detail. Errors or discrepancies in the results were finally analyzed further and rewritten in a final study report.

IV. Discussion

The perceived advantages of applying the BAPO/PCF framework for case studies are related to the structure and the broadness of the multi-perspective view of the framework. By applying the multi-perspective approach to a case study it becomes possible to collect results specific to the different perspectives of the company, as well as in time. This provides a contextual structure that makes it clear where and how to acquire information to answer the research questions, but also how to find information that can link the results together. For instance by pointing out a role within the studied company that has knowledge about several perspectives, the information gained from this knowledge can be used to draw cross-perspective conclusions about the cause of a particular

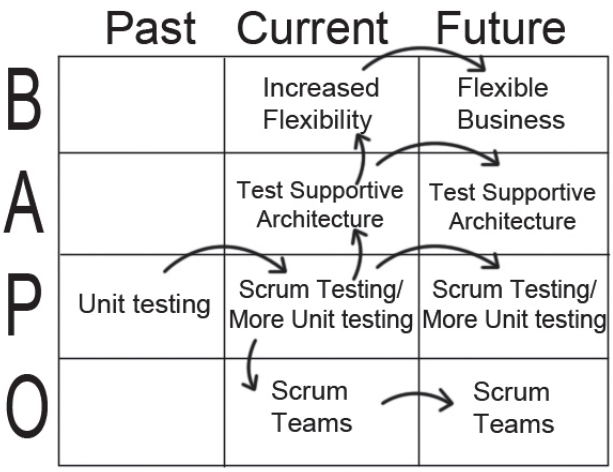


Fig. 4. Cause-effect chain example

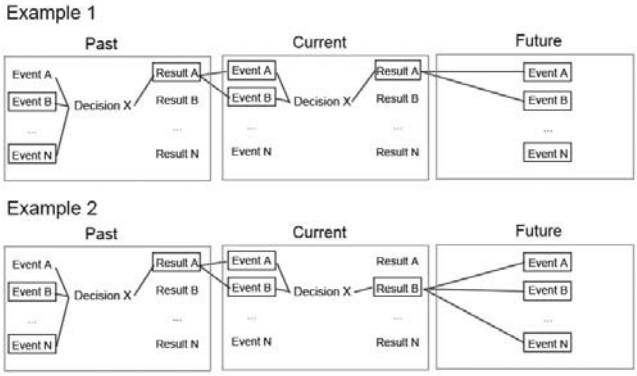


Fig. 5. General Cause-effect chain structure

result. By looking at the entire matrix, including these cross-perspective results, longer chains of cause and effect can also be drawn that provides a deeper as well as broader undestanding of results and of the company. Internal validity is also improved with this approach since as described by Wohlin et al. 2000 [11], a factor that is investigated because of its effects on another factor may itself be affected by a third factor, and if the third factor is overlooked there may be a threat to internal validity. Hence the BAPO/PCF framework improves the internal validity by giving the researcher the means of finding these connections between different factors that could have otherwise been overlooked with a narrower scope. It is however important to recognize that this framework, like other case studies, does not solve the issue regarding to what extent a factor within a given perspective affects another factor in another given perspective. A change within a company is seldom localized to a certain perspective of the company but rather has ripple effects to several different perspectives. An example from the company study is how changes to a company's business goals resulted in the introduction of new processes, new architectural development methods as well as

organizational change.

Other advantages that are side effects of the BAPO/PCF framework's structure include support for project planning, visualization of the validation through triangulation, as well as to visualize the research results. The framework supports research plans that are flexible and allow events, such as research opportunities, to be taken advantage of as they arise during the study. Visualization of the triangulation provides an overview of the validation effort and also makes it easy for non-researchers to see how the results have been validated, for instance making it easier to trust results and conclusions that are previously unknown to the studied company. Result visualization has similar advantages in terms of providing an overview of the collected results, and help to find connections between the collected information.

As for general case studies this design does not limit what research methods that are applicable for acquiring information, but unlike many other case study designs that rely on purely qualitative methods for data elicitation and analysis, this design allows longitudinal design concepts to be used as well, which are quantitative. This is made possible by the chronological axis of the matrix that links the past to the future and therefore allows quantitative metrics to be developed. An example of such a metric could be organizational change over time, i.e. the company's growth rate. Such information can also be used deductively to draw more plausible predictions about the future, for instance how the growth of the company will continue or decrease.

The design does provide context to the research results but by broadening the research it becomes necessary to sacrifice information depth in projects with fixed budgets. Hence one of the BAPO/PCF framwork's greatest strengths is also its primary weakness. There is a trade-off that must be made when using the framework, which states that in order to gain information depth in one perspective, information depth will have to be sacrificed in another perspective. The consequences of this trade-off has not been investigated in this study, and since the design has only been used during one empirical case this subject is still open to speculation. The broadness of the design does however make it more suitable for research with open ended questions where very little or nothing is known about the phenomenon under study, rather than research with narrow research questions of a more deductive nature.

The results and conclusions that were developed during the company study proved to have high validity, however since the design has only been used in one study it is uncertain if the same results would be achieved in another study. The design is flexible in many ways, including the core concept of the analytical matrix, but once again, since only the BAPO/PCF configuration has been tested nothing can be said about another configuration even though it can be speculated that another configuration could be more beneficial in another company context.

V. Conclusions

This paper introduced the BAPO/PCF framework for structuring case studies in software engineering and ensuring they cover multiple perspectives. The framework was evaluated in a case study of verification and validation practices in a Swedish software company.

The combination of the BAPO (Business, Architecture/technical, Process and Organization) and PCF (Past, Current and Future) dimensions resulted in 12 sub areas to consider in designing and executing the case study. The matrix allowed the research questions to be connected to the research effort, as well as providing the researcher with tools for result visualization, project planning and result validation. The structure also allowed an analysis of cause-effect chains across perspectives and in time providing a broader understanding and increased validity. Most importantly the framework helped uncover issues and connections the company themselves were not aware off.

The largest perceived disadvantage is that the approach can become too broad and therefore require considerable effort to cover all 12 sub areas. This can be addressed by sacrificing depth of analysis within less prioritized sub areas but this needs further research.

In summary, a flexible yet powerful case study research design can be created by adding structure through the use of an analytical matrix and a simple process to adapt it to the context being studied. The analytical matrix used, based on BAPO and PCF, can be of general value for such software engineering research.

References

[1] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Software Engineering*, vol. 14, no. 2, pp. 131–164, 2009.

[2] H. Sharp, N. Baddoo, S. Beecham, T. Hall, and H. Robinson, "Models of motivation in software engineering," *Information and Software Technology*, vol. 51, no. 1, pp. 219–233, 2009.

[3] R. Feldt, L. Angelis, R. Torkar, and M. Samuelsson, "Links between the personalities, views and attitudes of software engineers," *Information and Software Technology*, vol. 52, pp. 611–624, June 2010. [Online]. Available: http://dx.doi.org/10.1016/j.infsof.2010.01.001

[4] A. L. Lederer and J. Prasad, "The Validation of a Political Model of Information Systems Development Cost Estimating," in *Proceedings of the 1991 Conference on SIGCPR*, ser. SIGCPR '91. New York, NY, USA: ACM, 1991, pp. 164–173.

[5] A. Magazinius, S. Börjesson, and R. Feldt, "Exploring Intentional Distortions in Software Cost Estimation," *In submission*, 2011.

[6] F. Van Der Linden, J. Bosch, E. Kamsties, K. Känsälä, and H. Obbink, "Software product family evaluation," *Software Product Lines*, pp. 107–109, 2004.

[7] K. Schwaber *et al.*, "Scrum development process," in *OOPSLA Business Object Design and Implementation Workshop*, vol. 27. Citeseer, 1995, pp. 10–19.

[8] R. Moriarty and J. Bateson, "Exploring complex decision making units: A new approach," *Journal of Marketing Research*, vol. 19, no. 2, pp. 182–191, 1982.

[9] E. Börjesson, "Multi-Perspective Analysis of Software Development: a method and an Industrial Case Study."

[10] M. Patton, "Enhancing the quality and credibility of qualitative analysis." *Health Services Research*, vol. 34, no. 5 Pt 2, p. 1189, 1999.

[11] C. Wohlin, P. Runeson, and M. Höst, *Experimentation in software engineering: an introduction*. Springer Netherlands, 2000.

Usability evaluation: a survey of software development organizations

Ardito C., Buono P., Caivano D., Costabile M.F., Lanzilotti R.
Dipartimento di Informatica
Università di Bari Aldo Moro
Bari, Italy
{ardito, buono, caivano, costabile, lanzilotti}@di.uniba.it

Bruun A., Stage J.
Department of Computer Science
Aalborg University
Aalborg, Denmark
{bruun, jans}@cs.aau.dk

***Abstract*—The importance of usability engineering in software development is acknowledged by an increasing number of software organizations. This paper reports from a survey of the practical impact of usability engineering in software development organizations. The survey was conducted in Southern Italy, replicating one conducted in Northern Denmark three years earlier. The results show that the number of organizations conducting some form of usability activities is nearly the same, but there are important differences in the understanding of usability. The key advantages emphasized by the respondents are product quality, user satisfaction and competitiveness in both surveys. The main problems emphasized are developer mindset, resource demands and customer participation.**

Usability engineering; software development; survey.

I. INTRODUCTION AND BACKGROUND

It is now widely acknowledged that usability engineering is a key area in software development. Since the beginning of '90 (see [1]), various methods for usability engineering have been developed and assessed; an overview is in [2]. There is also a considerable amount of research of the relevance of these methods in software development practice. Evaluating software systems for usability has been documented to be economically beneficial in terms of increased sales [3], increased user productivity [4], reduced training costs [5], and decreased needs for user support [6].

Many software organizations devote a substantial and increasing amount of resources on determining the right functionality of their products. Yet there are numerous examples of systems that fail despite having the right functionality, simply because the prospective users cannot use the system for its intended purpose [7]. A problematic or incomprehensible user interface is a typical source of such problems. Usability is a measure of the extent to which prospective users are able to apply a system in their activities [8]. A low level of usability means that users cannot work out how to use a system, no matter how elaborate its functionality is [1].

Despite the fact that documented benefits of usability engineering exist and software developers state that usability is even more important than functionality [9], the literature provides studies that show software development organizations have limited or no usability engineering activities. Thus, it seems as if the benefits have little impact on software development practice. This has been shown not only in studies performed in the '90 (e.g. [10]), but also in more recent ones. While many authors have discussed the benefits and challenges of usability engineering in general, fewer have studied specific software development organizations in order to understand the reasons of the limited impact of usability engineering in practice. A case study of software organizations identified obstacles to increased deployment of usability engineering methods [11]. Another study found that usability is interpreted differently by different actors, depending on their formal roles, and they often experience more focus on efficiency and economy [12]. In a study of the way usability evaluations are conducted in practice, a significant gap between this practice and the literature was identified [13].

In [14], a survey has been conducted in order to identify possible obstacles that prevent organizations to actually take usability into account in their software development practices. The study involved 39 software development organizations, located in a region of Denmark. It consisted of a questionnaire survey, whose aim was to determine whether software development organizations were evaluating the usability of their software and to identify key obstacles.

A limitation of the study performed in Denmark (we refer to it as "DK-Study" in the remaining of the paper), as the authors clearly said, is that it was conducted in a limited geographical area in Northern Europe. This motivated us to replicate the study in a completely different area, namely a region in Southern Italy (we refer to it as "Ita-Study").

The paper has the following organization Section II describes design and execution of the Ita-Study, while Section III reports its results. Section IV discusses and compares results of Ita-Study and DK-Study. Section V provides conclusions and indicates future work.

II. ITA-STUDY: DESIGN AND EXECUTION

The study described in this paper replicates the one in [14] by involving organizations located in the Apulia region in Southern Italy. It aims at investigating the current usability engineering practices in software development organizations.

A. Method

As first step in the study, we conducted an online questionnaire survey using the same questionnaire of the DK-Study. Special care has been devoted to avoid any influence of the results of the DK-Study in analysis of the collected data and in results interpretation. Then, a comparison with the results of the DK-Study has been performed.

B. Participants

In order to be consistent with the DK-Study, we considered organizations with the following characteristics: a) they develop software with a graphical user interface, (e.g. web applications, mobile applications); b) they develop for customers or for internal use; c) they are located in the Apulia region in Southern Italy (University of Bari is in this region); d) they employ more than a single person.

In order to find the organizations, we used search engines on the Web, our own knowledge of organizations, and we looked at the list of organizations of the DAISY-net consortium (www.daisy-net.com), which brings together ICT (Information and Communication Technology) companies in the Apulia region. The considered organizations have their headquarter located in Apulia, even if most of them work in the whole Italy. The product types of most organizations are distributed between web sites, Internet banking, data warehouse, mobile applications, and business management software.

C. Data collection

Data from the selected organizations have been collected through an online questionnaire. The questionnaire used in the DK-Study, which was written in Danish, was translated to English by its authors. Then, the Italian researchers translated it to Italian; finally, a Danish native speaker, who has lived and worked in Italy for 20 years, checked the Italian version with the original Danish questionnaire. The questionnaire, available at "http://tinyurl.com/questaziende", contains open and closed questions, aiming at acquiring information about: 1) the organizations (number of employees, product types, platforms, development method); 2) their understanding of the term "usability evaluation"; 3) organization's use, experiences with and opinions about usability evaluation.

For each selected organization, we identified a contact person, who was called by phone to explain the purpose of the study and to invite her/his organization to participate in it. The contact person indicated the more appropriate person in the organization who could participate in our study.

All the forty-six contacted organizations agreed to fill in the online questionnaire. An email containing the link to the questionnaire was sent a few days later. twenty-seven filled in the questionnaire in few days; the remaining nineteen have been solicited again. Ten organizations did not respond. At the end, we got thirty-six questionnaires.

D. Data analysis

The collected data have been analyzed by three Italian HCI researchers. In order to avoid influences from the DK-Study, they had no knowledge of that study. However, they have been asked to adopt the same methodology. For the closed questions they have made a quantitative analysis. For the open questions they have followed the grounded theory approach: they have individually analyzed the data from each of the open questions and have put codes on sentences. Then, the code for each sentence has been discussed among the three of them, and a single code has been agreed upon. They have individually assigned codes to categories. Again, the individually assignment of codes to categories has been agreed upon in a joint session. This process has resulted in a list of categories and codes, which has been used to get a condensed overview of the results from the questionnaire.

III. ITA-STUDY: RESULTS

This section reports the results obtained by the questionnaire.

A. Understanding of "usability evaluation"

The main important questions the survey addresses are whether the organizations perform usability evaluations and what possible problems and/or advantages they find. Preliminary to this, it is necessary to understand what the respondents actually mean by usability evaluation. Thus a first significant question was "describe what you understand by the concept *usability evaluation*".

The thirty-six answers have been coded in five categories: Evaluation of usability, User involvement, Usability definition, Accessibility test, Do not know. As shown in Figure 1, thirteen respondents have provided an acceptable explanation of usability evaluation. For example, one said: "Usability evaluation is a process to verify that a software product has the following features: short learning time, fast task execution, low error rate, easy to remember commands, high user satisfaction". Another said that usability evaluation "is a set of measures obtained from activities performed to evaluate effectiveness, efficiency, ease of use, etc., of the products that the company produces and/or uses". Five respondents explicitly referred to usability evaluation as a test or a process involving end users (User involvement category); an example is the following answer: "Usability evaluation is a field test, which involves end users who are the target of the system, as well as the customer". We consider this category different from the previous one since we want to highlight the fact that these respondents are not using other usability evaluation methods that do not involve end users, e.g. inspections and analytical methods. Fifteen respondents provided a more or less acceptable definition of usability, thus indicating that they are aware of what usability is, but not of how actually to evaluate it. One organization put more emphasis on accessibility than usability; this can be explained by considering that accessibility of software for public institutions is forced in Italy by a specific

law of year 2003, called "Stanca Law", after the politician (Stanca) who proposed it. Finally, two respondents explicitly said that they do not know what usability evaluation is ("Do not know" category).

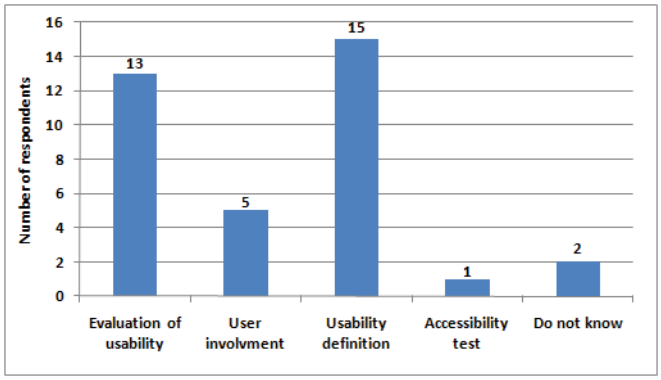


Figure 1. Respondents' understanding of "usability evaluation".

B. *Deployment of usability evaluation*

In order to avoid misinterpretations that would compromise the possibility to compare the answers of all respondents in the following questions, the following two definitions of usability evaluation were provided: 1) the analysis of the user interface to identify interaction problems through methods performed by usability experts, with the possible involvement of users chosen from the system's target group; 2) the use of methods to check if the system (and any other possible user guidance) is: easy to learn, easy to remember, effective to use, satisfactory to use and to understand".

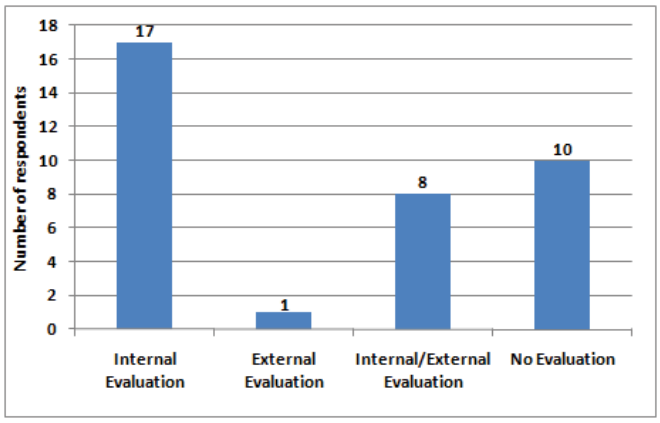


Figure 2. Type of usability evaluation performed by the respondent organizations.

With reference to these definitions, it was asked if the organization performs usability evaluations. Twenty-six respondents out of thirty-six said that their organization performs usability evaluation. As reported in Figure 2, seventeen respondents say that their organization performs usability evaluations internally (i.e. with its own personnel), one performs external evaluations (i.e. performed by external consultants), eight organizations perform both internal and external evaluations. Finally, ten organizations do not execute any usability evaluation, even if four of them say that software developers are asked to take into account usability principles in their work.

C. *Problems in usability evaluation*

The twenty-six respondents who said that their organization performs usability evaluations have been asked to report the problems they encountered in introducing and performing usability evaluations in their software life cycle. Answers to this open question have been classified in the following categories, as reported in Figure 3: No suitable methods, Resource demands, User availability, Developer mindset, No problems, Do not know. Some respondents indicated more than one problem.

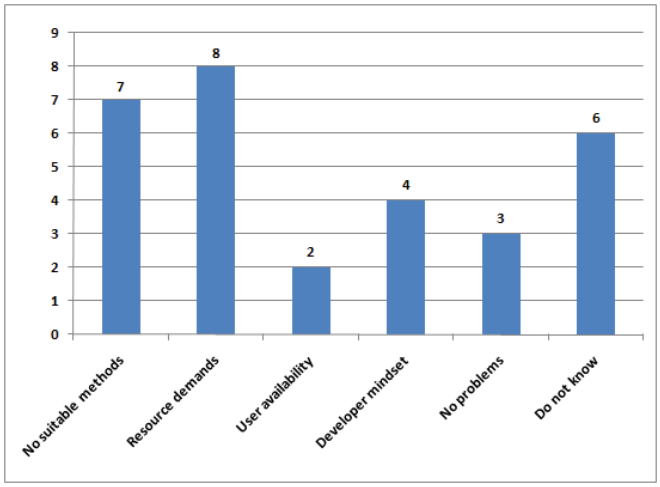


Figure 3. Problems in usability evaluations, as reported in the questionnaires.

No suitable methods. Seven respondents indicated the lack of suitable methods for usability evaluation as one of the main problems. One answer was "Lack of agile methodologies for the evaluation". Another said: "There is no standard for usability evaluation. Each customer has his own specific reality with problems of different types, and this prevents to identify standard methods and metrics which are applicable".

Resource demands. Eight respondents said another main obstacle to performing usability evaluations is that they require a lot of resources in terms of cost, time, and involved people. More specifically, one clearly said: "There are many problems of time and resource demands"; another said: "There is a conflict between time required by evaluations and the need of resource optimization that the organization has". Other answers addressed the cost for designing and performing usability evaluations, and the problems to allocate time and personnel to the evaluations.

User availability. As shown in Figure 1, some respondents think that usability evaluation necessarily involves users. Thus, it is not surprising that a problem in carrying out usability evaluation is user availability. Indeed, it is well known that it is not easy to involve users in the evaluations. This is one of the main reasons why methods that are not user-based, e.g. analytical methods and inspections, are often employed in usability evaluations. It is worth mentioning the following answer: "… it is difficult to get the time and resources

necessary to perform usability tests from the customer". It remarks another well known misunderstanding of people working in industry, who often refer to customers and not to real end users. Interesting is the answer of another respondent, who highlights the difficulty, as well as the importance, of involving real users, since "… user testing in a lab often does not provide reliable results".

Developer mindset. This category refers to problems due to the fact that some professional developers still have their main interest on functionality and efficiency of the code, i.e. on qualities of a software system which are of great interest for developers but have no impact on end users. The HCI researchers who analyzed the questionnaire answers proposed different names for this category. At the end we decided to call this category Developer mindset for analogy to the DK-Study. One respondent clearly said that it is difficult to consider usability among the software quality they address in their development process. He also mentioned that they do not allocate specific budget for usability evaluation: we considered this a resource demand problem. It is worth remarking that another respondent explicitly said that "usability evaluation is considered sometimes a waste of time".

No problems. Three respondents have explicitly answered that they did not find any problem. We cannot say whether this means lack of motivation in filling the questionnaire or they actually did not remember any problem.

Do not know. In this category we have grouped six respondents: four have explicitly said that they do not know about problems (one of them said that he is not the person in the company directly involved in performing evaluations); the other two did not say anything explicit about problems.

D. *Advantages of usability evaluation*

Next question for the respondents who said that their organization performs usability evaluations asked to report the advantages they got in performing usability evaluations. Answers to this open question have been classified in the following categories, as reported in Figure 4: Quality improvement, User satisfaction, Competitiveness, Resource saving, No advantages, Do not know. Some respondents indicated more than one advantage.

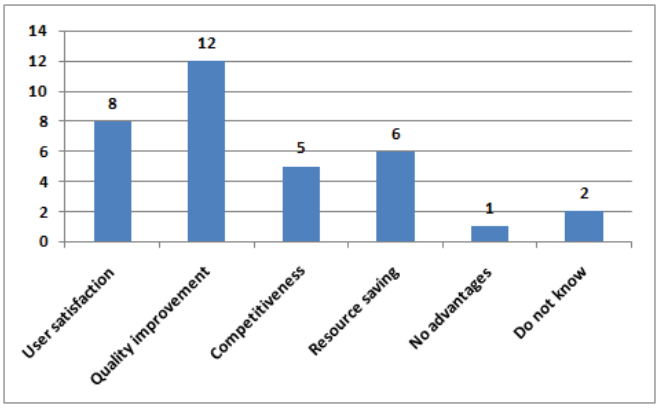


Figure 4. Advantages of usability evaluation, as reported in the questionnaires.

Quality improvement. Most answers pointed out that an advantage of usability evaluation is quality improvement of the developed products. These are some answers: "We can offer a better product to the end user"; "For sure, the product becomes closer to users' needs and expectations"; "The final product is more complete"; "[with usability evaluation] user interfaces keep improving, balancing user requirements and application complexity". A more articulated answer is: "We can increase the involvement of the end user and customer in the development process. It is a cost, since requirements are often completely modified after a usability evaluation. However, the quality of the final product improves, since the customer is active in the development process and that will produce software that is really useful and usable". As we can see, this respondent actually believes that usability evaluation will greatly improve software quality; the consideration about the cost is mainly due to the fact that, as it is well known, when usability evaluation is performed too late in the product life cycle, often it shows many problems deriving from not adequate requirements analysis, and, at this point, redesign would have a high cost.

User satisfaction. Eight respondents explicitly indicated user satisfaction as another advantage of usability evaluation. One said: "Greater satisfaction of final users, especially of regular users". This shows that the organization cares about customer fidelity and is aware that usability evaluation may help in better satisfying its customers. Other respondents indicated that usability evaluation helps ensuring that the final product better meets user needs and foster product acceptability and involvement of final users.

Competitiveness. Five respondents addressed an important objective of organizations, namely competitiveness on the market. In their opinion, usability evaluation helps increasing organization competitiveness as well as customers' appreciation. Moreover, one respondent explicitly mentioned sales increase as an advantage. Another said: "We can improve our business performances". Other mentioned advantages are: "Software will be used by many more users"; "To ttract new customers and improve relationships with current ones".

Resource saving. Despite the fact that resource demand was considered a problem by some respondents (see Section IV C), six answers report resource saving as advantages of usability evaluation. Those people pointed out that performing usability evaluation at right time in the product life-cycle actually reduces overall costs. One respondent said: "The risk to allocate resources for developing products that will have problems in their use is reduced". Another said: "Development costs decrease, thus saving time and money". Other two respondents pointed out cost saving during maintenance and evolution phase after product release, reducing the need of revisions.

No advantages. One respondent explicitly answered that usability evaluations do not give any advantage. Again, it is impossible to say if he does not see any advantage or if he did not pay enough attention to the questionnaire.

Do not know. Two respondents explicitly said that they do not know about advantages since they are not directly involved in performing evaluations.

E. *Organizations not performing usability evaluation*

Six of the ten organizations that do not carry out usability evaluations have a number of employees ranging from 60 to 200, three have about 50 employees and the last one is a small company with four employees. The structure of the questionnaire was such that, when the respondent answered that his/her organization does not perform usability evaluation, four more questions were presented, the first one was an open question asking to explain the motivations for not performing usability evaluation. One respondent addressed a resource demand problem: "I think that there are not economic and temporal margins to do this type of evaluation". Several respondents reported problems that can be considered in the Developer mindset category. For example, one said: "… our current development process does not include usability evaluation and we do not have expertise for performing them". Another said: "Our company does not have personnel allocated to these activities. I do not know why". A third one mentioned: "Lack of adequate culture". Another respondent provided a more articulated answer: "… many of our products are customizations of existing systems already on the market; in such cases, our approach is to align them to the logic of use of the existing product". This answer shows a clear misunderstanding of the concept of usability and of the need of usability evaluation.

One respondent also focused on the future when saying: "Currently our need [of usability evaluation] is still in its infancy. It's very likely that in the near future metrics and measurements to assess usability during the development of a product will be available". Another respondent said: "Even if in our company there is not a team allocated to usability evaluation, people developing software and performing requirement analysis have to take into account usability principles in their work".

The final three questions for a person who said that his/her organization does not perform usability evaluation were closed questions. The first one asked: "Has it been considered to introduce usability evaluation in your organization, possibly by demanding it from external consultants?". Of the total of ten respondents, three answered Yes, two answered No, and five said Do not know. The second closed question asked: "If the problems that prevent your organization from performing usability evaluations could be minimized, would your organization be eager to perform usability evaluation?". Four respondents said Yes, six said Do not know. Finally, the third question was: "Do you think that the products of your organization could improve with usability evaluation?". Eight respondents said Yes, two said Do not know.

IV. DISCUSSION AND COMPARISON WITH DK-STUDY

Looking at the first important question in the submitted questionnaire, namely the understanding of the term "usability evaluation", only two respondents said that they do not know its meaning, and one emphasized accessibility evaluation rather than usability evaluation. All the other thirty-three respondents either provided answers that refer to methods to evaluate usability or gave a correct definition of usability (fifteen respondents) without mentioning methods to evaluate it. The answers provided to the same question in the DK-Study, which has been performed in 2007, show that thirteen respondents actually addressed functionality tests instead of usability evaluation. This indicates that organizations are actually becoming more aware of what usability really is (our study has been conducted during November-December 2010), but either they are still reluctant to evaluate it in the software life cycle or they do not have a clear understanding of how to evaluate it.

The percentage of organizations that perform usability evaluation was also almost the same between the two studies. In the DK-Study thirty-nine organizations were considered: twenty-nine (74%) said that they perform usability evaluation, ten (26%) said that they do not; in Ita-Study, twenty-six (72%) do, ten (28%) do not.

By carefully looking at the other answers provided in the Ita-Study by the twenty-six organizations who said they do usability evaluation, we did not find any indication of problems due to the used evaluation methods: none mentioned difficulties when performing a specific method, when gathering information during tests, etc. This makes us question how carefully usability evaluation is performed by such organizations. Moreover, the answers to the open question about the background of the persons performing usability evaluation indicated that they have the Italian equivalent of Master degrees in Computer Science or in Computer Engineering. In most cases, such persons, especially if they do not belong to younger generations, have very limited knowledge about human factors (HCI courses only recently have been introduced in such curricula).

The results from the DK-Study revealed that the organizations not performing usability evaluations are aware that they would improve their products, but the obstacles are greater than the expected advantages. When the ten organizations in the Ita-Study that do not perform usability evaluation, were asked if they are considering to introduce usability evaluation, only three said Yes. This pushes usability researchers and practitioners to deeply consider how to change this situation, devoting more attention on how to transfer academic work into practical value for industry. It is responsibility of academics to translate scientific articles, which formally describe evaluation methods, into something that makes sense for companies and it is ready to be applied.

Comparing the problems in introducing and performing usability evaluation, only some category of problems were common to both studies, namely Resource demand and Developer mindset. In the DK-Study, the most frequently mentioned obstacle was "Developer mindset", i.e. many developers have their minds set more on programming aspects, technical challenges and functionality of the product than its usability. The second highest obstacle was the felt "Resource demand" of usability evaluation. Another mentioned obstacle was "Customer participation", which refers to the difficulties to get customers to participate in usability engineering activities. Overall, it emerged that many software organizations lacked practical knowledge about the way usability engineering activities should be conducted. The fact that in the DK-Study there are twelve problems in the category Developer mindset, while in the Ita-Study there are seven, is a further indication

that, as time goes by, software developers are changing their mind, becoming more aware of what usability is and of the importance of usability evaluation.

The DK-Study identified two further categories of problems, called Test participants and Customer participation; the first one refers to the difficulties in finding and motivating users to participate in usability tests; the second one refers to the difficulties to convince customers of the value of usability evaluations and getting them to participate actively in a project. In the Ita-Study, none mentioned the importance of involving customers in the project. Two respondents mentioned the difficulties in finding users to be involved in usability tests. Thus the category User involvement in the Ita-Study is equivalent to the category Test participants in the DK-Study. The last category in the DK-Study is Conducting test, which refers to problems regarding the way tests are conducted. None of the respondents mentioned similar problems in the Ita-Study; instead, another category in this latter study is No suitable methods, since the lack of methods suitable to be used in industry was indicated as a problem by seven respondents.

Considering advantages in conducting usability evaluations, in the Ita-Study "Quality improvements" was mentioned most frequently, followed by "User satisfaction". The DK-Study revealed an almost identical tendency with respect to categories and their ordering. In that study "System improvements" was mentioned by sixteen respondents followed by ten mentioning "Customer satisfaction"; five named "Marketing value", i.e. the equivalent of Competitiveness. However, the category "Resource saving" found in the Ita-Study was not found in the DK-Study.

V. CONCLUSION

This paper reports from a survey of the practical impact of usability engineering in software development organizations. The survey was conducted in Southern Italy during November-December 2010. It replicated a survey conducted in Northern Denmark three years earlier. The results show differences in the understanding of usability, where the concept is better understood by the Italian respondents. This shows that, as time goes by, software developers are becoming more aware of what usability is and of the importance of usability evaluation. The amount of organizations conducting some form of usability activities is nearly the same. The key advantages emphasized by the respondents are product quality, user satisfaction and competitiveness in both surveys. The problems emphasized differed more, with more practical problems emphasized in the DK-Study.

We are planning to complement these studies with more in-depth understanding of the advantages and problems of usability engineering as they are perceived by the individual organizations. Moreover, it would be worth focusing on software developers who are motivated to increase the usability of the products they develop. The key question to be addressed is why developers who are clearly motivated do not increase the adoption of usability engineering methods in their development processes. Once more data will be collected, it should be useful to analyze the correlation between organization characteristics (size, sector, process in use, activities performed etc.) and the adoption of usability engineering methods. Finally, the use of ethnographic methods could provide another means to get an in-depth understanding of the socio-technological realities surrounding everyday software development practice [15], thus providing other hints on how to overcome obstacles to a wider account for usability engineering.

ACKNOWLEDGMENT

The research reported in this paper has been partially supported by: 1) EU COST Action IC 0904 TWINTIDE (Towards the Integration of Transectorial IT Design and Evaluation); 2) DIPIS (DIstributed Production as Innovative System) Project sponsored by Apulia Region; and 3) WPU (Web Portal Usability) financed by the Danish Research Councils (grant number 2106-04-0022).

REFERENCES

[1] J. Nielsen, Usability Engineering, Morgan Kaufmann, 1993.

[2] D. Scapin and E. Law, "Proceedings of Mause International Workshop on Review, Report and Refine Usability Evaluation Methods (R3-UEM)," 2007.

[3] C.M. Karat, "A comparison of user interface evaluation methods," John Wiley & Sons, Inc., 1994, pp. 203-233.

[4] C.M. Karat, "Cost-justifying usability engineering in the software life cycle.," Handbook of Human-Computer Interaction, M. Helander, T. K. Landauer and P. Prabhu, eds., Elsevier, 1997, pp. 767-778.

[5] S.M. Dray and C.M. Karat, "Human factors cost justification for an internal development project," Academic Press, Inc., 1994, pp. 111-122.

[6] S. Reed, "Who defines usability? You do!," PC Computing, vol. Dec, 1992 pp. 220-232.

[7] J.D. Gould and C. Lewis, "Designing for usability: key principles and what designers think," Commun. ACM, vol. 28, no. 3, 1985, pp. 300-311.

[8] J. Rubin, Handbook of Usability Testing: How to Plan, Design, and Conduct Effective Tests, Wiley, 1994.

[9] M. Frese and W. Hesse, "The work situation in software-development " Methods & Tools, 1995.

[10] S. Rosenbaum, S. Bloomer, D. Rinehart, J. Rohn, K. Dye, J. Humburg, J. Nielsen and D. Wixon, "What makes strategic usability fail?: lessons learned from the field," ACM, 1999, pp. 93-94.

[11] I. Boivie, C. Aaborg, J. Persson and M. Lofberg, "Why usability gets lost or usability in in-house software development," Interacting with Computers, vol. 15, no. 4, 2003, pp. 623-639.

[12] A. Cajander, J. Gulliksen and I. Boivie, "Management perspectives on usability in a public authority: a case study," Proc. Nordic conference on Human-computer interaction (NordiCHI 2006), ACM, 2006, pp. 38-47.

[13] M. Nørgaard and K. Hornbæk, "What do usability evaluators do in practice?: an explorative study of think-aloud testing," Proc. Conference on Designing Interactive systems (DIS '06), ACM, 2006, pp. 209-218.

[14] J.O. Bak, K. Nguyen, P. Risgaard and J. Stage, "Obstacles to usability evaluation in practice: a survey of software development organizations," Proc. Nordic conference on Human-computer interaction (NordiCHI 2008), ACM, 2008, pp. 23-32.

[15] H. Sharp, C. deSouza and Y. Dittrich, "Using ethnographic methods in software engineering research," ACM, 2010, pp. 491-492.

Maximizing the Financial Benefits Yielded by IT Projects While Ensuring their Strategic Fit

Antonio Juarez Alencar, Gustavo Taveira, Eber Assis Schmitz and Angelica Dias
Institute of Mathematics and Eletronic Computer Center
Federal University of Rio de Janeiro
Rio de Janeiro, Brazil

Alexandre Correa
Center for Exact Sciences and Informatics
State of Rio de Janeiro Federal University
Rio de Janeiro, Brazil

Abstract— **As competition for market space continues to increase worldwide and the information technology (IT) budget of many organizations these days comprises hundreds of projects running concurrently across different business functions, decision making units and geographical locations, selecting IT projects for investment has become a challenge in itself, not to mention the need to maximize the benefits yielded by these projects. This paper presents an optimization model, based upon balanced scorecard and minimum marketable feature modules, that makes it easier for management to maximize the financial benefits yielded by a portfolio of IT projects at the same time that ensures the strategic fit and reduces both the risk and the need for capital investment. Furthermore, the application of the model is exemplified with the help of a real world inspired example. Besides, the potential benefits to business are also analyzed.**

Keywords- IT Portfolio Management, Economics of Software Engineering, Balanced Scorecard, IT Investment Analysis, Minimum Marketable Features and Strategic Fit.

I. INTRODUCTION

From a versatile programmable apparatus that makes processes to run faster and yield more reliable results, information technology (IT) grew into a transformation enabling tool capable of improving business performance and, in many cases, providing competitive advantages [1, 2]. Not surprisingly, IT became a central piece in the business strategy of organizations of different types, sizes and culture [3, 4].

Moreover, as competition for market space increased, IT became one of the main sources of capital investment across many lines of business. As a consequence, the portfolio of IT projects of any middle or large sized organization comprises hundreds of projects running simultaneously across a variety of lines of business, business functions, decision making units, geographies and having a diverse number of aspects to be considered about their customer, such as: occupation, gender, age and socio-economic status. Therefore, selecting the projects that are to be part of an organization's portfolio is frequently a challenge in itself [5].

In addition, each organization has its own strategy defined by many differently-thinking stakeholders and, if it wants to remain competitive in the long run, these projects are better to be aligned with this intended business strategy, so that all the available resources are consistently applied to achieve what is indeed relevant to business [6].

However, managers of any IT-related project that requires capital investment are expected to provide proper returns and are exposed to risk, let alone a large number of projects grouped into a portfolio. As such, IT-portfolio management should be treated with the same due diligent rigor as any other capital investment, in which stakeholders are often interested in keeping risk exposure under control, at the same time that the return on investment is maximized [7].

Despite the relatively extensive literature on IT portfolio management, little has been said on how to maximize the financial benefits of an IT portfolio while ensuring the strategic fit [8, 9].

This article goes towards filling this gap by presenting an optimization model that enable managers to maximize the financial benefits yielded by a portfolio of IT projects, at the same time they reduce risk, curtail the need for investment capital, and ensure the strategic fit. The use of the model, which is based upon balanced scorecard and minimum marketable feature modules, is exemplified with the help of a real world inspired example. Also, the potential benefits yielded by the optimization model to business are analyzed.

The remainder of this paper is organized as follows. Section 2 contains the conceptual framework the paper is based upon. Section 3 discusses how the benefits of a portfolio of IT projects can be maximized while ensuring its strategic fit. Section 4 exemplifies the application of the model using a real world inspired example. Finally, Section 5 presents the conclusions of this paper.

II. CONCEPTUAL FRAMEWORK

A. Minimum Marketable Features Modules

According to [10], the buying process of any products and services by consumers has four main components:

- There is a perceived need that urges to be satisfied;
- There is at least one product or service in the market that is believed to satisfy that need;
- Consumers have enough buying power to pay for the product or service; and
- Consumers are willing to pay what they have being asked for.

While many simple products and services have a reduced and relatively rigid set of features for which consumers are willing to pay, software most often has a large collection of such features that may be easily grouped into different sets [11].

According to [12-14], minimum marketable features (MMFs) are self-contained software units comprising features that have an intrinsic market value. These units create value to business in one or several of the following areas:

- Competitive differentiation - it allows the creation of service or products that are different from anything else being offered in the market;
- Revenue generation - it provides an extra value by offering the same quality as other products in the market for a better price;
- Cost savings - it saves money by making business processes cheaper to run;
- Brand projection - it promotes certain characteristics that make the service or product more attractive to customers; and
- Enhance loyalty - it influences customers to buy more, more often, or both.

Moreover, the total value brought to an organization by a software project consists of several interdependent MMFs, each one with its own development cost, precedence restrictions, financial and non-financial benefits.

Although an MMF is a self-contained software unit, it is often the case that it can only be delivered after other project parts have been completed. These parts may be either other MMFs or the architectural infrastructure, i.e. the set of basic features that offer no direct value to customers and, as a result, for which they are not willing to pay, but that are required by the MMFs [14].

The architectural elements, or AEs for short, can also be decomposed into self-contained deliverables, enabling the underlying architecture to be delivered ``on demand'', further reducing the initial investment of a project [13].

Fig. 1 presents the precedence graph of a software project consisting of a set of MMFs and AEs. Note that A, B, C, D are module building activities and that an arrow going from A to B, i.e. A → B, indicates that the work on module A has to be

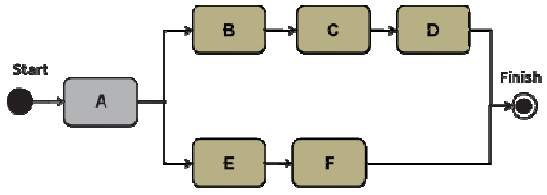


Fig. 1. The precedence graph of a set of MMFs and AEs.

completed before the work on module B can start.

No software provides revenues forever. After some time, market forces will inevitably make any software obsolete and promote the need to replace it by a more up-to-date and financially attractive alternative. The time elapsed between the beginning of a software development effort and its replacement is often referred to as *software lifetime* or, alternatively, as the *software window of opportunity*.

Table 1 shows the investment required by each module introduced in Fig. 1, together with its respective flow of net returns, considering a window of opportunity of ten periods. In this case both the investment required for the development effort and the flow of returns are accessed in thousands of US dollars.

TABLE I. REQUIRED INVESTMENT AND NET RETURNS YIELDED BY A SET OF AES AND MMFS (US$ 1.000).

Period	Software Modules					
	A	***B***	***C***	***D***	***E***	***F***
1	-80	-50	-40	-20	-50	-60
2	0	45	60	35	50	30
3	0	45	60	35	50	30
4	0	45	60	35	50	30
5	0	45	60	35	50	30
6	0	45	60	35	50	30
7	0	45	60	35	50	30
8	0	45	60	35	50	30
9	0	45	60	35	50	30
10	0	45	60	35	50	30

For instance, while ***B*** requires an initial investment of US$ 50K, it generates a net revenue of US$ 45K per period for a total of ten periods, once completed. Module ***F***, on the other hand, requires a higher investment of US$ 60K and generates a net revenue of US$ 30K per period.

Module ***A*** is the only one that yields no return, despite requiring the largest initial investment among all modules and, therefore, is an AE. As the remaining modules do provide some return on the investment required to their development, they are considered to be MMFs.

Because it is improper to perform mathematical operations on monetary values without taking into account an interest rate, in order to compare the business value of different MMFs and AEs, one has to resort to their discounted cash-flow [15]. Table 2 shows the sum of the discounted cash-flow of each module introduced in Fig. 1, considering an internal rate of return of 2% per period.

TABLE II. DISCOUNTED CASH-FLOW CONSIDERING THE PERIOD IN WHICH THEIR RESPECTIVE DEVELOPMENT STARTS (US$ 1.000).

Period	Software Modules					
	A	***B***	***C***	***D***	***E***	***F***
1	-78	311	441	260	351	181
2	-77	269	384	227	304	154
3	-75	227	328	195	258	126
4	-74	187	274	163	213	100
5	-72	147	220	131	168	74
6	-71	108	167	101	125	48
7	-70	69	116	70	82	23
8	-68	32	65	41	40	-1
9	-67	-5	16	12	-1	-26
10	-66	-41	-33	-16	-41	-49

Such a sum is the net present value (NPV) of all cash flow elements of a module considering that its development starts at period n ∈ [1..10]. For instance, according to the information

presented in Table 2, if ***B*** is developed in the first period, it yields an NPV of US$ 311K, i.e.

$$NPV(B,1) = -50 + \frac{45}{\left(1+\frac{2}{100}\right)^1} + \frac{45}{\left(1+\frac{2}{100}\right)^2} + \cdots + \frac{45}{\left(1+\frac{2}{100}\right)^9} = 311 \quad (1)$$

If ***B*** is developed in the second period, it yields an NPV of US$ 269K, in the third, US$ 227K and so on and so forth.

Obviously, not all modules can be developed in the first period. The precedence graph presented in Fig. 1 indicates that only ***A*** can be developed at that time. Because in this example each module requires exactly one period to be developed, ***C*** cannot be developed until the third period. Furthermore, each particular implementation order yields its own return on investment. For instance, sequence ***ADBCEF*** yields an NPV of US$ 866K, i.e.

$$NPV(ADBCEF) = NPV(A,1) + NPV(D,2) + \cdots + NPV(F,6) = 866 \quad (2)$$

while sequence ***ABDCEF*** yields US$ 875K.

Although MMF identification and ordering is an area that would benefit from further investigation, [12] shows that Function-Class Decomposition can be successfully applied to identify MMFs and AEs in a software project [16]. Moreover, both [13] and [17] provide the necessary means to find the implementation sequence that maximizes business value.

According to [18] the advantages of dividing a software project into MMFs are numerous:

- Large and complex systems can be developed from a relatively smaller investment,
- Return on investment in software development is maximized, together with return on investment in IT related areas and projects,
- Demand for shorter investment periods and payback time are addressed,
- Favor faster time-to-market of software and also of products that depends upon software development,
- Bring financial discipline into the software development, and
- Position software development process as a value creation activity in which business analysis is an integral part of it.

B. Balanced Scorecard

Although it is generally accepted that one cannot manage what one cannot measure, even in this day and age decision makers rarely consider measurement as an essential part of their strategy [19]. As an example, it is not unusual that managers introduce an innovative technology in their working environment with the intention of bursting performance and, without second thoughts, keep using the same short-term financial measures they have been using for years [20].

In the same way that there is no single medicine that can successfully tackle all illnesses, there is no single indicator that can properly report all business scenarios and activities. Therefore, when facing a new reality, one has always to question not only what measures provide better support for decision making, but also whether the old ones continue to be relevant [21].

Moreover, despite the best intentions that high management may have in making their strategy easy to understand and divulge among employees, goals such as "becoming the preferred supplier of our customers", "providing the best products to the market" and "being widely recognized as a top quality organization", which are so common in organizations' missions and strategic visions, do not translate easily into financial measures, not to mention terms that provide clear guidance to the efforts that are required at the operational level to have a strategy successfully implemented [22].

Therefore, if one wants that all the resources at the disposal of an organization are properly applied to achieve its strategic goals, one has to look beyond the classic financial measures for a more flexible set of indicators [20, 23].

Balanced scorecard, or BSC for short, is a strategic planning model and management methodology due to Kaplan and Norton [24]. It provides a multidimensional framework in which high level management can translate strategic goals into a coherent set of performance indicators that are easily understood by both middle management and operational personnel [25].

Since it has been firstly introduced in the early 1990s, the BSC has evolved from a performance measurement tool, designed to reveal and tackle the problematic areas within an organization [26], to a framework for implementing strategies and determining the strategic fit of an organization's human, information and organization capital [27].

The BSC model advocates that any business strategy should be translated into four distinct but related sets of performance indicators, i.e.

- *Financial* - reflecting how an organization is seen by their shareholders, both as a short-term and a long-term investment;
- *Customer* - describing how an organization is perceived by their customers, regarding their products, services, relationships and value-added;
- *Internal process* - revealing which aspects of business processes must improve and excel in order to satisfy shareholders and customers; and
- *Innovation and learning perspective* - showing how an organization can continue to evolve and create value.

A practical example of how a simplified version of a balanced scorecard might look like is presented in Fig. 2. Note that in a BSC table each perspective usually contains several

goals, whose achievements are signaled by one or more performance indicators (PIs) reaching pre-determined values by a deadline.

Perspective	Goals	Performance Indicator		
		Id	Description	Target (next 30 months)
Financial	Increase revenue	pi_1	Annual revenue	US$ 50 million
	Increase net margin	pi_2	Annual net margin	25%
Customer	Increase customer satisfaction with projects	pi_3	Quarterly survey mean satisfaction index	5 in a 1 to 7 scale
	Increase the percentageof retained customers	pi_4	Annual percentage of retained customers	20%
Internal Processes	Increase function points (FPs) delivered by employees	pi_5	Monthly average number of FPs delivered by employees	at least 20 points
	Decrease the number of nonproductive hours	pi_6	Monthly average percentage of nonproductive hours	15%
Innovation & Learning	Increase employees' technical skills	pi_7	Annual average number of training hours attended by employee	at least 80 hours
		pi_8	Percentage of certified project managers among employees	at least 20%
	Increase employees' job motivation	pi_9	Biannual survey mean motivation index	5 in a 1 to 7 scale

Fig. 2. A simplified version of an organization's balanced scorecard.

It is central to BSC's effectiveness that a causal relationship is established among the four set of PIs, or four perspectives, as they are most often referred to. Using the causal relationship perceived among perspectives, one is able to connect all of the non-financial measures ("drivers of the performance") to the financial measures ("final outcome"), showing the links by which specific improvements in the drivers are expected to lead to desired outcomes [28, 29].

In a well conceived BSC, the PIs in the learning and growth perspective are the drivers of the PIs of the internal processes and possibly some of the PIs of the remaining perspectives. Besides, the PIs of the internal processes perspective are expected to be the drivers of the PIs of the customer perspective and possibly some of the PIs of the financial perspective. Finally, the PIs of the customer perspective are intended to be the drivers of the financial PIs [30]. Fig. 3 summarizes these ideas.

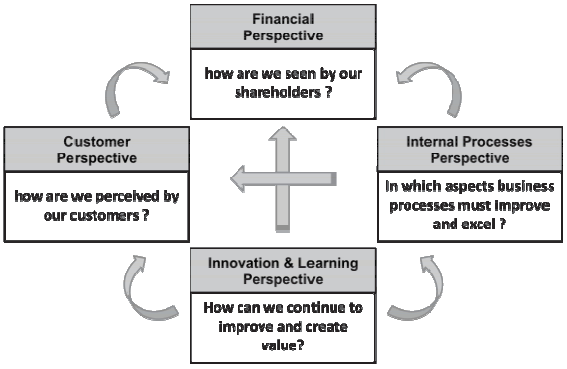


Fig. 3. A software organization balanced scorecard.

In the BSC model, this causal relationship among perspectives is expressed in a strategy map, which is a framework for linking intangible assets to shareholder value [21, 31]. Fig. 4 shows the strategy map that served as the basis for the construction of the BSC presented in Fig. 2.

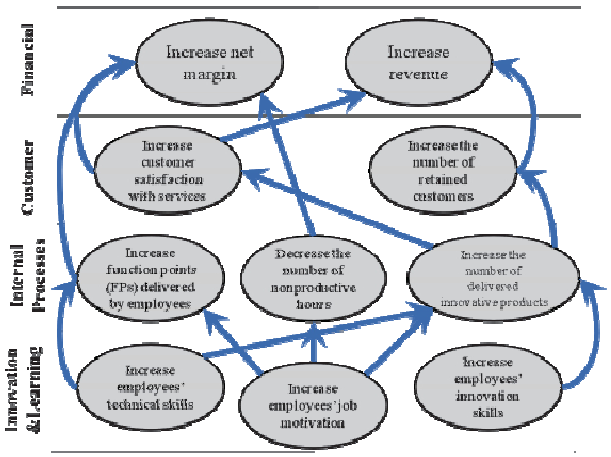


Fig. 4. The causal relationship among the four perspectives.

In a well conceived strategy map, the financial perspective is expected to describe tangible outcomes of a strategy in terms that are of interest to investors and shareholders, while the remaining perspectives define intangible value creation propositions that intends to generate those outcomes [20].

According to [20, 23, 30], when using the BSC model one should adhere to the following steps:

1. Revise the organization's mission and strategic vision;
2. Translate the strategic vision into a set of goals considering the financial, customer, internal processes, and innovation and learning perspectives;
3. Use a strategy map to identify the causal relationship that should exist among the goals in the four perspectives;
4. Determine the appropriate performance indicators to monitor each goal; and
5. Determine attainable but challenging target values for each performance indicator and a deadline for those targets to be reached.

Both the strategy map and the balanced score card should be revised on a regular basis as changes in market conditions may very well require adjustments in an organization's strategy, and consequently in its BSC model [32].

III. Maximizing the Appropriation of Financial Benefits

It is important to keep in mind that it is very unlikely that the target values in a BSC can be reached without further capital investment. Most often several interdependent projects have to be properly undertaken until those values can be matched, each one with its own capital requirement [33].

Once completed, each project either makes a direct contribution towards helping one or more PIs to reach their targeted values or enables other projects to make such a contribution. Anyway, when due, these contributions can be achieved at once or, most often, incrementally in the course of time. Besides, it is important to consider that all projects connected to a BSC may provide financial returns, even when related to non-financial perspectives.

Therefore, because for-profit organizations are business arrangements set up to provide return on investment to shareholders, when managing a portfolio of BSC related projects, one is interested in maximizing their financial return, while contributing as planned to reach stated goals by a pre-established deadline.

Among all the different kinds of projects that can be connected to a BSC, the IT based ones share the common property of being potentially factored out in MMFs and AEs. As a result, besides increasing return on investment, and shortening both the payback time and the time to market, these MMFs may also contribute to PIs so as their intended values can be reached.

A. Formalizing the Ideas

Since the portfolio $\boldsymbol{P}$ of projects connected to a BSC is usually composed of both IT and non-IT projects, it is helpful to define a function $\boldsymbol{f}$ that takes every project $\boldsymbol{p_i}$ in $\boldsymbol{P}$ to its corresponding set of MMFs and AEs.

If $\boldsymbol{p_i} \in \boldsymbol{P}$ is a non-IT project, than $\boldsymbol{f}$ takes $\boldsymbol{p_i}$ to an indivisible one and only MMF or AE, depending on whether $\boldsymbol{p_i}$ yields or not some financial return on the investment required for its development over the BSC *window of opportunity*.

In formal terms, a precedence graph like the one presented in Fig. 1 is a mathematical structure $\boldsymbol{G = (V, E)}$ composed of two sets, i.e.

- The set of vertices $\boldsymbol{V} = \{ v_1, v_2, \ldots, v_n \}$; and
- The set of edges $\boldsymbol{E} = \{ (v_i, v_j) \mid v_i, v_j \in V \}$.

Note that if $(v_i, v_j) \in \boldsymbol{E}$ than there is a dependency relation between vi e vj, such that if s and d are functions that respectively return the startingpoint and the duration of the development of a module,

$$(v_i, v_j) \in E \Rightarrow s(v_j) \geq s(v_i) + d(v_i);$$

A valid sequence $\boldsymbol{S} = v_j \cap v_k \cap \ldots \cap v_l$ is an ordered set of MMFs and AEs satisfyingthe following restrictions

- Every module in the sequence derives from the portfolio of projects connected to the BSC, i.e.

$$\forall \text{vi} \in \boldsymbol{S} \Rightarrow v_i \in p_k, \text{ onde } \{p_k \in \boldsymbol{P} \mid 1 \leq k \leq \#(\boldsymbol{P})\}$$

- Every module belonging to the sequence is listed exactly once, i.e.

$$v_i, v_j \in \boldsymbol{S} \text{ and } v_i = v_j \Rightarrow \text{i} = \text{j};$$

- The precedence relations between the modules listed in $\boldsymbol{S}$ must uphold at all times, i.e.

$$\forall \{v_i, v_j\} \in \boldsymbol{S} \text{ and } (v_i, v_j) \in \boldsymbol{E} \Rightarrow s(v_j) \geq s(v_i) + d(v_i);$$

- At any given time the total number of teams available to work on the development of the modules, i.e. *nteams*, cannot be exceeded, i.e.

$$\forall t_i \in \boldsymbol{T}, \#(\{v_k \in \boldsymbol{S} \mid s(v_k) = t_i\}) \leq nteams, \text{ where } \boldsymbol{T} \text{ is the BSC } window\ of\ opportunity;$$

- The development effort must start at period one, i.e.

$$\exists\, v_k \in \boldsymbol{S}, s(v_k) = t_0$$

Let $\boldsymbol{PI}$ be the BSC set of performance indicators. Also, let *current* and *target* be functions that return respectively the current and intended value for each $pi_i \in \boldsymbol{PI}$. Moreover, let *makespan* be a function that return the time necessary to implement all $v_i \in \boldsymbol{S}$. In these circumstances, one is interest in finding a valid sequence $\boldsymbol{S}$ that maximizes:

$$NPV(S) = \sum_{k=1}^{\#S} NPV\left(S(k), s\left(S(k)\right)\right) \quad (3)$$

where $\#\boldsymbol{S}$ and $S(k)$ are functions that return, respectively, the number of elements in $\boldsymbol{S}$ and the element in the k^{th} position, provided that the planned contribution of all project in $\boldsymbol{P}$ to reach the targeted values of performance indicators are fully realized within the BSC *window of opportunity*, i.e.

$$\forall pi_i \in \boldsymbol{PI}, current(pi_i) \geq \text{target}(pi_i) \text{ and } makespan(\boldsymbol{S}) \leq \boldsymbol{T} \quad (3)$$

IV. An Example

According to Benjamin Franklin (1706--1790), the American statesman, "a good example is the best sermon". Hence, the model presented in this paper is introduced with the help of a real-world inspired example.

A. Context Information

Consider a large mobile telecommunication carrier such as AT\&T, VERIZON, TELEFONICA and many others, which have millions of subscribers and provide different kinds of value-added services. For the purpose of this paper this organization is called World Mobile Telecom Corporation[1] or WMTC for short.

Technology advances in mobile telecommunications have brought significant changes in the services provided by mobile carriers around the world, creating opportunities and risks to which business must respond. As a result, WMTC and its competitors are fiercely struggling for any strategic advantages that would enable them to gain market-share and ultimately surpass each other.

[1]The data used to obtain the results presented in this paper is closely related to real data provided by one of the largest Latin America's mobile network operator. Because it reflects the current portfolio of on-going IT projects, the authors have been kindly requested no to disclose its name.

Moreover, a growing number of customers conscious of their bargaining power has been demanding for better products and services for reduced prices, so the need for innovation and differentiation is increasing rapidly.

Therefore, if WMTC wants to advance its position as a major player in the mobile telecommunication business, it must select the right business strategy, making the best possible use of its human and financial resources.

WMTC's mission statement tells that the company wants to be recognized as "a highly trustable organization that provides valuable mobile telecommunication services to its customers". In its strategic vision WMTC declares that "the value that the organization creates to its customers stems from innovative products and services that excite the imagination of mobile telecommunication users, fulfilling their needs and desires".

After analyzing the forces acting upon the markets in which it does business (consumer's behavior, competition, uprising technology, interest rates, etc.) and considering its own strengths and weaknesses, WMTC decided to encode its business strategy in the strategy map introduced in Fig. 4. A simplified version of the corresponding balanced scorecard is presented in Fig. 2.

To reach its goals, among other actions, WMTC has decided to invest in the development of five innovative IT-based value-added services, i.e.

- *Goods & Services Purchase (GSP)* – which allows subscribers to use their mobiles as a credit cards to pay for goods and services in associated stores;
- *Mobile Entertainment Pass (MEP)* – which lets subscribers to search for movies, plays and shows, browse their synopsis and buy tickets directly from their mobile phones;
- *Global Navigator (GN)* – which enables subscribers to determine where they are, where others belonging to the same group of related people are, and to draw routes between any two points in a map;
- *Universal Translator (UT)* – which makes it possible for subscribers to communicate in a number of foreign languages using their mobile keyboard, microphone and speakers; and
- *Voice of the Customer (VoC)* – which let customers to compete for prizes by timely reporting on their buying experience, information that is crucial to perfecting the services of customer-oriented organizations.

B. Identifying MMFs and AEs

To take advantage of Denne and Cleland-Huang's ideas on minimum marketable features [13], the projects within the WMTC portfolio have being decomposed into AEs and MMFs using Function-class Decomposition [16], so that they can be incrementally delivered as "mini-projects". Tables 3, 4, 5, 6 e 7 introduce the AEs and MMFs identified by the WMTC's Project Management Office (PMO).

TABLE III. GOODS & SERVICES PURCHASE'S MMFS AND AES.

Id	Type	Name	Description
GSP_1	AE	Service Subscription	Allows customers to subscribe to and unsubscribe
GSP_2	AE	Credit Analisys	Figures the likelihood of a customer paying a debt
GSP_3	AE	M-Payment	Lets customers pay for goods and services they want to buy
GSP_4	AE	Refund	Allows customers to be refunded when returning goods and services they bought
GSP_5	MMF	Shopping	Entitles customers to shop for goods and services in associates stores

TABLE IV. MOBILE ENTERTAINMENT PASS' MMFS AND AES.

Id	Type	Name	Description
EP_1	AE	Service Subscription	Allows customers to subscribe to and unsubscribe
EP_2	AE	Credit Analisys	Figures the likelihood of a customer paying a debt
EP_3	MMF	Search movie	Allows customers to search for movies, plays and shows based on multiple criteria such as: title, cast, genre, district, etc
EP_4	MMF	Browse synopsis	Make it possible for customer to browse the synopses of movies, plays and shows
EP_5	MMF	Buy ticket	Allows customers to choose which specific seat they would like to purchase, charges the ticket value to the customer's account and provides a queue avoiding electronic ticket via SMS
EP_6	AE	Capture location	identifies the current geographical location of a customer
EP_7	MMF	Browse nearby places	Provides information on all theaters and showrooms in the vicinity of the current customer location

TABLE V. GLOBAL NAVIGATOR'S MMFS AND AES.

Id	Type	Name	Description
GN_1	AE	Service Subscription	Allows customers to subscribe to and unsubscribe
GN_2	AE	Credit Analisys	Figures the likelihood of a customer paying a debt
GN_3	MMF	Where am I?	Allows customers to know their geographic location
GN_4	MMF	Navigator	Lets customers select the best route between two points
GN_5	AE	Group Membership	Allows customers to create a group, apply for membership of existing groups and cancel an existing memberships
GN_6	MMF	Find my friends	Reveals the whereabouts of people belonging to the same group of acquaints

TABLE VI. UNIVERSAL TRANSLATOR'S MMFS AND AES.

Id	Type	Name	Description
UT_1	AE	Service Subscription	Allows customers to subscribe to and unsubscribe
UT_2	AE	Credit Analisys	Figures the likelihood of a customer paying a debt
UT_3	MMF	Basic language set	Lets customers to communicate in a reduced set of commonly used languages: English, French, Spanish and Portuguese
UT_4	MMF	Extended set	Provides an extended set of languages

TABLE VII. VOICE OF THE CUSTOMER'S MMFS AND AES.

Id	Type	Name	Description
VoC_1	AE	Service Subscription	Allows customers to subscribe to and unsubscribe
VoC_2	AE	Credit Analisys	Figures the likelihood of a customer paying a debt
VoC_3	MMF	VoC	Allows subscribers to freely express their opinion about ser- vices and products they have purchased
VoC_4	AE	Membership information	Enables customers to check upon the membership points they have earned and prizes they can win
VoC_5	MMF	Prize redemption	Lets customers exchange their membership points by the prizes of their choice

C. Establishing the precedence between MMFs and AEs

One of the main benefits of organizing MMFs and AEs in a portfolio is the possibility of more easily identifying common software units [34]. By factoring out such units one may reduce the portfolio's need for capital investment, increase profit and speed up time-to-market, at the same time that increases quality [35].

Analyzing the portfolio, the PMO identified the "Service Subscription" and the "Credit Analysis" as units that are common to two or more projects. From now on, these units are referred to as $\boldsymbol{CSU_1}$ and $\boldsymbol{CSU_2}$, respectively. Fig. 5 introduces the WMTC's portfolio precedence.

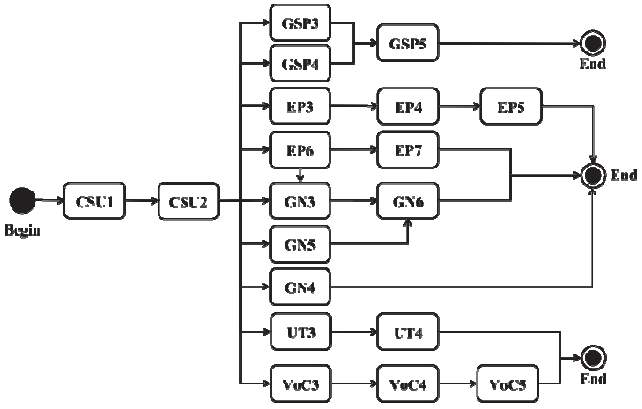


Fig. 5. WMTC's portfolio precedence graph.

D. Estimating the contribution of each MMFs and AEs

Once the AEs and MMFs have been identified and arranged into a precedence graph, their corresponding cost and revenues have to be estimated. Table 8 shows the estimates regarding the sum of the discounted cash-flow of every MMF and AE in the WMTC portfolio, considering a discount rate of 1\% per period.

To produce these estimates the WMTC PMO used average values drawn from a databank of similar projects. See [36, 37] for a discussion on alternative ways of making these estimates.

TABLE VIII. NPV OF THE AE AND MMF IN THE WMTC PORTFOLIO (US$ 1,000).

Module	Software Modules				
	1	2	3	4	5
CSU_1	-35	-34	-34	-34	-26
CSU_2	-80	-79	-78	-78	-59
GSP_3	-50	-49	-49	-48	-37
GSP_4	-18	-18	-17	-17	-13
GSP_5	1,141	1,115	1,089	1,064	-11
EP_3	713	699	685	670	-25
…	…	…	…	…	…
VoC_5	616	607	593	580	-15

E. Selecting the best implementation sequence

Because the number of MMFs and AEs is high, the number of possible implementation sequences is considerable. Even with the help of the precedence graphs introduced in Fig. 5, which has the effect of reducing the number of possible sequences of MMFs and AEs, the problem is still too computer consuming to be tackled by brute force. Hence, the WMTC appealed to the branch and bound algorithm for MMF sequencing developed by [17].

Although branch and bound algorithms may perform as poorly as their brute force counterparts in the worst case scenario, in many circumstances it can provide the optimum solution in a polynomial time [38]. Table 9 presents the results obtained by the algorithms according to the number of software development teams used by WMTC.

TABLE IX. PROJECT PERFORMANCE INDICATORS ACCORDING TO THE NUMBER OF DEVELOPED TEAMS.

# of Dev. Teams	NPV (US$ 1K)	Makespan (Periods)	PI Reaching Planned Value	Capital Invest. (US$ 1K)
1	7,313	19	70%	210
2	8,216	11	100%	258
3	8,444	8	100%	381
4	8,467	7	100%	385
5	8,559	7	100%	439
6	8,559	7	100%	439

It is important to note that if just one development team is used, not all performance indicators in the WMTC balanced score card will reach their intended target value, within the BSC window of opportunity.

On the other hand, there is a certain point from which increasing the number of development teams leads to neither a higher NPV nor a shorter total makespan. For example, using five or six development team leads to exactly the same NPV and total makespan.

Moreover, employing more development teams than necessary just adds to the complexity of managing the development of the portfolio without yielding any foreseeable benefits. According to [39], the more complex a project is, the more difficult it is to keep it on track.

As a result, the decision of the number of development teams to use resides on the amount of capital investment. The more capital investment is made available, the shorter the total makespan and the higher the total NPV.

As competition in the telecommunication market brought down the margins in almost all of the products and services, WMTC was able to allocated only US$ 300K for the portfolio. Hence, the logical choice according to the information displayed in Table 9 is to use only 2 development teams.

V. Conclusion

The Kaplan and Norton's balanced scorecard provides valuable insights regarding an organization strategy, assembling all their strategic goals and performance indicators. Therefore, undertaking the projects that will provide the highest NPV while achieving one or more of the performance indicators targeted values, within the BSC window of opportunity, is a must to any for-profit organization.

To accomplish such a goal, this paper introduces an optimization model based upon Kaplan and Norton's balanced scorecard and Denne and Clelang-Huang's minimum marketable features. The model allows organizations to maximize the return on investments of their portfolio while ensuring its strategic fit.

References

[1] J. V. D. Ende and W. Dolfsma, "Technology-push, demand-pull and the shaping of technological paradigms - patterns in the development of computing technology". Journal of Evolutionary Economics vol. 15 (2005) pp. 83–99.

[2] C. Chen, J. H. Lim and T. C. Stratopoulos, "Sustainable value creation: The role of it innovation persistence". In: Americas Conference on Information Systems, San Francisco, CA, USA, Association for Information Systems (2009).

[3] B. D. Reyck, Y. Grushka-Cockayne, M. Lockett, S. R. Calderini, M. Moura, A. Sloper,: "The impact of project portfolio management on information technology projects". International Journal of Project Management vol. 23 (2005) pp. 534–537.

[4] E. G. Swedin and D. L. Ferro, "Computers: The Life Story of a Technology". Westview Press (2004).

[5] M. Jeffery and I. Leliveld, "Best practices in it portfolio management". MIT Sloan Management Review vol. 45 (2004) pp. 41–49.

[6] G. S. Kearns and R. Sabherwal, "Strategic alignment between business and information technology: A knowledge-based view of behaviors, outcome, and consequences". Journal of Management Information Systems vol. 23 (2006) pp. 129 – 162.

[7] B. Maizlish and R. Handler, "Information Technology Portfolio Management Step-by-Step: Unlocking the Business Value of Technology". John Willey & Sons, Inc (2005).

[8] R. Kumar, H. Ajjan and Y. Niu, "Information technology portfolio management: Literature review, framework, and research issues". Information Resource Management Journal vol. 21 (2008) pp. 64–87.

[9] A. W. K. Tan and P. Theodorou, "Strategic Information Technology and Portfolio Management". IGI Global (2008).

[10] P. Kotler, G. Armstrong, "Principles of Marketing". Prentice Hall (2009)

[11] T. Little, "Value creation and capture: A model of the software development process". IEEE Software vol. 21 (2004) pp. 48–53.

[12] M. Denne, and J. Cleland-Huang, "Software by Numbers: Low-Risk, High-Return Development". Prentice Hall (2003).

[13] M. Denne, and J. Cleland-Huang "The incremental funding method: Data-driven software development". IEEE Software vol. 21 (2004) pp. 39–47.

[14] M. Denne, and J. Cleland-Huang "Financially informed requirements prioritization". In: 27th international conference on Software Engineering, St Louis, Missouri, USA, ACM New York, NY, USA (2005) pp. 710–711.

[15] L. J. Gitman, "Principles of Managerial Finance". Prentice Hall (2008).

[16] C. K. Chang, J. Cleland-Haung, S. Hua and A. Kuntzmann-Combelles, "Function-class decomposition: A hybrid software engineering method". IEEE Computer vol. 34 (2001) pp. 87–93.

[17] A. J. Alencar, E. A. Schmitz and E. P. de Abreu "Maximizing the business value of software projects: A branch & bound approach". ISAS-2 (2008) pp. 162–169.

[18] E. A. Schmitz and A. J. Alencar, "Defining the implementation order of software projects in uncertain environments". In: International Conference on Enterprise Information Systems, Barcelona, Spain (2008) pp. 23–29.

[19] I. Feller, "Performance measurement and the governance of american academic science". Minerva vol. 47 (2009) pp. 323–344 10.1007/s11024-009-9129-z.

[20] R. S. Kaplan and D. P. Norton, "Having trouble with your strategy? then map it". Harvard Business Review (2000) pp. 50–61.

[21] R. D. Behn, "Why measure performance? different purposes require different measures". Public Administration Review vol. 63 (2003) pp. 586–606

[22] H. Norreklit, "The balance on the balanced scorecard - a critical analysis of some of its assumptions". Management Accounting Research vol. 11 (2000) pp. 65–88.

[23] R. S. Kaplan and D. P. Norton, "The strategy-focused organization". Harvard Business School Press (2001).

[24] R. S. Kaplan and D. P. Norton, "The balanced scorecard - measures that drive performance". Harvard Business Review (1992) pp. 71 – 79.

[25] W. C. Rivenbark and E. J. Peterson, "A balanced approach to implementing the balanced scorecard". Popular Government vol. 74 (2008) pp. 31–37.

[26] H. Eilat, B. Golany, and A. Shtub, "R&D project evaluation: An integrated DEA and balanced scorecard approach". Omega vol. 36 (2006) pp. 895–912.

[27] R. S. Kaplan and D. P. Norton, "Organizational capital: Leadership, alignment, and teamwork". Balanced Scorecard Report (2004) pp. 1–7.

[28] C. Ittner, D. Larcker and T. Randall, "Performance implications of strategic performance measurement in financial services firms". Accounting, Organizations and Society vol. 28 (2003) pp. 715–741.

[29] A. Asosheh, S. Nalchigar and M. Jamporazmey, "Information technology project evaluation: An integrated data envelopment analysis and balanced scorecard approach". Expert Systems with Applications vol. 37 (2010) pp. 5931–5938.

[30] R. S. Kaplan and D. P. Norton, "The balanced scorecard: Translating strategy into action". Harvard Business School Press (1996).

[31] R. S. Kaplan and D. P. Norton, "The strategy map: guide to aligning intangible assets". Strategy Leadership vol. 32 (2004) pp. 10 – 17.

[32] P. R. Niven, "Balanced Scorecard Step-by-Step: Maximizing Performance and Maintaining Results". John Willey & Sons, Inc (2006).

[33] R. S. Kaplan and D. P. Norton, "Alignment: Using the balanced scorecard to create corporate synergies". Harvard Business School Press (2006).

[34] G. Taveira, A.J. Alencar and E. A. Schmitz, "A method for portfolio management and prioritization: An incremental funding method approach". In: International Conference on Enterprise Information Systems. vol. 3., Funchal, Madeira (2010) pp.23–33.

[35] R. S. Pressman, "Software Engineering: A Practitioner's Approach". 7th edn. MacGraw-Hill (2009).

[36] D. Vose, "Risk Analysis: A Quantitative Guide". John Willey & Sons, Inc (2008).

[37] D. W. Hubbard, "How to Measure Anything: Finding the Value of Intangibles in Business". John Willey & Sons, Inc (2010).

[38] M. J. Brusco and S. Stahl, "Branch-and-Bound Applications in Combinatorial Data Analysis. Springer (2005).

[39] K. B. Hass, "Managing Complex Projects: A New Model". Management Concepts (2008)

Model Checking Framework-based Applications with AspectJ Assistance

Zebin Chen
Microsoft
Redmond, WA, USA
zebinc@microsoft.com

Stephen Fickas
Department of Computer and Information Science
University of Oregon
Eugene, OR, USA
fickas@cs.uoregon.edu

Abstract— **We built Smart Home applications for the Cognitively Impaired population. We have chosen to work with an existing framework, OSGi, which allows us to develop specific applications more quickly. We use a combination of traditional testing and formal verification to insure these applications will cause no harm to the cognitively impaired users of our systems. This paper will focus on our results to date of using model checking to verify OSGi applications. We have created a formal model parallel to OSGi, which can be reused as a modeling framework to plug in and check OSGi applications. The trick has been to vary and combine different features of OSGi (i.e., abstract away the details that are not relevant to the particular property we wish to verify, but include the details when they are relevant), since a feature tends to involve code scattering in multiple classes. We have found that aspect-oriented programming, using AspectJ, is a potential solution. Using aspects, we have been able to prune large portions of OSGi in a controllable manner. One obstacle to this approach is that AspectJ will introduce native code so that an AspectJ program can't be checked by Java PathFinder (JPF). We report our efforts to date to resolve this problem and enable model checking AspectJ programs in general.**

Keywords- Model Checking; Java PathFinder; AspectJ; Framework; OSGi

I. INTRODUCTION

Our group has actively participated in the design and development of Smart Home applications [1][2]. We have chosen to work on the OSGi platform [3], which offers us standardized ways to manage the software lifecycle and more quickly develop applications across platforms. We are also advocates of formal methods, and have relied on model checking to uncover bugs in real systems [4][5]. We use a combination of traditional testing and formal verification to insure that the OSGi-based applications we build will cause no harm to the cognitively impaired users of our systems. As the OSGi framework and its applications are both developed in Java, we chose the model checker Java Pathfinder (JPF) [6][7], which directly checks Java bytecode.

In the process of checking OSGi applications, we soon found ourselves in a dilemma. On the one hand, OSGi applications are based on the OSGi framework; hence, it seems natural to build a formal model paralleling the OSGi framework and reuse the modeling framework to check OSGi applications. On the other hand, this approach is complicated by the very nature of model checking. A major obstacle to formal modeling is state space explosion, where the interleaving of processes leads to an exponential increase of system states that is beyond the capacity of a typical computer. This restriction requires a formal model being reduced to its bare essentials before the verification step. However, since OSGi applications may use different features of OSGi, it is nearly impossible to come up with a modeling framework that has just enough details for all OSGi applications. We will need some way to vary features of a modeling framework to check various applications.

The Object-Oriented nature of Java has offered some help for customizing the modeling framework. For example, one can override slots and hooks to specialize the modeling framework. However, there are some crosscutting concerns that may not naturally fit in this paradigm. For example, when we are interested only in the stale references problem [3][11], we don't need permission check in the framework and may remove all related fields and statements to save the state space. On the other hand, when we want to assure that no malicious application spoils the OSGi framework, we have to add back the missing fields and statements to enable permission check. Changing a feature is often a tedious task that involves modifications in multiple files, and it will be even worse when we have to come up with a specific combination of different features.

In our experience, such crosscutting concerns have prevented us from effective reuse of formal models and incurred much overhead in model maintenance. In view of these difficulties, we have found that aspect-oriented programming (AOP), using AspectJ [9][10] for Java, is a potential solution. Using AspectJ, we have been able to prune large portions of the OSGi code-base that are not relevant and vary features in a modular way. We report our findings in this paper.

The rest of the paper is organized as follows. We briefly explain the model checker Java Pathfinder in section II, and the application domain, OSGi applications, in section III; in section IV we use real examples of OSGi applications, to show that we can vary features of a formal model with AspectJ in a modular way; in section V, we show a necessary step, the construction of a specialized JPF, to enable verifying AspectJ programs in

general. We summarize the contributions and the future work in section VI.

II. JAVA PATHFINDER

Java PathFinder is an explicit model checker that directly checks Java bytecode. It has a specialized virtual machine, which takes on the role of a model checking engine like state cache, query, comparison, and restoration. Through a well-defined component architecture, JPF allows one to customize the search procedure. For example, it has a Model Java Interface (MJI) to intercept Java method calls, which may be used to resolve native methods (by executing native methods outside the JVM, but store interesting results in the JVM) and reduce state space (by excluding state information in the transition from the JVM). The extension schemes in JPF also allow expert users to experiment with novel search heuristics and state representations, which can be conveniently reused through the well-defined component interface.

There are several restrictions on JPF. First, it is not able to check platform-specific native methods like I/O methods, since the execution information is not available to the JVM. Second, due to state space explosion, it typically deals with Java programs with no more than 10k lines-of-code [7]. Therefore, before using JPF to check a real system, one would have to (a) resolve Java native methods with pure Java implementations or the MJI, and (b) apply various reduction techniques to construct a model with only the bare essentials for the actual system. In addition, one may also need to specify the interactions with the environments (e.g., user inputs) in a closed form. This is general to model checking since model checkers can only handle finite systems for a complete verification. In JPF this process can often be combined with the step to resolve native methods.

III. THE OSGI FRAMEWORK

For the purpose of explanation, we briefly introduce the application domain, i.e., the OSGi framework and OSGi applications [3]. The OSGi specification articulates a generic component structure for Java modularization. The unit for modularization is a bundle, which is a jar file typically composed of a manifest file, Java classes, native library and other resources. An OSGi framework has a base framework to fulfill the core functionality of the OSGi specification, e.g., manage the lifecycle of bundles. In particular, when starting a bundle, the base framework instantiates a bundle activator designated in the manifest file and invokes a callback function BundleActivator.start (i.e., a bundle activator is a Java class that implements org.osgi.framework.BundleActivator). When stopping a bundle, the framework invokes another callback function BundleActivator.stop. These callback functions are the main places to extend functionalities in a module. Each bundle runs in a separate JVM environment to avoid interferences between bundles; bundles communicate with each other through events delivered by the framework and services managed by the framework. A service is the unit to dynamically import/export functionality by a bundle. The OSGi framework maintains a mapping to associate service names to service references: a bundle (or more exactly, a bundle activator) can register a service by adding a mapping in the table, and use a service by first looking up a service in the mapping table.

A common pitfall in OSGi applications is the stale references problem. It happens when a service used by a consumer bundle has actually been unregistered by the producer bundle. This problem is acknowledged in the OSGi specification, and an auxiliary class, ServiceTracker, has been created to track valid services. We will uncover violations belonging to this category in our case studies. An interested reader can refer to [3] [11] for more information about the stale reference problem and some of its treatment at the application level.

There exist several reference implementations for the OSGi specification. Knopflerfish is a leading implementation and has been certified to be OSGi compliant [5]. It is composed of a base framework to fulfill the OSGi specification, mandatory bundles for basic functionality like input/output and optional bundles for extended functionality like logging. We base our case studies on the Knopflerfish framework. An interested reader can refer to [3][8] for a full explanation of OSGi and the Knopflerfish framework.

IV. ADVICE VERIFICAITON VIA ASPECTJ

We have built a formal model paralleling the OSGi specification, to ease modeling OSGi applications [10]. In this section, we point out some crosscutting concerns in the formal model, and show that we are able to modularize such concerns with AspectJ and conveniently specialize the base model with various features.

A. Add Modeling Code to Check Correctness

For the purpose of testing and model checking, we often add additional code to record system status and check the status in appropriate places. Such code is not part of the base model (not even part of the implementation code) and should be added only when we model check relevant properties.

Consider the example in Figure 1. It is the formal model one would reasonably derive with to check the stale references problem: the statements in *italic font* are the extra code solely added for verification purpose. It mandates that all services extend a BasicService (line 5), which has a boolean field valid to indicate whether a service is currently available. Upon the invocation of a service, the service will check whether it is currently available to other bundles (line 7). From the point of the framework, the field valid is set to true (line 14-19, line 24-29) when a service is registered successfully (line 13, line 23), and set to false (line 37-40) when a service is removed from the registration table in the framework (line 36). The code, solely for the purpose of model checking (all lines in italic font), spans multiple places of several Java files (e.g., ServiceRegistrationImpl.java, BundleContextImpl.java and all service implementation files), which is cumbersome to vary and interact with other model checking features.

```
1. public class BasicService {
2.   public volatile boolean valid = false;
3.   public ReentrantLock service_lock=new ReentrantLock();
4. }
5. class DictionaryImpl extends BasicService implements
DictionaryService {
6.   public boolean checkWord(String word) {
7.     assert(valid);
8.     ...// do the actual work
9.   }
10.}
11.public class BundleContextImpl {
12.  public ServiceRegistration registerService(String[]
clazzes, Object service, Dictionary properties) {
13.    ServiceRegistration sr=...// do the actual work
14.    if (sr!=null) {
15.      if (service instanceof BasicService) {
16.        BasicService ds = (BasicService)service;
17.        ds.valid=true;
18.      }
19.    }
20.    return sr;
21.  }
22.  public ServiceRegistration registerService(String clazz,
Object service, Dictionary properties) {
23.    ServiceRegistration sr=...// do the actual work
24.    if (sr!=null) {
25.      if (service instanceof BasicService) {
26.        BasicService ds = (BasicService)service;
27.        ds.valid=true;
28.      }
29.    }
30.    return sr;
31.  }
32.}
33.public class ServiceRegistrationImpl {
34.  Object service;
35.  private void unregister_removeService() {
36.    ...//remove the service
37.    if (service!=null && service instanceof BasicService) {
38.      BasicService ds = (BasicService)service;
39.      ds.valid=false;
40.    }
41.  }
42. }
```

Figure 1. Adding model checking code w/o AspectJ

The crosscutting concerns in Figure 1 can be conveniently modularized in a single aspect. As shown in Figure 2, the rule that all services shall additionally extend BasicService is specified by the declare parents phrase (line 1). The rule that the field BasicService.valid shall be set to true when a service is registered is realized by the pointcut that matches the completion of a successful registration (line 18-21) and the advice (line 22-25). The rule that the field valid shall be set to false when a service is unregistered is realized by the pointcut that matches the completion of the unregistration (line 11-13) and the advice (line 14-17). The rule that the service's validity shall be checked upon the invocation to its functions is realized by a pointcut that matches the access of a service's functions (line 2-4), and an advice that does the assertion (line 5-9). Whether to perform the verification of the stale references problem can be easily specified as inclusion or exclusion of the aspect while compilation.

```
1. declare parents: (DictionaryImpl) extends BasicService;
2. pointcut ServiceFunction(Object service) :
3.   execution(* checkWord(..)) && this(service)
4.   && this(DictionaryService);
5. before(Object service): ServiceFunction(service) {
6.   if (service instanceof BasicService) {
7.     BasicService bs = (BasicService) service;
8.     assert (bs != null && bs.valid);
9.   }
10.}
11.pointcut ServiceUnregister(ServiceRegistrationImpl sr):
12.  execution(* unregister_service(..)) &&
13.  this(ServiceRegistrationImpl) && this(sr);
14.after(ServiceRegistrationImpl sr) : ServiceUnregister(sr) {
15.  if (sr.service != null && sr.service instanceof BasicService)
16.    ((BasicService) sr.service).valid = false;
17.}
18.pointcut ServiceRegister(BundleContextImpl bc, Object
 service):
19.  this(bc) && this(BundleContextImpl) &&
20.  args(service) && if (service!=null) &&
21.  execution(ServiceRegistration registerService(.., Object,
 Dictionary));
22.after(BundleContextImpl bc, Object service)
 returning(ServiceRegistration sr):ServiceRegister(bc,
 service){
23.  if (sr != null && service instanceof BasicService)
24.    ((BasicService) service).valid = true;
25.}
```

Figure 2. Modularizing model checking code with AspectJ

B. *Vary Security Feature*

In addition to specifying correctness properties, we can also leverage AOP to conveniently choose a specific combination of features to check. As mentioned, we shall remove all non-mandatory fields and statements from the base model, and only add them back when they implement a specific feature relevant to the particular property to be verified. This methodology is critical to crack the state explosion problem and shall be enforced when possible.

However, we often find that adding back a feature involves adding fields and statements in multiple places of various files, and they are often different from their counterparts in the application code due to the result of abstraction. Enforcing such unfamiliar programming logic across multiple files is a daunting task when we have to check combinations of different features.

Consider the formal model in Figure 3. The statements in *italic font* enforce permission check for OSGi operations. A permission handler is created in Framework (line 5) and referred in Listeners and Services. The framework checks the permission when a bundle adds itself as a SynchronousBundleListener, to avoid a silly bundle blocking the whole framework (line 14-17). Similarly, the framework also checks the permission for registering a service to avoid a malicious bundle to preempt other valid services (line 31). These statements (in *italic font*) are not in the base model and are only needed when checking properties related to privileged operations. They scatter in more than ten places of three files (only a fragment of them are shown in Figure3), which is

tedious and error-prone for frequent addition and removal of features in model checking.

```
1. public class Framework {
2.   PermissionOps perm;
3.   public Framework(Object m) throws Exception {
4.     ...// Do other initialization
5.     perm=new SecurePermissionOps();
6.   }
7. }
8. public class Listeners implements {
9.   PermissionOps secure;
10.  Listeners(PermissionOps perm) {
11.    secure = perm;
12.  }
13.  void addBundleListener(Bundle bundle, BundleListener
listener) {
14.    if (listener instanceof SynchronousBundleListener) {
15.      secure.checkListenerAdminPerm(bundle);
16.      ...// do the actual work to add listeners
17.    } else ... // do the actual work to add listeners
18.  }
19.}
20.class Services {
21.  private PermissionOps secure;
22.  Services(PermissionOps perm) {
23.    secure = perm;
24.  }
25.  ServiceRegistration register(BundleImpl bundle, String[]
classes, Object service, Dictionary properties) {
26.    ...
27.    for (int i = 0; i < classes.length; i++) {
28.      String cls = classes[i];
29.      if (cls == null)
30.        throw new IllegalArgumentException("...");
31.      secure.checkRegisterServicePerm(cls);
32.      ... // do actual registering...
33.    }
34.  }
35.}
```

Figure 3. A formal model to check permission voilation in OSGi

```
1. static PermissionOps Framework.perm=new
SecurePermissionOps();
2. pointcut BL (Bundle bundle, BundleListener bl):
3.   execution(* addBundleListener(Bundle, BundleListener))
4.   && args(bundle) && args(bl);
5. before(Bundle bundle, BundleListener bl) : BL (bundle, bl)
{
6.   if (bl instanceof SynchronousBundleListener) {
7.     Framework.perm.checkListenerAdminPerm(bundle);
8.   }
9. }
10.pointcut ServiceRegistration(String classes[]):
11. execution(ServiceRegistration
register(..))&&args(classes);
12.before(String[] classes): ServiceRegistration(classes) {
13.  for (int i=0; i<classes.length; i++) {
14.    Framework.perm.checkRegisterServicePerm(classes[i]);
15.  }
16.}
```

Figure 4. The security model advised with AspectJ

We can modularize the inclusion of security in a single aspect. As shown in Figure 4, we use an intertype declaration to add back the permission handler (line 1). The invocation of permission check when adding a SynchronousBundleListener is matched by a pointcut (line 2-4), and advised before it is actually added (line 5-9). The invocation of permission check when registering a service is matched by a pointcut (line 10-11), and advised before it is actually registered (line 12-16). Compared with the needed modifications across multiple places of three files, adding permission check with AspectJ is modularized in a single aspect and much easier to change.

C. Change the Granularity of Atomicity

Despite the existence of partial order reduction, program slicing and other techniques to save the state space, we often need to manually enforce atomicity in various places of a model. On the one hand, manually enforcing atomicity may significantly reduce the search space, so that errors may be uncovered much faster and with less memory - in many cases it is also the only way to complete the verification run before running out of resource (e.g., memory and time). On the other hand, we may want to change the atomic code block when we are exploring potential solutions to a concurrency error.

```
1.public class ConsumerActivator implements
BundleActivator{
2.  public void start(BundleContext context) throws
Exception{
3.   ...
4.   DictionaryService service=...//look up service in
framework
5.   if (service!=null && service instanceof BasicService) {
6.      BasicService bs = (BasicService)service;
7.      bs.service_lock.lock();
8.      if (service.valid)
9.        service.checkWord(...);
10.     bs.service_lock.unlock();
11.   }
12. }
13.}
14.public ServiceRegistrationImpl {
15.  Object service;
16.  public void unregister() {
17.    ...
18.    if (service instanceof BasicService) {
19.      BasicService bs = (BasicService)service;
20.      bs.service_lock.lock();
21.      bundle.framework.listeners.serviceChanged(new
ServiceEvent(ServiceEvent.UNREGISTERING, reference));
22.      bs.service_lock.unlock();
23.    }
24.  }
25.}
```

Figure 5. Varying atomicity to explore counter examples

For these reasons, we would like to have a mechanism to conveniently vary the granularity of atomicity. However, the conventional approach to do this is fairly involved. For example, Figure 5 shows a potential solution to avoid the stale references by enforcing atomicity when invoking a service (line 5-12) and unregistering a service (line 18-23). We have to entangle the code (that changes atomicity) with other statements in the two files, and similar code will have to spread in each service declaration file if we want to enforce such atomicity for them.

```
1. pointcut atom_check(DictionaryService ds):
2.   target(ds) && execution(boolean checkWord(..))
3.   && if (ds instanceof BasicService)
4.   && !cflow(adviceexecution());
5. void around(DictionaryService ds) : atom_check(ds) {
6.   if (ds instanceof BasicService) {
7.     BasicService bs = (BasicService)ds;
8.     bs.service_lock.lock();
9.     proceed(ds);
10.    bs.service_lock.unlock();
11.  } else proceed(ds);
12.}
13.pointcut atom_unregister(ServiceRegistrationImpl sri):
14.  execution(void unregister(..)) && target(sri)
15.  && !cflow(adviceexecution());
16.void around(ServiceRegistrationImpl sri) :
atom_unregister(sri) {
17.  if (sri.service instanceof BasicService) {
18.    BasicService bs = (BasicService) (sri.service);
19.    bs.service_lock.lock();
20.    proceed(sri);
21.    bs.service_lock.unlock();
22.  } else proceed(sri);
23.}
```

Figure 6. Advising atomic blocks with AspectJ

The daunting task of enforcing atomicity across multiple files can be modularized by aspects. For example, we can intercept the execution of a non-atomic block with an around clause, and enforce atomic execution of the block with a shared, exclusive lock. As shown in Figure 6, we use an around pointcut to intercept the intended atomic block (line 1-4), and enforce atomicity for the action sequence (checking the service validity and using the service) in line 5-12. Similarly, the around pointcut in line 13-15 intercepts the intended atomic block, and the advice in line 16-23 enforces the desired atomicity. By this way, we can enforce arbitrary atomicity without scattering code here and there, which will ease feature management at the model level and improve the efficiency in exploring candidate solutions. In our experience, we do find a verified solution for the stale references problem in this approach.

V. MODEL CHECKING ASPECTJ PROGRAMS

In section IV, we have shown that formal models have crosscutting concerns that may be elegantly modularized via AspectJ assistance. However, such superiority of aspects is not practically useful if the Java bytecode woven by AspectJ can't be checked by JPF. In particular, we are concerned whether AspectJ will introduce extra native code to a pure Java model. This question is also significant for applications developed with AspectJ but now to be checked by JPF.

A simple AspectJ program like Figure 7 shows our fears are well founded. This program merely gets the signature of a joint point and involves no I/O from the surface. After we compile the program with the AspectJ compiler and check it with JPF, JPF reports the error as shown in Figure 8. Since all statements irrelevant to AspectJ are written in pure Java, the symptom shows that AspectJ introduces native code to a pure Java program. In this example, it occurs when the woven bytecode calls the native code in the AspectJ runtime library. A closer look at Figure 8 reveals that the NullPointerException is thrown when System.getProperty(String, String) is called (line 3). This is natural since it is a system-level function, which gets the environmental information outside the scope of the Java virtual machine.

```
1. public class Framework {
2.   public static void main(String[] args) {
3.     Framework fw = new Framework();
4.   }
5. }
6. aspect FrameworkConstructor {
7.   pointcut FrameworkConstructor(Framework f) :
8.     execution(Framework.new(..))
9.     && !cflow(adviceexecution()) && this(f);
10.  after(Framework f): FrameworkConstructor(f) {
11.    Signature sig = thisJoinPoint.getSignature();
12.  }
13.}
```

Figure 7. A simple AspectJ program that fails JPF

```
1. error #1: gov.nasa.jpf.jvm.NoUncaughtExceptionsProperty
2.java.lang.NullPointerException:calling
'getProperty(Ljava/lang
/String;Ljava/lang/String;)Ljava/lang/String;' on null object
3. at java.lang.System.getProperty(System.java:663)
4. at org.aspectj.runtime.internal.CFlowCounter. getSystem-
PropertyWithoutSecurityException(CFlowCounter.java:78)
5. at org.aspectj.runtime.internal.CFlowCounter.
selectFactoryForVMVersion(CFlowCounter.java:55)
6. at org.aspectj.runtime.internal.CFlowCounter.
<clinit>(CFlowCounter.java:29)
7. at edu.uoregon.osgi.mc.FrameworkConstructor.
ajc$preClinit(Framework.aj:1)
8. at edu.uoregon.osgi.mc.FrameworkConstructor.
<clinit>(Framework.aj:native)
9. at edu.uoregon.osgi.mc.Framework.<init>(Framework.aj:3)
10.at edu.uoregon.osgi.mc.Framework.main(Framework.aj:5)
```

Figure 8. Error message from JPF

We can use the MJI scheme in JPF to resolve native methods in the AspectJ runtime library, or customize the runtime library with a pure Java implementation. It is beyond the scope of this paper to fully describe the details to resolve native code in the AspectJ runtime library. As an example, we briefly explain the procedure to resolve native methods invoked by the program in Figure 7.

Defining a peer function in MJI is similar to calling a native method in Java Native Interface (JNI). A MJI method for System.getProperty(String, String) is shown in Figure 9. The function signature has name mangling so that it allows proper association in case of function overloading. The particular implementation in Figure 9 allows reading system properties outside the closed JVM.

After this peer method is installed, JPF will intercept the call to System.getProperty(String, String) by matching the function name. Thereafter, the host JVM executes the peer method to get the environment information (line 8), and stores

the result in a special area. JPF fetches the result from this special area as if it is read from the environment.

```
1. public static int getProperty__Ljava_lang_String_2Ljava
_lang_String_2__Ljava_lang_String_2 (MJIEnv env, int
clsObjRef, int keyRef, int defRef) {
2.   int r = MJIEnv.NULL;
3.   if (keyRef != MJIEnv.NULL) {
4.     String k = env.getStringObject(keyRef);
5.     String defaultString = env.getStringObject(defRef);
6.     if (k==null)
7.       return MJIEnv.NULL;
8.     String v = System.getProperty(k);
9.     if (v != null)
10.      r = env.newString(v);
11.    else if (defaultString!=null) {
12.      r = env.newString(defaultString);
13.  }
14.  return r;
15.}
```

Figure 9. Peer method for System.getProperty(String, String)

As a result to date, we have created a MJI abstraction library and customized the AspectJ runtime library, to help resolve native methods introduced by AspectJ. We also created a test suite that includes all sample programs (excluding those that have native code in the Java program) from [9][10]. In our testing, all native code introduced by the AspectJ keywords has been successfully resolved, and we are currently investigating the whole runtime library (~180K) of AspectJ.

VI. SUMMARY

When we use model checking to verify real-world Smart Home applications, we have found a strong need to vary features of a formal model and customize the verification procedure. This is required as part of the efforts to conquer the state space explosion problem, by removing irrelevant details. However, varying features for a formal model tends to result in modifications scattering in multiple Java files, and it is even worse when we vary the combination of different features to study feature interactions. With examples, we have shown that these crosscutting concerns can be modularized with aspects, and can thus be easily enabled and disabled. We also point out that AspectJ may introduce native code to a pure Java program, and show an example that we can leverage the MJI scheme to solve it. We are currently building a MJI abstraction library, which will benefit not only our paradigm to advise model construction with AspectJ, but also AspectJ applications in general that have been developed without the awareness of model checking. In general, if we deem a model checker as an explorer for counter examples, the overall methodology enables easy exploration of solution space with very different features and goals.

We are also investigating the performance overhead introduced by AspectJ and looking for ways to minimize such impact. This occurs because additional state variables are introduced, e.g., aspect initialization. In our current experience, the state space increase of a complex model is relatively small and doesn't appear to be a show stopper. This problem can be further mitigated with search heuristics, e.g., one can enforce atomicity for aspect initialization. This brings up an interesting distinction between model checking AspectJ programs and using AOP techniques to ease model checking Java programs.

REFERENCES

[1] Z. Chen, and S. Fickas, "The plain old television in a smart apartment", in Proceeding of the First International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateComm 2005), San Jose, CA, Dec 2005.

[2] S. Fickas, C. Pataky, and Z. Chen, "DuckCall: tracking the first hundred yards problem", in Proceeding of the Eighth SIGACCESS, Portland, OR, Oct 2006.

[3] The OSGi Alliance, OSGi Service Platform Core Specification (Release 4), Aug 2005.

[4] M. Feather, S. Fickas, and A. Razermera, "Model-checking for validation of a fault protection system", in Proceedings of IEEE International Symposium on High Assurance Systems Engineering, Boca Raton, 2001, 32-41.

[5] Z. Chen and S. Fickas, "Do No Harm: Model Checking eHome Applications", in First International Workshop on Software Engineering for Pervasive Computing Applications, Systems, and Environments (SEPCASE '07) at ICSE 2007.

[6] W. Visser, K. Havelund, G. Brat, S. Park, and F. Lerda, "Model Checking Programs", In Automated Software Engineering Journal, vol. 10, no. 2 (Apr.2003), 203-232.

[7] P. Mehlitz, W. Visser, and J. Penix, "The JPF Runtime Verification System", manual accompanied in JPF distribution, http://sourceforge.net/projects/javapathfinder

[8] Knopflerfish, http://www.knopflerfish.org/

[9] A. Colyer, A. Clement, G. Harley, M. Webster, eclipse AspectJ: Aspect-Oriented Programming with AspectJ and the Eclipse AspectJ Development Tools, Addison-Wesley, 2005.

[10] AspectJ web site, http://www.eclipse.org/aspectj

[11] K. Gama and D. Donsez, "A Practical Approach for Finding Stale References in a Dynamic Service Platform", in Proceedings of the 11th International Symposium on Component Based Software Engineering (CBSE 2008), Oct 2008, Karlsruhe, Germany.

User-defined Scenarios in Ubiquitous Environments: Creation, Execution Control and Sharing

Matthieu Faure, Luc Fabresse
Ecole des Mines de Douai, Douai, France
{Matthieu.Faure, Luc.Fabresse}@mines-douai.fr

Marianne Huchard
LIRMM - UMR 5506, CNRS and Univ. Montpellier 2, Montpellier, France
huchard@lirmm.fr

Christelle Urtado and Sylvain Vauttier
LGI2P / Ecole des Mines d'Alès, Nîmes, France
{Christelle.Urtado, Sylvain.Vauttier}@mines-ales.fr

***Abstract*—Ubiquitous computing provides a dynamic access to different functionalities of networkable electronic devices. Whereas basic services have limited use, predefined complex services cannot encompass every end-user's needs nor be adapted to a set of services that are dynamically discovered in an open environment. Alternatively, users need to be provided with means to express their requirements, choosing precisely which services to compose into a scenario of their own. In service-oriented computing, some systems propose mechanisms to develop tailored components that provide composite services; however they are not adapted to end-users, have limited composition capabilities and/or do not consider several characteristics of ubiquitous environments (such as multiple users and devices).**

This paper presents a novel user-centric system called SaS for mobile personal devices. SaS provides end-users with an easy access to services and a simple GUI to combine them into complex scenarios. A new architectural description language is used to specifically support scenario creation by service composition. Scenario may be shared among users and devices. SaS offers scenario execution control for example to start and stop it but also to query the current state of a scenario. In addition, SaS proposes some mechanisms to maintain scenario availability in case of service/device unavailability. SaS is currently implemented in a proof-of-concept prototype on top of OSGi.

Keywords*—*Ubiquitous computing, service-oriented computing, user-centric system, service composition, scenario creation.

I. INTRODUCTION

With the rise of ubiquitous computing [1], [2], we are surrounded by electronic devices (such as smart phones or TVs) that propose a huge amount of services through public or private networks. According to the OASIS organization, *"a service is a mechanism to enable access to one or more capabilities"* [3]. In service-oriented computing (SOC) [4], and specially in home automation [5], [6], [7], efforts have been made to facilitate the use of these electronic devices through their services. As shown in Figure 1, the more complex user requirements can only be satisfied by compositions of multiple services provided by multiple devices. Different service composition means have been studied and proposed [8], [9], [10], [11]. However, they are not designed for end-users without technical knowledge.

Enabling end-users to describe their own *scenarios* is a first improvement and a step towards ambient intelligence [12]. In addition, users should easily manage the created scenarios and have access to them from several control devices (PDA, mobile phone or laptop). Moreover, ubiquitous environments imply that several users might be eager to share scenarios. Scenarios should therefore be exported in the environment to be shared and reused.

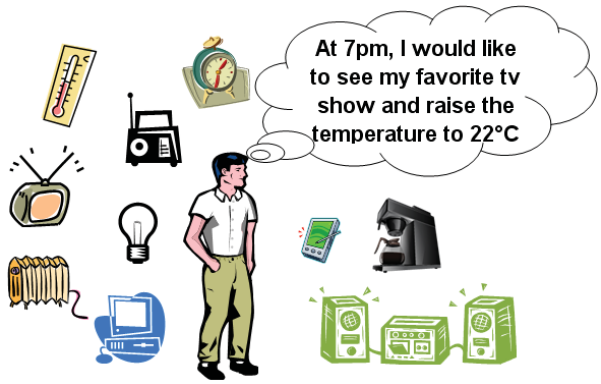


Figure 1. User's main issue

In this paper, we propose the SaS (Scenarios as Services) system specifically designed for end-users to describe, use and share *scenarios* in a high level manner. SaS comprises a new architecture description language (ADL) [13], [14], [15] dedicated to user scenario creation. SaS also integrates a graphical user interface (GUI) based on this ADL. This GUI presents a classified and filtered view of the services available from a set of widely spread devices and provides tools to easily *compose* selected services. SaS integrates scenarios as *regular services*. This enables easy scenario execution control, scenario sharing among users and hierarchical composition of scenarios. The SaS system has been prototyped. Its current implementation is on top of OSGi [16].

The remainder of this paper is structured as follows. Section II introduces the context of this work, presents the requirements for open and distributed environments and discusses the state of the art. Section III presents the first part of our proposition: scenario creation. Section IV is dedicated to scenario execution control and sharing. Section V presents the architecture of the prototype implementation. Finally, section VI evaluates our proposition, concludes and draws some perspectives to this work.

II. User-Centric Systems in Ubiquitous Environments

This section first describes the terminology of ubiquitous environments and especially that of user-centric systems. It then presents the requirements that are mandatory for that kind of systems. It finally compares some of the main state of the art approaches regarding these requirements.

A. Terminology

Ubiquitous systems involve multiple *users* and multiple *devices* that each provide a set of *services*. A device is an electronic object (such as a clock). Devices publish services (such as `Time`). Each service provides one or more *operations* (such as `getTime` or `setTime`). They are called "capabilities" in the SOA norm [3] by the OASIS consortium. End-users use these operations as an access to functionalities of devices.

Devices can interoperate but the overall goal that the system has to achieve always comes from users. Users can be simple consumers or technical experts that command devices, their needs can always be considered as *scenarios* which are combinations of operations. However, we choose to name this combination a *service composition* because services are not stand-alone elements and to stick to the terminology used in SOC.

A SaS system is deployed on a mobile personal device. The system and the device on which it is deployed together define a SaS *platform*. A SaS platform participate in one or more networks which constitute the platform's environment. The global environment is the union of all the environment of its constituting SaS platforms.

B. Requirements for user-centric systems

User needs always constitute a scenario. User-centric system must therefore enable *scenario creation*. As seen in Figure 1, user scenarios are not always simple service aggregations but can imply conditions, control statements and logical operators. Users should thus be able to *compose services* according to their needs. Most of the users are not technical experts. Scenario creation should therefore be *user-friendly* and adapted to devices. Ubiquitous environments imply multiple users and devices. So, created scenarios should be available into the environment and *shared* among users. Users must be able to easily start and stop created scenarios and check scenario status and execution advancement. Thus, the system should control the *scenario life-cycle*. Moreover, already created scenarios should be easily modified and *recomposed* into other ones. In addition, devices that provide services and/or scenarios can disappear. The system must therefore *maintain* scenario execution and availability in case of device disappearance.

C. State of the art

With the requirements established previously, we can analyze some of the main systems that provide a solution for ubiquitous environments and enable end-users to create scenarios.

- **SLCA** [17] provides developers with the capability to compose web services. A composite service contains proxy components attached to involved web services. SLCA enables hierarchical service composition. In addition, it is an event-based system which adapts to environment changes. In case of service unavailability the composite service replaces it if an appropriate service is found. If not, the composite service removes the proxy component attached to this service.
- **MASML** [6] is a multi-agent system for home automation. Scenarios are defined with an XML syntax and consist of a sequence of service operation invocations. MASML XML documents can embed ECMA scripts [18] to add logic elements. A mobile agent is in charge of scenario execution. It is moving to each appropriate device with the scenario description file to execute it. This enables scenario advancement tracking but not parallel execution.
- **SODAPOP** [19] proposes an innovative approach based on the same observation than us: user needs are scenarios. What is important is the goal to achieve. The main hypothesis is that each service contains informations about its initial conditions and its effects. SODAPOP automatically classifies new services with these informations. Then, it can combine some of them to reach the user's goal.
- **SASHAA** [20], [21] is one of our previous work, focused on ubiquitous systems for home automation. It enables end-users to create scenarios with Event - Conditions - Action rules through an appropriate GUI. It is an adaptive system which creates a new component for each new device.

Table I compares these systems with respect to the requirements that we identified in II-B. Symbol $\surd$ means that the requirements is fulfilled, - signifies that it is partially accomplished and X represents an absence of solution.

Systems	Scenario Creation	Advanced Service Composition	User friendliness	Scenario sharing	Scenario Lifecycle	Hierarchical Composition	Scenario Maintenance
SASHAA	$\surd$	-	$\surd$	X	-	X	$\surd$
SLCA	$\surd$	$\surd$	X	X	X	$\surd$	-
MASML	$\surd$	$\surd$	X	X	-	X	$\surd$
SODAPOP	$\surd$	X	-	X	X	X	X

Table I
SYSTEM COMPARISON WITH OUR REQUIREMENTS

Except for SAASHA, we can notice in Figure I that all these works propose programming tools for developers. They are not directed to end-users. In addition, scenario sharing is never took into consideration. For scenario life-cycle control, SASHAA only enables to start and stop scenarios whereas MASML just allows users to check scenario advancement. Because of this, we decided to propose a new system which best meets all the expectations of user-centric systems for ubiquitous environments.

III. The SaS System: Scenario Creation

A. Overview of SaS

The purpose of SaS which stands for *Scenarios As Services* is threefold:

1) help end-users create scenarios by service composition,
2) monitor scenario execution on a given platform,
3) export scenarios into the environment for future use or sharing.

To do so, several steps are necessary that define a user-centric cycle, as illustrated by Figure 2:

0. The system (placed into a user device) discovers the services available in its neighboring environment.
1. SaS classifies service operations depending on their providers (devices) and services. It then displays them.
2. Users can compose several available services to create a scenario. This is possible through a dynamically adaptive graphical user interface based on our ADL.
3. The created scenario is translated into a descriptor file. It therefore becomes easily transmissible and can be shared with other platforms and users.
4. Next, SaS analyzes the scenario descriptor. It extracts information about the different services involved and how they are composed.
5. The system creates a composite with the involved services and a generated *manager*. This manager handles the services according to the previously made user choices.

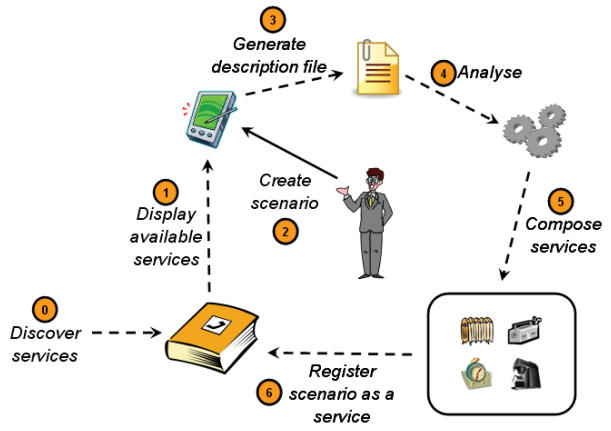


Figure 2. Overview of the proposed SaS scenario creation and reuse cycle

6. Finally, the manager, which is in the composite, registers the scenario as a new service into the environment. It becomes accessible from other devices and shared among end-users. Moreover, it can be composed into a new scenario.

B. Scenario creation

This section describes scenario creation, which is the main part of our system. It consists in three steps: service selection, scenario construction by service composition and scenario export. As described in Section III-A, the first functionality of the SaS system is service discovery. Some protocols already exist that do so (*e.g.* UPnP [22], SLP [23], Jini [24]) along with different extra functionalities. To be as interoperable as possible, our SaS system does not prescribe the use of a particular discovery protocol. Once the available services are discovered, they can be listed and ordered by SaS to enable service selection.

1) Service Selection: Every service proposes one or more operations (for example, the *light* service might offer two operations: `getValue` and `setValue`). To define a scenario, users always select operations, but the name of the service and the identity of the provider device do not always matter. For example, to print a document a person generally chooses his favorite printer, accesses the specified service and selects the appropriate operation. Alternatively, if he needs to know what time it is, he directly selects the `getTime` operation, no matter which clock or service provides it.

Service selection in SaS sticks to this requirement. SaS proposes three filtered views to select available operations: by device type (*e.g.* the list of available printers), by service name (*e.g.* the *printService* service) or directly by operation name (*e.g.* the `print` operation). If users select a device, services provided by this device are then proposed to choose from. If users select a service, operations that compose this service are then proposed to choose from. Moreover, distinct devices can propose services with the same name, sometimes with additional operations. SaS groups these services together and displays all the available operations collectively.

2) Service Composition: SaS enables service composition thanks to an ADL and its GUI. Depending on user choices, created scenarios can be then exported into the environment.

a) Presentation of the ADL: In order to help end-users create scenarios that correspond to their needs, we propose a new ADL. It is simple and tailored to scenario creation. Compared to other programming languages for service composition (like BPEL [25]) which are imperative and designed for executable process, our ADL is a high level language, declarative and destined to end-users. With this ADL, one can declare both services and scenarios.

- Service declaration
  We define a service by a device (its provider), a name and an operation list. This list cannot be empty. Operations have a return type (which can be *void*) and can have typed parameters. We represent only the main elements of the grammar in Listing 1.

```
<service> ::= service <device> <service_name> <op_list>

<op_list> ::= ( <operation> ; )*

<operation> ::= operation <operation_name> (
   [<parameter_list>] ) : <return_type>

<parameter_list> ::= <parameter_type> (,<parameter_type>)*

<return_type>::= <type>
<parameter_type> ::= <type>
```

Listing 1. Service declaration with the Backus–Naur Form (BNF)

- Scenario declaration
  By definition, a scenario has a name and an action list. An action can be:
  - **an operation invocation**: a service operation is invoked, with its parameter values. Users can directly enter parameter values or invoke another service operation to create the desired value (operation composition). SaS checks if parameter types conform to the service definition.
  - **an alternative (*if - else*)**: conditions compare the result of two service operations or the result of a service operation and a value chosen by the user.
  - **a repetition loop**: enables `while` loops iterations while a condition remains satisfied. Alternatively, it is possible to precise how many `times` a series of actions should be invoked.

Listing 2 describes the main elements of a scenario declaration using the BNF notation.

```
<scenario> ::= scenario <scenario_name> <action_list>

<action_list> ::= { <action> + }
<action> ::= <op_invocation> ; | <alternative> | <repeat>

<op_invocation> ::= [<device>] <service_name>.
  <operation_name>([<parameter_list>])
<parameter_list> ::= (<op_invocation>|<parameter_value>)
                  (, (<op_invocation>|<parameter_value>))*

<alternative> ::= if <cplx_cond><action_list> [<else_clause>]
<else_clause> ::= (else <action_list> )*
<cplx_cond> ::= (<condition> (<log_operator><condition>)*)
<condition> ::= <op_invocation> <comp_operator>
                (<op_invocation> | <compare_to_value>)
<repeat> ::= (while <cplx_cond> | <repeat_value> times)
             <action_list>

<log_operator> ::= and|or|not
<comp_operator> ::= < | <= | > | >= | ==
```

Listing 2. Grammar of the scenario declaration using the BNF notation

b) The graphical user interface: This ADL syntax is simple and declarative as we can see in the example on the right of Figure 3. Nevertheless, SaS proposes a more user-friendly option to create scenarios through a graphical representation of the ADL. Users therefore do not manipulate the ADL anymore but compose service operations with basic instructions (based on the *operators* of our ADL): `if`, `else`, `while`, `times`, `and`, `or`, `not`, $<$, $>$, $\leq$, $\geq$, $==$. For parameters entries, users can select an operation result or choose fixed values and apply an arithmetic operation (such as $+$, $-$, $*$, $/$).

Once the scenario is defined, users can choose to export it into the environment. They also have to specify if the scenario can be redeployed into another SaS platform. Thanks to a transformation process, the scenario is then automatically transcribed into our ADL. Figure 3 (right) shows the scenario transcription with a simplified version of the GUI (left).

3) Composite service creation: After the scenario is created, SaS analyzes its description file to create a composite that manages

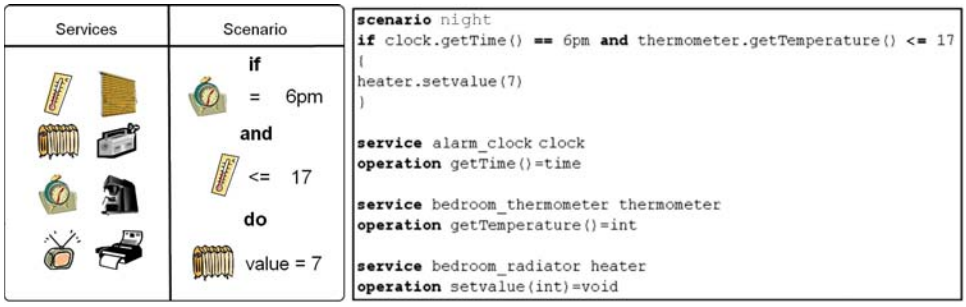


Figure 3. Scenario Transcription: from our ADL

its deployment. This composite includes references to the services chosen in the scenario and a *Scenario Manager*. The manager has two roles: manage the different services and export the scenario as a new service in the system according to users preferences.

Depending on user choices, services instantiated inside the composite are specific to a device or come from any of its available providers[1]. In this last case, if the service provider disappears, SaS dynamically recomposes the composite that implements the scenario to integrate another implementation of the same service (if available). Figure 4 illustrates composite services with an example. The scenario is simple and placed in a home automation environment: at 6pm, close the main door and set the thermostat at 7. There are three services: only one is defined from a specific device (the main door). Others are instantiated from any devices that provide these services.

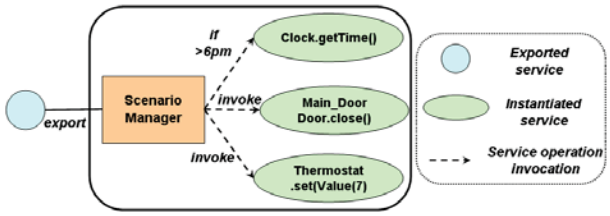


Figure 4. A Composite Service

IV. Scenario execution control and sharing

Once scenarios are created, they can be shared among users. In addition, scenarios are easily manageable and should stay available into the environment in case of service or device unavailability.

A. Scenario sharing

As seen in subsection III-B3 the Scenario Manager registers the scenario as a new service. The scenario can then be used as a service and, as such, composed into a new coarser grained scenario (scenario hierarchical composition). This service has four operations: `start`, `stop`, `getScenarioState` and `getDescriptor`. It does not describe the functioning of the scenario: it hides services and their interactions inside the composite. This guarantees encapsulation. Reflexion is nonetheless provided (composites are not black boxes but gray boxes) thanks to a service operation: `getDescriptor`. This operation provides access to the scenario descriptor file. Users can directly read this file or obtain a visual transcription of the scenario on his GUI.

Figure 5 illustrates scenario sharing. The user of the SaS platform named A creates and exports a scenario. This scenario is registered as a new service. It is then discovered by platforms B and C. The user of platform B recomposes this scenario into a new one, whereas the user of platform C just gets an overview of the scenario on its GUI.

[1]An optimized selection scheme is a perspective.

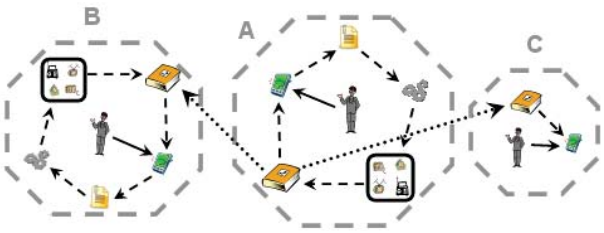


Figure 5. Overview of SaS scenario sharing

B. Scenario execution control

We define scenarios as service compositions. We can see a scenario as an active entity, which evolves, changes from one state to another. Moreover, the execution contains several steps which can fail or succeed. For example, users that discover a scenario might want to know if this scenario is currently in execution. If so, it is important to check which steps have been executed, which succeeded and what are the next steps. This is why, SaS manages scenarios' life-cycles and enables to check scenario execution status.

1) Scenario life-cycle: Scenarios are dynamic. They can be executed, stopped, have a missing service... The state diagram of Figure 6 illustrates the different states of scenario life-cycle which are:

- **Installed,** the scenario has been deployed and registered (and so discovered) as a new service. SaS automatically checks if the different involved services are present.
- **Ready to launch,** all involved services are available. If a service disappears, the scenario goes to the previous state.
- **Running,** the scenario has been launched and is currently executed. The scenario could finish and come back to the previous state or be interrupted.
- **Stopped,** the scenario has been stopped by a user or a service inside the scenario disappeared. The scenario is paused waiting to be restarted or to be executable again by the appearance of an appropriate service.

Figure 6. State diagram of scenario life-cycle

2) Scenario running state advancement: Users must know which scenario operations are running and which have already been executed. This is why, SaS registers scenario execution progress. To do so, SaS considers the *Running* state of the scenario life-cycle as

a succession of stages: the operation invocations. These stages are evaluated depending on their types:

- **Functions**, if an operation is invoked to obtain a result, SaS logs this operation as done when we obtain the result. If an error occurs, SaS continues to execute the scenario if possible (*i.e.* if the operation result is not needed) and displays a warning to the user.
- **Procedures**, if the operation does not return a result, SaS logs it as *executed* when the operation is invoked.

In addition, SaS logs the execution times of the different scenario steps. Users can see when the scenario began and how long every operation took. So, SaS enables users to check the current scenario position and control its correct advancement. Users can get these informations thanks to the `getScenarioState` operation.

C. Scenario availability

With scenario export as new services, users can have access to the same scenario on several devices, however, they might want access to it even if the original provider is off. This is why, scenario access should be maintained if the original provider disappears.

To do so, SaS enables scenario redeployment on other platforms. This is possible because SaS differentiates scenarios from available services. When a scenario appears, every SaS platform downloads the scenario description. Thus, users can directly have an overview of the scenario definition and platforms can redeploy the scenario if the original scenario provider disappears.

V. System Design and Implementation

This section describes the design and implementation of the SaS prototype. It is an ongoing work implemented in Java over OSGi [16], [26] with iPOJO [27]. OSGi is a popular SOC framework that is widely adopted by industry for developers to create *bundles* (Java components). iPOJO is based on OSGi and follows the Service-Oriented Component Model [28]. The main idea is that a component should only contain business logic as in EJB 3.0 [29] (*EJB entities*); SOC mechanisms should seamlessly be handled by the component container as container-managed cross-cutting services.

A. Model

As shown in Figure 7, four components compose SaS. Each of them is packaged as an OSGi bundle because it is safer and easier to update.

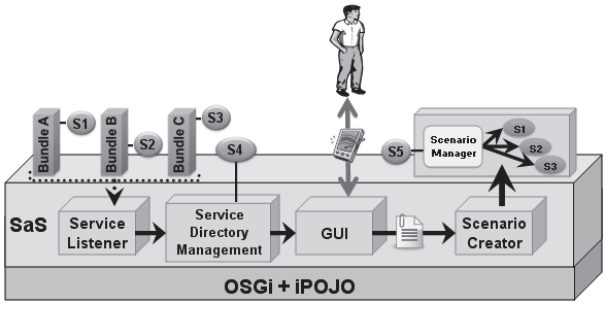


Figure 7. The SaS prototype implemented over OSGi and iPOJO

- **Service Listener**. This bundle obtains, orders and dynamically updates the list of available services from the OSGi context.
- **Service Directory Management (SDM)**. It is the intermediate between the *Service Listener* and the GUI responsible of managing (adding, removing) services and scenarios in the *Service Directory*.
- **GUI**. The GUI is platform and operating system specific (PDA, mobile phone, Android, iOS,...). It provides a graphical representation of our ADL which provides users with the capability to see the available services and compose them.
- **Scenario Creator**. It creates scenario bundles which are composed of the selected services and a scenario manager. This latter manages services inside the composite and exports the scenario as a service.

B. Insights into the SaS prototype

This subsection presents the implementation of the main functionalities of the SaS system.

The latest version (4.2) of OSGI now supports distribution (RFC 119 [26]). So, for service discovery, SaS uses the API of distributed OSGi which can be implemented by many discovery protocols. Then, the *Service Listener* retrieves the list of available services from the directory provided by the OSGi framework and sends it to the *SDM*. This latter orders and classifies the service list. The GUI[2] displays available devices, services and operations. It filters the displayed services (*resp.* operations) if a specific device (*resp.* service) is selected. A user creates a scenario through the GUI which stores it in an XML-based description file. This description becomes easily transmissible and promptly interpretable and analyzable since XML is a standard as an exchange format. Using this description, the *Scenario Creator* of SaS automatically (i) generates a scenario manager which (ii) exports and manages the scenario. Finally, the scenario (iii) is redeployed on other SaS platforms according to user preferences. The remainder of this section describes more deeply this three important steps.

1) Scenario Manager generation: Starting from the XML-based description of a scenario, SaS generates a Java class that represents the manager of this scenario (*Scenario Manager*). To do so, the first step consists to parse the XML description file using a *SAX* parser [30]. SAX translates XML elements into a sequence of Java instructions. Then, SaS generates a *Scenario Manager* class with the Javassist [31] library that enables dynamic byte code edition such as creating classes or modifying existing classes. The *Scenario Manager* is generated as a class that implements the *ScenarioManagerInterface* interface. This interface declares four public methods including a `start` method which is automatically filled in the *Scenario Manager* class with the Java instructions resulting from the SAX parsing.

2) Scenario export and execution control: SaS uses the iPOJO API [32] to dynamically create an OSGi composite bundle that packs together the generated *Scenario Manager* and the involved services. This composite bundle is then installed and started into the OSGi platform. Finally, the *Scenario Manager* registers a new service inside the OSGi directory, specifying its capability to execute four public methods (`start, stop, getScenarioState` and `getDescriptor`).

For scenario execution control, the *Scenario Manager* creates a log file every time the `start` service operation is invoked with the invoker platform *id* and the current time. Then, the *Scenario Manager* logs in this file every service invocation success (or failure) with time. With this log file, SaS knows at every moment if the scenario is currently in execution, when it began, who launched it and which steps are already executed. These informations are available through the service operation `getScenarioState`.

3) Scenario automatic deployment: As seen in V-B1, all scenarios implement the same Java interface (*ScenarioManagerInterface*). So, SaS can easily recognize them. Thus, when a scenario is discovered as a new service, the *Service Directory Management* automatically gets the scenario description file thanks to the `getDescriptor` operation provided by the service. The scenario is not deployed again but SaS keeps the XML description file. *SDM* sends the scenario description to the GUI. If the original provider disappears, another SaS platform may redeploy the scenario if its directory contains all the involved services.

[2]which is still under development.

VI. Evaluation and Conclusion

With the SaS system, we propose a newly user-centric system that meets the expectations of ubiquitous environments. First, we provide *scenario creation* by service composition. Users can create *complex scenarios* that correspond to their needs thanks to an appropriate ADL. This ADL is simple, user-oriented and proposes an alternative graphical view to be *accessible for everyone*. SaS exports scenarios as new services into its environment, thus, users can easily *share* their scenarios. Moreover, SaS manages *scenario life-cycle*: it enables users to start and stop scenarios, check scenarios status and scenario execution advancement. Moreover, users can get an overview of a scenario specification (scenario introspection capability) thanks to a descriptor file and *reuse a scenario* as a service being part of a new encapsulating scenario composition (scenario hierarchical composition). SaS also tries to *maintain scenario availability*. Locally, if a service involved into a scenario disappears, SaS tries to replace it into the composite. Globally, when a SaS platform disappears, scenarios exported by this platform are redeployed on other ones to remain available. In conclusion, the SaS system presented in this paper satisfies all the requirements defined is section II-B. A prototype of SaS in Java over OSGi and iPOJO is currently under development.

We have three major perspectives. First, we want to evaluate SaS to show its simplicity for non technical end-users by comparing it with existing tools such as Yahoo Pipes [33], Automator [34] and Scratch [35]. Such tools provide non-technical end-users of the capability to graphically develop small applications by composing elements. Then, we plan to define some recovery strategies to anticipate service loss such as *caching* and *hoarding*. Finally, we want to improve scenario distribution and propagate scenarios into the network.

Acknowledgements

This work is partially supported by a grant from the CARNOT M.I.N.E.S Institute (*http://www.carnot-mines.eu/*).

References

[1] M. Weiser, "The computer for the 21st century," *Scientific American*, pp. 78–89, 1995.

[2] H. Schulzrinne, X. Wu, S. Sidiroglou, and S, "Ubiquitous computing in home networks," *IEEE Communications*, pp. 128–135, nov 2003.

[3] OASIS, "Reference Model for Service Oriented Architecture 1.0," pp. 12 – 13, oct 2006. [Online]. Available: http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.html

[4] M. P. Papazoglou, "Service-Oriented Computing : Concepts , Characteristics and Directions," in *Proc. of the 4th International Conference on Web Information Systems Engineering*. IEEE Computer Society, 2003, pp. 3–12.

[5] A. Bottaro, A. Gérodolle, and P. Lalanda, "Pervasive service composition in the home network," in *Proc. of the 21st International Conference on Advanced Networking and Applications*, 2007, pp. 596–603.

[6] C.-L. Wu, C.-F. Liao, and L.-C. Fu, "Service-Oriented Smart-Home Architecture Based on OSGi and Mobile-Agent Technology," *IEEE Transactions on Systems, Man and Cybernetics, Part C (Applications and Reviews)*, vol. 37, no. 2, pp. 193–205, mar 2007.

[7] D. Valtchev and I. Frankov, "Service gateway architecture for a smart home," *Communications Magazine, IEEE*, pp. 126–132, 2002.

[8] M. Bakhouya and J. Gaber, *Agent Systems in Electronic Business*. IGI Publishing, 2007, ch. Service Composition Approaches for Ubiquitous and Pervasive Computing Environments: A Survey, pp. 323–350.

[9] J. Bronsted, K. M. Hansen, and M. Ingstrup, "Service composition issues in pervasive computing," *IEEE Pervasive Computing*, vol. 9, pp. 62–70, 2010.

[10] A. Urbieta, G. Barrutieta, J. Parra, and A. Uribarren, "A survey of dynamic service composition approaches for ambient systems," in *Proceedings of the 2008 Ambi-Sys workshop on Software Organisation and MonIToring of Ambient Systems*, ser. SOMITAS '08, 2008, pp. 1–8.

[11] N. Ibrahim and F. Le Mouël, "A Survey on Service Composition Middleware in Pervasive Environments," *International Journal of Computer Science Issues (IJCSI)*, vol. 1, pp. 1–12, 2009. [Online]. Available: http://hal.inria.fr/inria-00414117/en/

[12] E. Aarts and B. de Ruyter, "New research perspectives on Ambient Intelligence," *Journal of Ambient Intelligence and Smart Environments*, vol. 1, pp. 5–14, 2009.

[13] P. Clements, "A survey of architecture description languages," in *Proc. of the 8th international workshop on software specification and design*. IEEE Computer Society, March 1996, pp. 16–25.

[14] S. Vestal, "A Cursory Overview and Comparison of Four Architecture Description Languages," Honeywell, Tech. Rep., February 1993.

[15] P. Mishra and N. Dutt, "Architecture description languages," *IEEE proc. - Computers and Digital Techniques*, vol. 152, no. 3, p. 285, 2005.

[16] OSGi Alliance, "OSGi Service Platform Core Specification Release 4," 2005. [Online]. Available: http://www.osgi.org/download/r4v40/r4.core.pdf

[17] V. Hourdin, J. Tigli, S. Lavirotte, G. Rey, and M. Riveill, "SLCA, composite services for ubiquitous computing," in *Proc. of the International Conference on Mobile Technology, Applications, and Systems*. New York, New York, USA: ACM Press, 2008, pp. 1–8.

[18] Ecma International, "ECMA-262: ECMAScript Language Specification," December 2009. [Online]. Available: http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-262.pdf

[19] J. Encarnaçao and T. Kirste, "Ambient intelligence: Towards smart appliance ensembles," *From Integrated Publication and Information Systems to Information and Knowledge Environments*, no. December, pp. 261–270, 2005.

[20] F. Hamoui, M. Huchard, C. Urtado, and S. Vauttier, "Specification of a component-based domotic system to support user-defined scenarios," in *Proc. of 21st International Conference on Software Engineering and Knowledge Engineering (SEKE 2009)*, July 2009.

[21] ——, "Un système d'agents à base de composants pour les environnements domotiques," in *Actes de la 16ème conférence francophone sur les Langages et Modèles à Objets (LMO 2010)*, Mars 2010, pp. 35–49.

[22] UPnP Forum, "Understanding UPnP: A White Paper," 2000. [Online]. Available: http://www.upnp.org/download/UPNP_UnderstandingUPNP.doc

[23] C. Bettstetter and C. Renner, "A comparison of service discovery protocols and implementation of the service location protocol," in *Proc. of the 6th EUNICE Open European Summer School: Innovative Internet Applications*. Citeseer, 2000, pp. 13–15.

[24] G. Aschemann, R. Kehr, and A. Zeidler, "A Jini-based Gateway Architecture for Mobile Devices," *In Proc. of the Java-Informations-Tage (JIT99)*, p. 203–212, September 1999.

[25] OASIS, "Web services business process execution language version 2.0," april 2007. [Online]. Available: http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf

[26] OSGi Alliance, "OSGi Service Platform Enterprise Specification," pp. 15 – 27, march 2010. [Online]. Available: http://www.osgi.org/download/r4v42/r4.enterprise.pdf

[27] C. Escoffier and R. Hall, "Dynamically adaptable applications with iPOJO service components," *Proc. of the 6th international conference on Software composition*, pp. 113–128, 2007.

[28] H. Cervantes and R. Hall, "Autonomous adaptation to dynamic availability using a service-oriented component model," in *International Conference on Software Engineering (ICSE)*. IEEE, 2004, pp. 614–623.

[29] Sun Microsystems, "Enterprise javabeans specifications," may 2006. [Online]. Available: http://java.sun.com/products/ejb/docs.html

[30] S. Means and M. A. Bodie, *Book of SAX: The Simple API for XML*. No Starch Press, 2002.

[31] S. Chiba and M. Nishizawa, "An Easy-to-Use Toolkit for Efficient Java Bytecode Translators," *Proc. of the 2nd international conference on Generative programming and component engineering*, pp. 364–376, 2003.

[32] Apache Foundation, "ipojo api," 2010. [Online]. Available: http://felix.apache.org/site/apache-felix-ipojo-api.html

[33] Yahoo, "Rewire the Web." [Online]. Available: http://pipes.yahoo.com/pipes

[34] Apple, "Automator: Your personal Automation Assistant." [Online]. Available: http://www.macosxautomation.com/automator

[35] M. Resnick, J. Maloney, A. Monroy-Hernandez, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman, and Y. Kafai, "Scratch: Programming for Everyone," *Communications of the ACM*, vol. 52, no. 11, pp. 60–67, 2009.

SC-xScript: An Embedded Script Language for Scientific Computation in Embedded Systems

Reng Zeng Yu Huang Su Liu
Peter J. Clarke Xudong He

Florida International University
Miami, Florida 33199, USA

Gwendolyn W. van der Linden
Jon L. Ebert

SC Solutions, Inc.
Sunnyvale, CA 94085, USA

***Abstract*—As embedded systems become more widespread in industry it is important to investigate new ways of maintaining these systems that are less time consuming and conform to the industry standards for quality assurance. One of the key software components in many embedded systems is the controller logic whose responsibility is the coordination of sensors, actuators, user interfaces and machine interfaces. Making changes to the controller logic without having to recompile the entire system can offer major benefits to the development process.**

This paper introduces a lightweight scripting language for embedded systems, SC-xScript, and the environment that supports the compilation and interpretation of Sc-xScript programs. Using Sc-xScript, application engineers can rapidly and safely test changes to the control logic, customize a product for different customers or applications, and add extra functionality to applications, all without requiring a new release of the underlying embedded software product. The strength of SC-xScript is its small size, simplicity and portability making it suitable for embedded systems. SC-xScript combines procedural constructs with matrix driven data structures.

I. Introduction

An important aspect of developing and maintaining control systems for advanced processing equipment is making changes to the controller logic, to allow for enhancing the controller performance and adapting to changes in the system design or functional specifications. The controller logic is often part of a larger embedded software application that is responsible for integrating sensors, actuators, user interfaces and machine interfaces, and implementing the required functionality. Making changes to these embedded applications is often time consuming due to the quality assurance procedures commonly used in industry. Those procedures frequently involve several people at multiple departments, such as application engineering and software engineering, and require rigorous inspection and testing. By incorporating scripting into the embedded control logic application with a carefully restricted scope and strong error checking, application engineers can rapidly and safely test changes to the control logic, customize a product for different customers or applications, and add extra functionality to applications, all without requiring a new release of the underlying embedded software product.

For these and other reasons, embedded scripting is gaining popularity in the design of software systems and applications. In the programming paradigm of embedded scripting, an interpreter is embedded into a binary application program. The application can execute script code through the embedded interpreter. Many interpreters are available for embedded scripting: Tcl/Tk [1], Python [2], Ch [3], and Lua [4], among others. However, none of the available solutions satisfy the requirements of the different embedded platforms one encounters in the field. In many cases it is not allowed to make system calls in task code, and the code must meet real-time execution requirements, and has limited memory available.

The above restrictions have led to the development of SC-xScript, a lightweight embedded script language for embedded systems that combines a simple procedural syntax with matrix driven data structures. SC-xScript is statically typed, uses parametric polymorphism for functions, and the implementation runs by interpreting bytecode generated by its compiler. It has automatic memory management of a single contiguous heap where bytecode, data and temporary variables are stored. SC-xScript can process matrix expressions, and it currently only supports two dimensional matrices. SC-xScript is small and simple, suitable for embedded systems. For porting purposes, it is implemented in ANSI C, and does not use system calls such as `malloc()` and `printf()`.

SC-xScript has the following properties:

1) Declares local variables explicitly with their size and type while input and output arguments of functions are only named to provide parametric polymorphism. Meanwhile a static type system is maintained by checking each input argument and output argument of functions against each function invocation for type safety.
2) Provides a scripting language supporting linear algebra operations. Inconsistencies or constructs that are hard to parse are avoided.
3) Uses pass-by-reference or pass-by-value only and does not support pointer types.
4) Script compilation happens at initialization.
5) Allows compile-time verification of suitable matrix dimensions for each operation, including the operations related to input and output arguments of functions against each function invocation.
6) Allows compile-time verification of the required heap size; recursive functions are not allowed.
7) Allows the running of multiple scripts in one program.

Scripts are compiled at embedded system initialization and then executed at each interval (the scripts typically run periodically at a fixed interval). Before each execution call the input variables are set from the calling program and after the call the output variables are read by the calling program. The input and output variables are agreed upon for each embedded application, and often include sensor readings, parameters, flags, reference signals, and actuator commands.

The rest of the paper is organized as follows. In Section II, we discuss the syntax of the language SC-xScript. In Section III, we present the overall design of our implementation. In Section IV, we present our static type system including parametric polymorphism. In Section V, we discuss the related works. In Section VI, we conclude the paper.

II. Syntax of the Script Language

Script languages are distinct from the core code of the application, which is usually written in a different language. Script languages are interpreted from source code, while the applications they control are traditionally compiled to native machine code. SC-xScript, as introduced in this paper, is a simple script language inspired by the Matlab [5] and GNU Octave [6] languages with influences from C, which can be easily embedded into C/C++ applications with the help of an interpreter for the script language.

The complete grammar of SC-xScript is presented in the Appendix. The language consists of three parts: variable declaration, function definition and statements, in which the function definition part is optional (Line 1 of the Appendix). The first part of the language is used to declare variables, in which each variable is a matrix whose elements are of type integer or double (Lines 2-10 of the Appendix). The second part of the language is optional, and is used for function definitions. As Lines 12-15 of the Appendix show, both input arguments and output arguments are only a variable name (identifier) without any type information. It offers the flexibility of writing a single function that handles different types of arguments. The third part of the language contains the statements, in which each statement could be an `assignment` statement, `if` statement, `iterative` statement or `jump` statement. For `assignment` statements (Lines 20-22 of the Appendix), it could be either assigning the value of an arithmetic expression to a variable or an element of a variable, or assigning the return value of a function call to return variables. The `if` statement, `jump` statement and `iterative` statements (including `while` loop and `for` loop) are similar to the C language (Lines 23-29 of the Appendix).

III. Overall System Design

Figure III shows the subsystems and the relationships between the subsystems. The `ScriptAPI` contains the functions that invoke the `Compiler` and the `Interpreter` subsystems. The `Compiler` and `Interpreter` use the `HeapManager` during the compilation and interpretation of the script to manage the allocated heap. The heap is a chunk of memory allocated to the SC-xScript Runtime

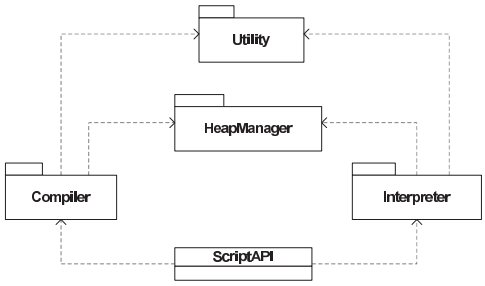


Figure 1. The Architecture of Implementation

System when it is invoked. The `HeapManager` maintains the heap and provides the functions required to allocate memory within the heap for the objects in the `Compiler` and the `Interpreter`. The `Interpreter` does not contain any subsystems and its main purpose is to execute the bytecodes generated by the `Compiler`. The `Utility` contains several components used by the `Compiler` and the `Interpreter`.

A. Heap Management

This section explains how we organize the heap. The heap manager maintains the heap and provides the functions required to allocate memory for the objects in the Compiler and the Interpreter. The structure of the heap is shown in Table I. The first row of the table shows the size in bytes for those sections that are restricted in size. The second row shows the names assigned to the different sections on the heap.

There are several registers reserved in the beginning of heap as below.

1) PC, the program counter for the next bytecode to execute;
2) SP, the logical address of the beginning of Stack section;
3) DP, the logical address of the beginning of Data section;
4) ST, the logical address of the beginning of Symbol Table section;
5) RS, the result code;
6) MR, the message reference.

The starting offset in the heap for Instructions is placed at the end of registers. For the Data, Symbol Table Data, Stack and Temp sections, the starting offset in the heap is predefined as a ratio. All addresses used by the Instruction and Data sections are logical addresses, not physical addresses, which makes it easy to move sections in case one section is full while there is still space in other sections.

B. Bytecode Design

The compiler uses no intermediate representation; it emits bytecodes for the interpreter "on the fly" as it parses a script. The bytecodes are not using the traditional stack based implementation. Traditional stack based implementations for bytecodes usually have most values stored on the stack, while only a limited number of registers are used for instruction execution, and the instructions usually are referring to their operands implicitly. In SC-xScript most bytecodes use a three-address format, as depicted in Figure 2, and refer to their

Table I
LAYOUT OF HEAP (B REPRESENTS BYTES.)

4B	4B	4B	4B	4B	4B					
PC	SP	DP	ST	RS	MR	Instructions	Data	Symbol Table	Stack	Temp

Figure 2. Bytecode format

operands explicitly. In this way, excessive copying of values is avoided during pushing and popping to/from the stack, which is expensive for matrix manipulation, and reduces the total number of generated bytecodes. SC-xScript also uses part of the stack to store activation records of functions. Ierusalimschy et al. [4] use a similar mechanism called register based virtual machine, and state that the overhead brought by code size and decoding for register based virtual machine is ameliorated by several factors.

As for code size overhead, the bytecodes in a register based virtual machine have to specify their operands, three operands in our case, so it typically results in larger bytecodes than a stack based virtual machine. However, register based virtual machines usually generate less bytecodes than the stack based virtual machine, so there is not much difference in the total size. As for the decoding overhead, register based virtual machines have to decode their operands from the bytecodes, while stack based virtual machines typically use bytecodes with implicit operands. However, stack based virtual machines also spend some time manipulating implicit operands, and bytecodes in stack based virtual machines frequently need multi-byte operands, which make it impossible to fetch the operand at once because of the alignment. Davis et al. [7] argue in defense of register-based virtual machines and provide hard data on the improvement of Java bytecode. Winterbottom and Pike [8] also defend register based virtual machines based on their suitability for on-the-fly compilation.

IV. TYPE SYSTEM

SC-xScript has only two types: `INTEGER` matrix and `DOUBLE` matrix, and maintains a static type system. Each variable is declared explicitly with its size and type, except input and output parameters of functions. The input and output parameters of functions are only named without specifying their size and type, thus making SC-xScript more expressive. In other words, the functions and their input and output parameters are generic to handle values identically without depending on their type, providing abstraction within the script. In each statement of a function, the input and output parameters are checked for type safety against each function invocation, thus maintaining full static type safety. Therefore, SC-xScript has a static type system, and uses parametric polymorphism for functions. The script in SC-xScript is to be used as a whole by external programs, and functions cannot be accessed by external programs.

A. Storage of Type Information

For SC-xScript, types are attached to values rather than to variables. Each value stored in either the Data or Temp section has a 32-bit header to store the type information. The dimension of each matrix is specified in the script. A single integer is represented in an `INTEGER` matrix with one row and one column, and a single double is represented in a `DOUBLE` matrix with one row and one column. By using a matrix to represent each data item, all data operations become matrix operations. This greatly simplifies the data manipulation making SC-xScript small and fast. Besides `INTEGER` matrix and `DOUBLE` matrix as user defined types, SC-xScript has one internal type: `REF`, which is not to be declared by the script, it is for internal usage only. Each parameter of a function in SC-xScript, including input parameters and return values, is only represented by an `ID` without specifying the type information. Therefore, the type of parameters could be different in each function invocation, depending on the data passed by the function invocation. `REF` is introduced as a reference to a matrix to allow function supporting matrix with different dimensions as input or output.

Representation of values in the heap is shown in Table II. The first row shows the number of bits allocated to each field. The second, third, and fourth rows contain the type information for the data in the matrix. Our implementation of SC-xScript uses 3 bits to indicate whether the matrix is a `INTEGER` matrix, `DOUBLE` matrix or a `REF/NREF` matrix. In the future, we can also use these 3 bits to indicate `INTEGER` scalar and `DOUBLE` scalar to improve performance. One bit is reserved for type checking purpose to indicate whether there is data following the header, because the matrix header is required during type checking to store the numbers of rows and columns, but additional space is not necessary to be allocated for the matrix values. Our implementation of SC-xScript uses 14 bits to store the number of rows, supporting up to 16383 rows. It also uses 14 bits to store the number of columns, supporting up to 16383 columns. Because the matrix values are stored right after the matrix header, it is not necessary to use any bits to store the offset of matrix values. For the `REF` matrix, we use 28 bits to store the address of the destination matrix, which is the offset in the data section of the heap. Hence our implementation of SC-xScript supports a data section of up to 256MB, which should be sufficient for the intended applications.

To define the structure shown in Table II in a compact manner in ANSI-C, if using a tagged union as shown in Program 1, proper memory alignment will result in each matrix header needing 64 bits instead of 32 bits, which consumes unnecessary space on the heap.

To store the matrix header as compactly as possible, SC-

Table II
LAYOUT OF SC_MATRIX IN THE HEAP.

3bits	1bit	14bits	14bits	0bit
INT	tc	Number of Rows	Number of Columns	Data Offset
DOUBLE	tc	Number of Rows	Number of Columns	Data Offset
REF	tc	Offset of target SC_Matrix		
NREF	tc	Offset of target SC_Matrix		

Program 1 Data Structure of Matrix using tagged union

```
1  typedef struct
2  {
3    unsigned rows:14;
4    unsigned cols:14;
5  }Data;
6  typedef struct {
7    unsigned type: 3;
8    unsigned tc: 1;
9    union{
10     Data d;
11     unsigned ref;
12   } u:28;
13 }SC_Matrix;
```

xScript uses 14 bits for `rows` and 14 bits for `cols` to represent each matrix header while introducing two functions to read or write the offset of reference in those two 14-bits. `MatrixRefRead` reads the offset of reference based on calculation from `rows` and `cols`, and `MatrixRefWrite` writes the offset of reference back to `rows` and `cols`.

B. Type checking for Functions

The statements in functions demand a special strategy for type checking, because each function can be called multiple times with different types of variables. Each statement in a function should be checked against each function invocation, because the parameters of a function are just variable IDs without any type information specified; the type information is specified at each function invocation. For example, suppose there is one function defined as `function a=mult(b, c)`, followed by two invocations of the function, as shown in Program 2, are legal because the statement `a=b*c` is legal for each function invocation.

The script shown in Program 3 is illegal, because in the second function invocation `A2=mult(B2,C2)`, the statement

Program 2 Legal script for multiple times of calling a function

```
1  int A1[2,3], B1[2,4],C1[4,3];
2  int A2[8,9], B2[8,15],C2[15,9];
3  function a = mult(b, c)
4  {
5    a=b*c;
6  }
7  A1=mult(B1,C1);
8  A2=mult(B2,C2);
```

Program 3 Illegal script with multiple function calls

```
1
2  int A1[2,3], B1[2,4],C1[4,3];
3  int A2[8,9], B2[8,15],C2[15,20];
4  function a = mult(b, c)
5  {
6    a=b*c;
7  }
8  A1=mult(B1,C1);
9  A2=mult(B2,C2);
```

in function `a=b*c` is illegal, as `a` is a matrix with 8 rows and 9 columns, while `b*c` returns a matrix with 8 rows and 20 columns.

SC-xScript does not check the function definition itself, since there is no type information attached to the parameters. Instead, it checks each function invocation. For each function invocation, the parameters have a known and fixed type, which is stored with the parameters. Taking Program 3 as an example, `each of a`, `b` and `c` is a `REF`, so when the type checking algorithm arrives at the statement `A2=mult(B2,C2)`, it points `a` to `A2`, `b` to `B2` and `c` to `C2`. Section IV-A explains how a `REF` variable points to another variable.

C. Code Generation for Functions

It seems straightforward to generate bytecodes for a function call: just jump to the beginning of the desired function. However, it is required to take the function return into consideration. In SC-xScript, there is no recursion, but it allows a function to be called multiple times, and it allows a function to call another function. Therefore, it is required to know where to return for each function invocation. For example, a function `f1` calls a function `f2`, the function `f2` calls a function `f3`, so when the function `f3` is about to return, there should be a place to store the return address of functions `f1`, `f2` and `f3`. SC-xScript implements this by introducing a small stack section in the heap to store the return addresses.

When calling a function, the generated bytecodes push a return address onto the stack, and then jump to the beginning of the desired function. When returning from a function, the generated bytecode pops an item from the stack as the return address, and jumps to the return address.

In the bytecodes for function definition, both input and output parameters are represented by a variable of type `REF`. At each function invocation it must be determined where the `REF` variable must point to.

To summarize, when calling a function the generated bytecodes must take the following steps.

1) Bind the given address of parameters to each `REF` variable. Taking Program 2 as an example, bind `a` to `A2`, bind `b` to `B2` and bind `c` to `C2`.
2) Push the return address onto the stack.
3) Jump to the beginning of the desired function.
4) Execute the statements in the function.
5) Pop one item from the stack as the return address.
6) Jump to the return address.

Table III
BYTECODES GENERATED FOR PROGRAM 2

OpCode	Operand 1	Operand 2	Operand 3	Description
MUL	1464	1456	1460	d=b*c
MOV	1452	1464	0	a=d
RJMP	0	0	0	Function Return
EXE	0	0	0	Mark the beginning of script
REFWR	1452	16	0	REFWR a, A1
REFWR	1456	44	0	REFWR b, B1
REFWR	1460	80	0	REFWR c, C1
FJMP	0	8	0	Call function at 0 with return address 8
REFWR	1452	132	0	REFWR a, A2
REFWR	1456	424	0	REFWR b, B2
REFWR	1460	908	0	REFWR c, C2
FJMP	0	12	0	Call function at 0 with return address 12
END	0	0	0	Mark the end of script

There is additional REF variable required for the temporary variable in function definition. Taking Program 2 as an example, a=b∗c leads to two bytecodes using a temporary d as follows.

$$d = b * c$$
$$a = d$$

Each function definition has only one corresponding copy of bytecodes, and each function invocation jumps to the same beginning address of the function. Therefore, the bytecodes for each function definition cannot include information on how much memory is required for the temporary variable d, since it is unable to determine the size of b∗c until the function invocation. Consequently, the compiler cannot figure out the space requirement of the temporary variables for the function, and must manage the temporary variables dynamically. The management of REF variables is explained in section IV-D.

D. Interpreter Design for Functions

Each function definition has only one corresponding copy of bytecodes, and each statement calling a function jumps to the same beginning address of the function, so the generated function bytecodes have to take care of the different types of parameters. For example, in Program 2, a=b∗c leads to two bytecodes using a temporary d as follows.

$$d = b * c$$
$$a = d$$

As mentioned before, each parameter does not have a fixed type, and depends on the specific function invocation. SC-xScript uses the REF type for the temporary variables as well as the parameters. However, the strategy to handle with REF type for the temporary variables is different than the one to handle the REF type for the parameters. Regarding the REF type for the parameters, the statement calling function can simply bind it to a logical address, so for the statements in the function definition the interpreter only needs to resolve REF variables to get the real values. However, in case of the REF type for the temporary variables, the interpreter needs to determine the size of the temporary variable dynamically and allocate the memory in the TEMP section of the heap. In the above example, b and c are INT or DOUBLE variables or REF variables, which could be resolved to INT or DOUBLE variables. Therefore, the size of d can be determined during the interpretation of the bytecodes, and the interpreter can allocate memory in the TEMP section of the heap for d, and bind d to the logical address of newly allocated memory.

Table IV
DATA SECTION FOR PROGRAM 2 - PHASE 0

Variable	Address	Type	Address of Destination Matrix
a	1452	REF	0
b	1456	REF	0
c	1460	REF	0
d	1464	REF	0

Since there are two different strategies to deal with the REF variables, SC-xScript has to figure out which strategy to take for a REF variable. Table III gives the example bytecodes for Program 2, Table IV gives the data layout of function variables in data section when the interpreter starts executing bytecodes, and Table V gives the data layout of function variables when the interpreter enters the first function call. When the interpreter executes the bytecode in function definition MUL 1464 1456 1460, it can resolve 1456 and 1460 to a destination matrix, while it cannot resolve 1464 since 1464 is pointing to 0, which actually means it is waiting for memory to be allocated. So for operand 1, which is 1464, the interpreter first needs to allocate memory in the TEMP section of the heap, according to the operation code and the size of matrices referred by operand 2 and operand 3. For example, the matrices referred by operand 2 and operand 3 are of size [2,4] and [4,3] accordingly, then the interpreter needs to allocate memory for operand 1, which is a REF type, with a matrix of size [2,3], because it is a matrix multiplication.

Table VI gives the data layout of function variables after the execution of the first function call, in which the temporary variable d is referring to the destination matrix at 4 in the TEMP section of the heap. The issue is then: when the interpreter executes the second function call, for the bytecode MUL 1464 1456 1460, the first operand 1464 is referring

Table V
DATA SECTION FOR PROGRAM 2 - PHASE 1

Variable	Address	Type	Address of Destination Matrix
a	1452	REF	16
b	1456	REF	44
c	1460	REF	80
d	1464	REF	0

Table VI
DATA SECTION FOR PROGRAM 2 - PHASE 2

Variable	Address	Type	Address of Destination Matrix
a	1452	REF	16
b	1456	REF	44
c	1460	REF	80
d	1464	REF	4

to 4, which is a matrix with size [2,3]. However, for this function call, it requires a matrix with size [8,9]. Then how the interpreter could be able to distinguish these specific `REF` variables from other `REF` variables for memory allocation? To resolve this, SC-xScript introduces an additional type, `NREF`, to provide temporary variables generated by the compiler for functions. This way, when the bytecode is going to write to a temporary variable marked as `NREF` type, it can always allocate memory for that.

V. RELATED WORK

A number of languages have been designed for replacing application-specific scripting languages by being embeddable in application programs. The application programmer (working in C or another system language) includes "hooks" where the scripting language can control the application. These languages serve the same purpose as application-specific extension languages but with the advantage of allowing some transfer of skills from application to application. JavaScript began as, and primarily still is, a language for scripting inside web browsers; however, the standardization of the language as ECMAScript has made it popular as a general purpose embeddable language. In particular, the Mozilla implementation SpiderMonkey is embedded in several environments such as the Yahoo! Widget Engine. Other applications embedding ECMAScript implementations include the Adobe products Adobe Flash (ActionScript) and Adobe Acrobat (for scripting PDF files).

Lua [9] is a lightweight, reflective, imperative and functional programming language, designed as an embedded scripting language. It is not interpreted directly from the textual Lua file, but is compiled into bytecode, which is then run on the Lua virtual machine. The difference of our work with Lua is that, in our work SC-xScript, everything is a matrix, that makes it easy for matrix operations. In Lua, Tables are the most important data structure. Our work offers a way for embedded system to easily use scripts for scientific computation.

Regarding scientific computation, two categories of general scientific software can be identified as 1) computer algebra systems that perform extensively symbolic mathematical evaluations such as Maple [10] and Mathematica [11] and 2) matrix computation systems that are designed for numerical computations and are well suited for engineering applications such as the Matlab [5] that dominates at the commercial market and the open source "clones" Scilab [12] and Octave [6]. Chonacky and Winch [13] offers an excellent comparative review of three well-established commercial products. The open source jLab [14] environment extends the potential of Java for scientific computing. It provides a Matlab/Scilab like scripting language that is executed by an interpreter implemented in the Java language. The scripting language supports the basic programming constructs with Matlab like matrix manipulation operators.

The control engineering design and analysis problems have grown increasingly complex with both technological breakthroughs and theoretical advances over the last few decades. A number of software packages have been developed to bridge the gap between modern control theory and the software/hardware implementation. The existing well-known interpretive software packages, such as MATLAB Control System Toolbox [15], MATRIXx [16], and Mathematica's Control System Professional [17] are commercially available for the purpose of computer-aided control system design, that offer scientific computation toolboxs to solve the complex design and analysis problems in the host, while our work is to deploy the scientific computation capability in the target embedded devices.

VI. CONCLUSIONS

In this paper we present SC-xScript, a lightweight script language for embedded systems, which combines simple procedure construct syntax with matrix driven data structures. It is statically typed, and the implementation runs by interpreting bytecodes generated by its compiler, has automatic memory management of a single contiguous heap where bytecode, data and temporary variables are stored. Its strength is in processing matrix expressions, and matrix is the only data structuring mechanism. It is small and simple, suitable for embedded systems. It is implemented in about 40,000 lines of code.

We attach the type information into the values instead of variables, so that when executing the generated bytecodes the interpreter does not have to do symbol table lookups to check the type information for operands, which saves time for each bytecode execution. As the matrix is the only data structuring mechanism and SC-xScript is statically typed, attaching the type information into the values by adding a matrix header make it simple to view the data section of the heap for debugging purpose.

We introduce a `REF` type to allow function definition to support different parameters, which is important for some functions as they may have to handle with all kinds of matrix size. Otherwise, the script user would need to identify the matrix size for each parameter to make the type fixed, and design functions with the same functionality for different size of a matrix.

We carefully designed the heap management to support both compiling and execution based on the same fixed heap, managing the space for bytecodes, data, stack and temporary

data. As the heap size is fixed, it may be possible that there is not enough space for the data section while there is still space left in the temporary section. SC-xScript solves this by supporting the movement of memory sections, which is simplified by using logical addresses.

By supporting matrix data and matrix expressions, while keeping small and simple, SC-xScript is suitable for embedded systems that require an embedded scripting language for scientific computation.

SC-xScript is portable to be embedded into any C/C++ programs, as it is implemented in ANSI C.

Further work may include optimizing the generated byte-codes to save space in the heap and improve the execution efficiency.

ACKNOWLEDGMENTS

This work was partially supported by the NSF of U.S. under award HRD-0833093, IIP-0450482 and Presidential Fellowship of Florida International University.

APPENDIX

Following is the completed Extended Backus-Naur Form (EBNF) specifying the grammar of SC-xScript.

```
prog = decllist, [funclist], statementlist;
decllist = {type, matrixlist, ";"};
type = "int" | "double";
matrixlist = matrixdef {",", matrixdef};
matrixdef = identifier, [ "[", (integer, [",", integer]
     | identifier, [",", identifier]) "]" ], ["=",
    matrixexp];
matrixexp = integer, [":", integer] | real | matrixleft
     |       "(", scalarlist, ")", ["'"] | "[",
    scalarlist, "]", ["'"] | identifier,"[","[",
    selector,"]",",","[",selector,"]","]";
scalarlist = selector, {";", selector};
selector = matrixrange, {",", matrixrange};
matrixrange = matrixindex, 2 * [":", matrixindex];
matrixindex = identifier | integer | real;
funclist = funcdec, {funcdec};
funcid = "function", (identifier | "[", outputarglist,
    "]"), "=", identifier, "(", inputarglist, ")";
funcdec = funcid, "{", decllist, statementlist, "}";
outputarglist = identifier, {",", identifier};
inputarglist = identifier, {",", identifier};
funccall = identifier, "(", actualarglist, ")";
actualarglist = arithmeticexp,{",",arithmeticexp};
statementlist = {statement};
statement = assignment|ifstatement|iterstatement|
    jumpstatement;
assignment =  matrixleft,"=",(arithmeticexp|funccall)
    ,";" | leftexp,"=",(funccall|"size","(",
    arithmeticexp,")"),";";
matrixleft = identifier,["[",matrixrange,[",",
    matrixrange],"]"];
leftexp = "[", identifier, ",", identifier, "]";
ifstatement = "if", "(", logicexp, "), "{",
    statementlist, "}", ["else", ("{", statementlist,
    "}" | statement)];
iterstatement = whileloop | forloop;
whileloop = "while","(",logicexp,")","{",statementlist
    ,"}";
forloop = "for", "(", init, ";", logicexp, ";",
    increment, ")", "{", statementlist, "}";
init = identifier, "=", integer;
increment = ("++"|"--"), identifier | identifier,
    ("++"|"--") | identifier, "=", identifier,
    ("+"|"-"), integer;
jumpstatement = ("continue"|"break"), ";";
arithmeticexp = matrixexp | builtinfunc,"(",
    arithmeticexp,")" | arithmeticexp
    ,("+"|"-"|"*"|"/"|".*"|".^"|"./"),arithmeticexp;
logicexp = arithmeticexp,("<"|">"|"<="|">="|"=="|"!="),
    arithmeticexp | logicexp,("&&"|"||"),logicexp |
    "!",logicexp;
builtinfunc = "sin" | "cos" | "exp";
identifier = alphabetic, { alphabetic | digit } ;
alphabetic = "_"|"A"| ... |"Z"|"a"| ... |"z" ;
digit = "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7"|"8"|"9";
real = (integer | decimal), ["E", integer];
decimal = integer, ".", unsigned | integer, "." | ".",
    unsigned;
integer = ["-"], unsigned;
unsigned = digit, {digit};
```

REFERENCES

[1] J. K. Ousterhout, *Tcl and the Tk Toolkit*. Addison Wesley, 1996.

[2] G. van Rossum, "Scripting the Web with Python," *World Wide Web J.*, vol. 2, no. 2, pp. 97–120, 1997.

[3] H. H. Cheng, "Scientific Computing in the C^H Programming Language," *Sci. Program.*, vol. 2, no. 3, pp. 49–75, 1993.

[4] R. Ierusalimschy, L. de Figueiredo, and W. Celes, "The Implementation of Lua 5.0," *Journal of Universal Computer Science*, vol. 11, no. 7, pp. 1159–1176, 2005.

[5] D. J. Higham and N. J. Higham, *MATLAB guide*, 2005.

[6] J. W. Eaton., *GNU octave manual*, Network Theory Ltd, 2002.

[7] B. Davis, A. Beatty, K. Casey, D. Gregg, and J. Waldron, "The case for virtual register machines," in *IVME '03: Proceedings of the 2003 workshop on Interpreters, virtual machines and emulators*. New York, NY, USA: ACM, 2003, pp. 41–49.

[8] P. Winterbottom and R. Pike, "The design of the Inferno virtual machine," in *In IEEE Compcon*, 1997, pp. 241–244.

[9] R. Ierusalimschy, L. H. de Figueiredo, and W. C. Filho, "Lua—an extensible extension language," *Softw. Pract. Exper.*, vol. 26, no. 6, pp. 635–652, 1996.

[10] E. Kreyszig, *Advanced Engineering Mathematics: Maple Computer Guide*. New York, NY, USA: John Wiley & Sons, Inc., 2000.

[11] M. Trott, *The mathematica guidebook: programming*. Berlin: Springer, 2004.

[12] N. R. Campbell Stephen, Chancelier Jean-Philippe, *Modeling and Simulation in Scilab/Scicos*. Springer, 2006.

[13] N. Chonacky and D. Winch, "Reviews of Maple, Mathematica, and Matlab: Coming Soon to a Publication Near You," *Computing in Science and Engineering*, vol. 7, no. 2, pp. 9–10, 2005.

[14] S. Papadimitriou and K. Terzidis, "jLab: Integrating a scripting interpreter with Java technology for flexible and efficient scientific computation," *Comput. Lang. Syst. Struct.*, vol. 35, no. 3, pp. 217–240, 2009.

[15] A. Grace, J. L. Alan, J. N. Little, and C. M. Thompson, *Control system toolbox for use with Matlab: User's guide*. Infoscience | Ecole Polytechnique Federale de Lausanne (Switzerland), 1995.

[16] R. Walker, J. Gregory, C., and S. Shah, "MATRIXx: A data analysis, system identification, control design and simulation package," *IEEE Control Systems Magazine*, vol. 2, no. 4, pp. 30 – 37, 1982.

[17] B. Palancz, Z. Benyo, and L. Kovacs, "Control System Professional Suite," *IEEE Control Systems Magazine 25*, no. 4, pp. 67–75, 2005.

Context-aware Services for Multiple-Users

Ichiro Satoh
National Institute of Informatics
2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan
E-mail: ichiro@nii.ac.jp

Abstract

This paper presents a framework for providing context-aware services in public spaces, e.g., museums. The framework is unique among other existing context-aware systems in implementing services as mobile agents and supporting groups of users in addition to single users. It maintains a location model as containment relationships between digital representations, called virtual counterparts, corresponding to people, terminals, or spaces, according to their locations in the real world. When a visitor moves between exhibits in a museum, it dynamically deploys his/her service provider agents at the computers close to the exhibits via virtual counterparts. When two visitors stand in front of an exhibit, service-provider agents are mutually executed or configured according to the member of the visitors. To demonstrate the utility and effectiveness of the system, we constructed location/user-aware visitor-guide services and experimented with them for two weeks in a public museum.

1 Introduction

Many of the early applications of pervasive computing have focused on interaction between individual users and their environments [1, 2]. They have paid little attention to environments that might effectively sense and respond to groups of co-located people. However, much of our time is spent in shared physical spaces, e.g., offices, factories, schools, and public spaces, so it is important to consider how an environment might effectively sense and respond to groups of co-located people as well as individual people.

When there are multiple users in a space, context-aware services should be customized to according to the combination of people in current places in addition to the people themselves. For example, when a married couple stands in front of digital signage, advertising content displayed on the signage should be specific to neither mens' nor women' items. When a visitor stands in front of an exhibit in a museum, where another visitor is already standing, annotation services provided from a terminal close to the exhibit may be provided to the existing visitor, and then the new visitor sequentially, or customized to the two visitors.

This paper presents a framework for providing context-aware services to not only individual people but also groups of co-located people in specified spaces. Since such services tend to depend on the co-location of people, the framework maintains a location model for the real world.

2 Background

This section describes basic ideas behind the framework presented in this paper.

2.1 Example Scenario

Suppose a context-aware visitor-guide system is in a museum. Most visitors to museums lack sufficient knowledge about exhibits in the museum and they need supplementalary annotations on these. However, as their knowledge and experiences are varied, they may become puzzled (or bored) if the annotations provided to them are beyond (or beneath) their knowledge or interest. User-aware annotation services about exhibits are required. For example, when a user stands in front of an exhibit, an annotation service about the exhibit is provided in his/her personal form on a stationary terminal close to the exhibit. However, there are multiple users in a public space. Suppose a visitor arrives in front of an exhibit, where another visitor is standing and receiving an annotation service from on a terminal there. We have several approaches to solving this. 1) After the annotation service for the existing visitor has finished, an annotation service for the new visitor is provided. 2) The annotation service for the existing visitor is shared by and customized to the existing and new visitors. In fact, we constructed and provided such a context-aware visitor-guide system to several museums [3]. This problem was serious during rush hours at the museums.

2.2 Design Principles

To provide services according to groups of co-located people, we need to model the locations of the people and the terminals that provide the services. This framework maintains a symbolic location model about their locations to select and customize services. It maintains the locations of people, physical entities, and places in the real world as structural relationship between programmable entities, called *virtual counterpart* objects, corresponding to them.

It uses the spatial co-location of physical entities as a primary attribute for selecting communication partners. Communication partners, including terminals, should also be selected according to their co-locations. Co-locations between people and terminals are modeled as a spatial relationship between the virtual counterparts corresponding to the people and the terminals. The framework allows us to explicitly define the condition that activate and configure services provided to the groups of people in the same space. The condition is defined as a combination of relationships between slots in a counterpart. As seen in 1, each slot has one input port and one or more output ports to receive or issue events. The output ports can be classified into the following types.

- *Arriving* port: when a slot receives a counterpart, it issues an event to another counterpart from the port.
- *Staying* port: when a slot holds a counterpart, it periodically issues events to another counterpart from the port.
- *Left* port: when a slot sends a counterpart, it issues an event to another counterpart from the port.

The framework enables us to define a one-to-one connection from each of the output ports of a slot to be explicitly connected to the input port of another slot contained in the same counterpart. Also, we introduce many-to-one connections, in the sense they have multiple input ports and one output port.

- *And* connections: when it receives events from all its input ports, it issues an event to its output port.
- *Or* connections: when it receives events from at least one of its input ports, it issues an event to its output port.

Context-aware communication between counterparts are defined as a graph notation consisting of connections between ports.

We discuss differences between the framework presented in this paper and our previous frameworks. We earlier constructed a location model for pervasive computing environments [3]. Like the framework presented in this paper,

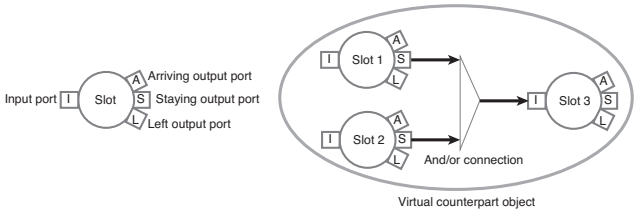


Figure 1. Slot and connection

the model represented spatial relationships between physical entities (and places) as containment relationships between their programmable counterpart objects and deployed counterpart objects at computers according to the positions of their target objects or places. We previously presented a context-aware museum guide system [4]. Our previous model and systems provided no support to location-aware communications and group-aware communications, unlike the framework presented in this paper.

3 Design and Implementation

Our framework consists of four subsystems: 1) context-aware directory servers, called CDSs, 2) virtual counterpart management systems, 3) agent runtime systems, and 4) service provider agents. The first is responsible for reflecting changes in the real world and the location of users when services are deployed at appropriate computers. Our system can consist of multiple CDSs, which are individually connected to other servers in a peer-to-peer manner. Each CDS only maintains up-to-date information on partial contextual information instead of on tags in the whole space. The second manages a structure of virtual counterparts according to up-to-date information on the state of the real world, such as the locations of people, places, and things.

The third is running on stationary computers, which are located at specified spots close to exhibits in a museum and are equipped with user-interface devices, e.g., display screens and loudspeakers. It is responsible for executing and migrating service-provider agents, like existing runtime systems for mobile agents [4]. The fourth is an autonomous entity that defines application-specific services for visitors. It is implemented as one or more mobile agents. Virtual counterparts are used as forwarders in the sense that they migrate mobile agents from one terminal to the next. The framework deploys and executes mobile agents at computers near the positions of the users instead of at remote servers. As a result, mobile agent-based content can directly interact with users, where RPC-based approaches, which other existing approaches are often based on, must have network latency between computers and remote servers. Mobile agents can help to conserve these limited resources, since each agent only needs to be present at the computer while the computer needs the content provided by that agent.

3.1 Location-sensing systems

The framework itself is independent of any location-sensing system. The management system has interfaces for monitoring its underlying location-sensing systems. Tracking systems can be classified into two types: proximity and lateration. The first approach detects the presence of objects within known areas or close to known points, and the second estimates the positions of objects from multiple measurements of the distance between known points. The CDSs support the two types, but they map geometric information measured by the latter sensing systems to specified areas, called *spots*, where the exhibits and the computers that play the annotations are located. This is because most context-aware services in public spaces should be provided within specified spaces rather than at specified geometric points. Each CDS has its own local database to maintain the locations of visitors and their agents. When a CDS detect the presence/absence of people, physical entities, or computers, it tries to discover virtual counterparts corresponding to them by sending other CDSs and runtime systems for virtual counterparts through UDP multicast communications.

3.2 Virtual counterpart management

The framework itself is independent of programming languages, but the current implementation uses Java (J2SE version 1.5 or later versions) as an implementation language to define the framework itself and virtual counterparts.

The framework manages a location model as an acyclic-tree structure of virtual counterparts, where each virtual counterpart can be defined as a self-contained computing entity without any description of a counterpart hierarchy. The management system provides a container for the counterpart corresponding to a virtual counterpart. The container technology developed by Enterprise Java Beans provides interfaces for components and enables them to transparently adapt to runtime services, e.g., it manages transactions. That is, this framework provides each agent corresponding to a counterpart with a wrapper, called a *tree container*. Each container includes its target agent, its attributes, and containment relationships between itself and its parent container and between itself and its child containers. As a result, a hierarchy of containers is maintained in the form of a tree structure, which has the component tree nodes of containers.

3.3 Virtual counterpart

Each virtual counterpart is defined as a mobile agent and a whole or part hierarchy of counterparts is maintained as a acyclic-tree structure of virtual counterparts. The framework provides agent runtime system(s) on computer(s). The system is responsible for managing and exchanging virtual counterparts and controls messages or counterparts with runtime systems running on different computers (if the framework is maintained in one or more computers). The runtime system was designed for this framework, but is similar to other existing Java-based mobile agent systems.

Each agent can have one or more activities implemented using the Java thread library. The system can control all the components in its container hierarchy, under the protection of Java's security manager. Furthermore, the system maintains the life cycle of the agents: initialization, execution, suspension, and termination. When the life cycle state of an agent changes, the system issues certain events to the agent and its descendent agents.

Counterpart migration within a container hierarchy occurs merely as a transformation of the tree structure of the container hierarchy. When one counterpart is moved to another in the same computer, a sub-tree, whose root corresponds to its container and branches, including counterparts, is migrated to the container that maintains the destination. When a counterpart can be deployed in another subtree maintained on another computer, the system marshals the state of the counterpart, e.g., instance variables and the counterpart embedded within its container sub-tree, into a bit-stream and transmits their serialized state program code through TCP sessions by using the underlying mobile agent runtime system.

3.4 Context-aware communication

Each service should be activated through connections between slots for target people or physical entities and the slot that contains it. Each of slots in a virtual counterpart has its attribute and can hold at most one virtual counterpart that can satisfy the attribute. Although the attributed can be explicitly defined, the current implementation supports built-in attributes corresponding to users, unknown visitors, administrators, terminals, and services. Each slot also has more output ports and one input port to receive events. The former ports issue events at certain changes in the structure of counterparts and the latter port forwards receiving events to the counterpart contained in its port's. Each connection are maintained as event queues between its source slots and its destination slots. Connections are defined in LISP-like expressions and then evaluate them by using a LISP-based interpreter.

3.5 Service-provider object

Virtual counterparts corresponding to terminals deploy service-provider objects at the terminals. Each service-provider object is automatically deployed at and maintains per-user preferences on a user and record his/her behavior,

e.g., exhibits that they have looked at. The service-provider object can also define user-personalized services adapted to the user and access location-dependent services provided at its current computer. Each service-provider object is spatially bound to, at most, one user. When a user gets closer to an exhibit, our system detects his/her migration by using location-sensing systems and then instructs the user's counterpart objects to migrate to a computer close to the exhibit. Mobile agent help to conserve limited resources, because each service-provider object only needs to be present at the computer for the duration the computer needs the services provided by that service-provider object. The framework is open to define connections as long as they are subclasses of the classes for built-in connections.

4 Experience

We constructed and carried out an experiment at the Museum of Nature and Human Activities in Hyogo, Japan, using the proposed system. We did the experiment over two weeks. Each day, more than 60 individuals or groups took part. The experimental environment provided several spots in front of exhibits, which were specimens of stuffed animals, e.g., a bear, deer, racoon dog, and wild boar in an exhibition room of the museum. Each spot could provide animation-based annotative content about the animal, e.g., their ethology, footprints, feeding, habitats, and features, close to its location and had a terminal and Spider's active RFID reader with a coverage range that almost corresponded to the space.

The experimental system provides each visitor or group of visitors with colored pendants including RFID tags. The counterpart is spatially attached to a pendant assigned to each visitor. These pendants are colored, e.g., green, orange, or blue. Each spot has three kinds of slots. The first can contain counterparts for visitors, the second can contain counterparts for terminals that can execute services, the third can contain counterparts for museum staffs. These slots have connections as shown in Figure 2.

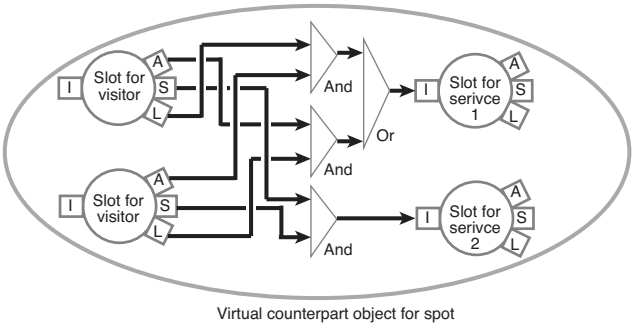


Figure 2. Connections between slots for museum guide.

For example, suppose that a visitor enters a spot with the specimen of a racoon dog and a terminal is located in the spot. His/her virtual counterpart is migrated to the slots for visitors' counterpart and service-provider objects are contained in the slots for terminals. A service-provider object is activated according to the connection for slot for service 1 in Figure 2. When another visitor enters the spot, his virtual counterpart is migrated to the slot for visitors in the counterpart corresponding to the spot. The connection executes a service for newly arriving visitors after executing services for existing visitors. The connection for slot for services in Figure 2 is to block executing services when specified conditions are satisfied even when some visitors have arrived.

5 Conclusion

We designed and implemented a framework for providing context-aware services in public spaces, e.g., museums. It supported groups of users in addition to single users and implemented application-specific services as mobile agents to deploy the services at terminals. It maintained a location model as containment relationships between digital representations, called virtual counterparts, corresponding to people, terminals, or spaces, according to their locations in the real world. When a visitor moved between exhibits in a museum, it dynamically deployed his/her service provider agents at computers close to the exhibits via virtual counterparts. When two visitors stood in front of an exhibit, service-provider agents were mutually executed or configured according to the member of visitors. To demonstrate the utility and effectiveness of the system, we constructed location/user-aware visitor-guide services and experimented with them for two weeks in a public museum.

Acknowledgments

This research is in part supported by grant from the Pro- motion program for Reducing global Environmental loaD through ICT innovation (PREDICT) of the Ministry of In- ternal Affairs and Communications in Japan.

References

[1] C. Rocchi , O. Stock , M. Zancanaro , M. Kruppa , A. Kruger: The Museum Visit: Generating Seamless Personalized Presentations on Multiple Devices, Proceedings of 9th international conference on Intelligent User Interface, pp.316-318, ACM Press, 2004.

[2] T. Kuflik, A. Albertini, P. Busetta, C. Rocchi, O.Stock, and M. Zancanaro: An Agent-Based Architecture for Museum Visitors' Guide Systems, Proceedings of Information and Communication Technologies in Tourism 2006, pp.57-60, Springer 2006.

[3] I. Satoh: A Location Model for Smart Environment, Pervasive and Mobile Computing, vol.3, no.2, pp.158-179, Elsevier, 2007.

[4] I. Satoh: Mobile Agents for Active Media Proceedings of International Conference on Software Engineering and Knowledge Engineering (SEKE 2010), pp.503-508, 2010.

Dynamic Service Choreography using Context Aware Enterprise Service Bus

Swapan Bhattacharya
Jadavpur University
Kolkata ,India

Jayeeta Chanda
B.P.Poddar Institute of Management &Technology
Kolkata, India

Sabnam Sengupta
B.P.Poddar Institute of Management &Technology
Kolkata, India

Ananya Kanjilal
B.P.Poddar Institute of Management &Technology
Kolkata, India

Abstract—**Enterprise Service Bus (ESB) is responsible for publishing and discovery of services in a global distributed delivery system. Context-aware systems offer entirely new opportunities for application developers and for end users by gathering context data and adapting systems' behavior accordingly. In this paper, we propose a Context Aware ESB (CA-ESB) that will publish and discover services based on location context. The main modules of the framework consist of Context Provider (senses location context), Context Aware Logic Module (decides which regional service to be selected based on location context) and Service Choreographer (choreographs selected services). We propose a graphical model named Context Aware Graph (CA-Graph) that will help us to dynamically choreograph the services. These modules along with other modules of SOA reference architecture will help the ESB to sense the location of users, to select the required services and dynamically choreograph those services. We define a set of metrics based on CA-graph and analyze the performance of CA-ESB. An algorithm is proposed that will dynamically choreograph the selected services based on location context. The results of the case study of an Insurance System are used to illustrate our approach.**

Keywords— Cloud Computing, Context-aware, Enterprise Service Bus, SOA based global delivery model, dynamic service choreography, CA-Graph

I. INTRODUCTION

Distributed Delivery Model[11] is a bi-directional sequence of activities consisting of requirements specification, analysis, design, development, integration, testing, and maintenance. These activities may not always be performed in a linear fashion, as there may be some overlap between and across certain processes. The distributed delivery model has gained immense importance as these days software is developed in a distributed manner with a common core component and various regional components interfacing with it.

Service-orientation requires loose coupling of services with operating systems, and other technologies that underlie applications. SOA separates functions into distinct units known as services, which developers make accessible over a network in order to allow users to combine and reuse them in the production of applications. These services and their corresponding consumers communicate with each other by passing data in a well-defined, shared format, or by coordinating an activity between two or more services.

In the distributed delivery model the services can be divided into core services and regional services. The core services will remain same for all applications and regional services will vary from one application to another application. The regional services interface with the global services to serve the overall business function requirements at each location. In this paper, we propose a Context Aware ESB (CA-ESB) that will publish and discover services based on location context. This type of ESB framework is very relevant for region specific global development scenario. A new graphical model named Context Aware Graph (Ca-Graph) is proposed that models the services/processes and their interconnections for all the locations. Discovery of services becomes much simpler using CA-ESB as the CA-Graph proposed here unveils the service choreography for each location dynamically. Analysis of CA-Graph using a metrics proposed in this paper clearly illustrates the performance of CA-ESB vis-à-vis a traditional ESB.

II. RELATED WORK

Areas related to SOA have become prominent research domains. Here we discuss about some of the relevant research works in the field of Enterprise service bus (ESB).

A dependable ESB framework that enables automated recovery from component failures is proposed in [1]. The authors in [2] propose an ESB framework to enable the content-based intelligent routing path construction and message routing. [3] presents computation-independent models (CIMs) and platform independent models (PIMs) for service oriented architectures.

Existing ESB-based system have difficulties to manage complex

events of real-world applications very well. A complex event-processing model based on the relational algebra is proposed in [4], and then it proposes a complex event processing oriented enterprise service bus. [5] presents Omnipresent, which is a service-oriented architecture for

context-aware applications that may be accessed from either mobile devices or Web browsers, and it is based on Web services page layout.

In [8], a design of a Dynamic Composition Handler on Enterprise Service Bus (ESB) is presented which analyze different types of service compositions to clarify what dynamic composition really holds in SOC.

In [9], service composition and discovery are not treated separately. Here a matching algorithm is developed that combines several services which are not known and need to be discovered. In [10], the flexible architecture of the discovery engine Glue2 is proposed which comes with a powerful set of discovery components (for functional matching, non-functional matching, data fetching, etc.) that can be executed in different order as required by specific execution workflows.

In our previous work in [7] a graph named D-SG is proposed to model design models in distributed environment. The sequence diagrams are interleaved such that the business process at a location spans several sequence diagrams modeling the common and regional use cases. This is modeled graphically using Distributed Scenario Graph (D-SG). Our work in this paper closely relates to this graph model.

In the domain of context aware software architectures, the current work proposes to encompass the design of a context aware ESB that will enable publishing and discovery of services based on location context of the users. Our approach is to incorporate context awareness to ESB unlike other existing work. Incorporating context awareness to ESB will enable the developer to control the algorithms related to context awareness centrally in the ESB(which act as a middleware) where as in the existing works in Context Aware SOA the context awareness is achieved through end devices. This kind of ESB framework will be of immense importance in global software development scenario where services can be region specific of the users.

III. SCOPE OF WORK

In this work, Context Aware Graph (Ca-Graph) is proposed that models the services/processes and their interconnections for all the locations. In this framework, the traditional ESB with the help of CA-graph and location information functions in such a manner that it gets the essence of CA-ESB that performs dynamic service choreography. An Algorithm is proposed which will perform the dynamic choreography of services based on location context. Some metrics are also proposed to analyse the performance of this CA-ESB framework.

In this work we have used some terms to describe our work. Before the description of our proposed work, we will define those terminology that are frequently used throughout this paper for better understanding of our work.

- Service Vs Process

The term *Process* is used when we describe a specific task in the Analysis Phase of SOA software development. Different processes are arranged together (choreographed) to render a specific functionality. *Service*s on the other hand are loosely coupled entities spread over a distributed system. Once the processes are choreographed for a particular application, they will be mapped with the loosely coupled services. So, processes are tightly coupled logical entities and services are loosely coupled physical entities. In this work , we will use *process* in parallel with *service* Once the processes are choreographed , the same process will be mapped with the services and that service will be either constructed or discovered in ESB.

- Regional Processes Vs Common Processes

In a global development scenario, the s/w development is done for requirements that cross regional boundaries. In an application, there will be multiple region specific processes for the same functionality. For example, GUI interface will be different for different location (language specific). Also, different regions have different business policies leading to multiple processes of the same function.

IV. PROPOSED CA-ESB FRAMEWORK

The CA-ESB (Context Aware Enterprise Service Bus) framework is shown in Figure 1.

The modules of CA-ESB are:

A. Service Consumer

They are the users of the application. In our context aware scenario, they are spread over multiple regions. Their request will be served based on the location context provided by the Context Provider.

B. Context Provider

Context Provider will provide location context of the users. The enterprise service bus will be provided with location context of the users. Based on this location context, the regional services will be chosen by the ESB. The location context can be the URL or IP address of the users. The output of this module (URL or IP address) will be forwarded to Context Aware Logic Module (CAL) for selecting region specific services.

C. Context Aware Logic Module (CAL)

Using the context data (i.e. The IP address or URL) of the user, this module will choose region specific services that are different for different users. Different regions have different business policies, different GUIs etc. So, the services will be different for different users of different location. This module will store the information regarding region specific services and select those services based on the location context of particular user. These selected

services will be used for choreography by the next module.

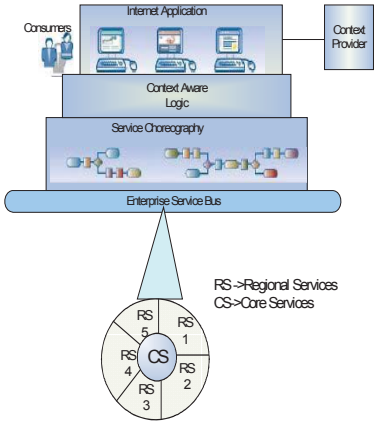


Figure 1: The CA-ESB Framework

D. *Service Choreographer*

This module choreographs the services chosen by CAL dynamically at run time. Services are regional services as well as core services. Dynamic choreography is required as the regional services are different for different applications (or users).

The fifth module is the traditional enterprise service buses that will work along with the other four modules give the flavor of CA-ESB

V. DYNAMIC SERVICE CHREOGRAPHY AND EFFICIENCY OF CA-ESB

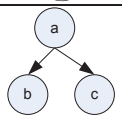
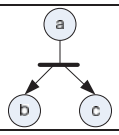
The Context aware logic (CAL) module of the CA-ESB framework stores the region specific process/ services information and selects services accordingly when location information is acquired and sent by the context provider. To model the process/service information in CAL module, we propose a Context Aware Graph and calculate the efficiency of CA-ESB compared to traditional ESB using the proposed graph. CA-Graph is used to model the processes of an application and their interconnections for different locations of users.

A. *CA-Graph: Graphical representation of processes*

We propose a graph called Context Aware Graph (CA-Graph) that will help us to categorize the processes of the application according to the location of the users. The processes represent the business processes of the SOA reference Architecture [6].

The processes will be mapped with the services in the service choreography module. The CA-Graph = (V, E) is a graph comprising of nodes/vertices and directed edges. The vertices represent processes and edges are drawn to connect the vertices based on the interconnection of the processes. Different graph construct for CA-graph is tabulated in table I.

TABLE 1: CA-GRAPH CONSTRUCTS

Grap_ Constr_ID.	Graph Construct	Meaning
1	1.L1	Naming syntax for regional process means Process 1 for location 1
2	2.C	Naming syntax for common processes which means process 2 for all location
3		Flow of events(or processes)
4		Process ‘a’
5		Exclusive-OR flow which mean from process ‘a’ the flow will move to either ‘b’ or ‘c’ depending on some condition
6		Parallel flow which means from event(process)’b’ and ‘c’ will occur in parallel after ‘a’

The key points of the graph are:

- Regional processes are labelled as <process name>. < location name > Eg. 1. L1, 2.L2 etc where L1, L2.etc stands for different location.
- Common processes will have extension C.
- Solid Arrow is used for flow of events
- Circle indicates events/ process.

B. *CA-ESB and CA-graph*

The process information in a tabular form will be stored in the Context Aware Logic (CAL) part of our proposed CA-ESB framework. This process represents a collective main process that consists of sub processes. This tabular information is named as Process Table .The Process Table has the following fields

- **Sub process name:** The name of sub processes that constitutes the main process.
- **Pre process:** This is the set of probable processes preceding the process.
- **Post process** This is the set of probable processes succeeding the process.
- **Type of Process:** This field indicates whether the processes are regional (R) or common (C).
- **Process Flow Type**: This field indicates the flow of the process with its post process (es). This can be normal flow (N), parallel flow (P) or exclusive-OR flow (E)
- **Total Processes:** This field consists of list of all possible location based processes for a particular sub process.

Table III in appendix represents a process table for the case study of insurance system that is explained in Section VI.

When the context provider provides the information regarding the location, the CAL module chooses the processes for that particular location from the process table along with common processes. These processes will be mapped with the services.
The next section briefly discusses the process of dynamic choreography.

C. *DYNAMIC SERVICE CHOREOGRAPHY*

The service choreographer does the dynamic choreography of services with the selected processes at runtime based on the process connections as modelled in CA-Graph.
As an example, if the context provider senses the location as location1 (L1), it will send the information to the CAL module. CAL module has the information regarding which sub processes are required for location L1 and Service Choreographer will do dynamic service choreography at runtime with the selected processes (or services). Once the services required for a particular location are determined, the ESB will publish only those services in the registry. This will reduce the time associated with the discovery of services.
The following algorithm will identify services based on location context from the process table and identify the scenario path composed of services in a particular order.
Algorithm:
a) Input: Location context as LocationName
b) Data Structure to be used:

- Graph construct schema table (TABLE I)
  Schema: GraphID, GrConstruct)
- Process table (TABLE III of appendix)
  Schema: ProcessId, ProcesssName, PreProcess, PostProcess, ProcessType, ProcessID, ProcessFlowType
- ProcessFlow
  Schema: Source, Destination

c) Steps to be followed:

1 .L: = GetLocationInput;
ProcessIDLoc is used to store instantaneous ProcessId locally
2. Set ProcessIDLoc: = 1;
3. LOOP
   T = GetTuple from Process Table
   where (ProcessId = ProcessIDLoc);
4. If (ProcessFlowType = 'N') then
   ProcessFlow = (Select GrConstruct when GraphId = 3 from Graph construct table)
5. Else If (ProcessFlowType = 'E') then
   ProcessFlow = (Select GrConstruct when GraphId = 5 from Graph construct table)
6. Else If (ProcessFlowType = 'P') then
   ProcessFlow = (Select GrConstruct when GraphId = 6 from Graph construct table)
7. Set ProcessFlow. Source: = T. ProcessId and
   ProcessFlow. Destination = T. PostProcess
8. If T.ProcessType == 'C'
   Then ProcessIDLoc = T. PostProcess
9. Else If T.ProcessType == 'R' then
   T = GetTuple from Process table
   where (LocationName = L AND
   ProcessID=ProcessIDLoc)
   Set
   ProcessIDLoc: = ProcessID. LocationName;
   Else continue;

END LOOP
d) Output:
Choreographed process based on the particular location context (As Figure 3)

D. *Performance Analysis*

In this section, we calculate the complexity metrics and search metrics for the CA-ESB The notations used to evaluate different performance metrics of CA-ESB is given in table II.

TABLE II: Notations

Notation	Meaning
C_{mx}	Complexity metrics of CA-Graph for traditional ESB
$C_{mx(}$(CA)	Complexity metrics of CA-Graph for CA-ESB
T_{mx}	Search metrics of CA-Graph for traditional ESB
$T_{mx(}$(CA)	Search metrics of CA-Graph for CA-ESB

- Complexity Metrics

Here we will calculate the complexity metrics for the CA-ESB. This metrics will give the measure of no. of processes and interconnections giving us an idea of the complexity of the application.
If we suppose a Global development scenario that consists of
No. of common processes=m
No. of regional processes for a particular main process=n
No of location = L
Total no. of regional process
= (No of location) x (No. of regional processes for a particular main process)
= n*L
So, total no. of processes
= (No. of common processes) x (No. of regional processes)
= m + n*L
In a CA-Graph, the processes (common as well as regional) are represented as the vertices of a graph and the communication between processes are represented as the edges of the graph.
Total no. of vertices of the CA-graph = m + n*L
Then, the maximum complexity of the graph (when all the m+n*L processes communicate with all other processes)
C_{mx}= (m + n*L) (m + n*L – 1) ---------- (1)

When CA-ESB using Context Aware Logic discovers the services, only the services pertaining to a particular location along with common services become visible in the CA-ESB for discovery. As a result the graph effectively reduces to Subgraph for a particular region having m+n processes.

The maximum complexity of the graph (when all the m+n processes communicate with all other processes)

C_{mx} (CA) = (m + n) (m + n –1) ---------- (2)

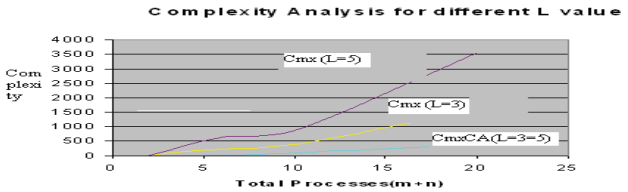


Figure 2: Comparison of CA-ESB vis-à-vis traditional ESB

Based on equations (1) and (2) we chose different values of L=1, 3 and 5 and plotted the C_{mx} and C_{mx} (CA) values corresponding to normal ESB and CA-ESB respectively. As evident from the graph in figure 2 that the C_{mx} (CA) values are lower than the C_{mx} values indication that the use of CA-ESB significantly reduces complexity.

- Search Metrics

We use a metrics to mathematically compare the time complexity involved in searching and discovering services in a CA-ESB as against a normal ESB

If CA-ESB is not used, all the processes/services will be visible, the total vertices (processes/ services) to be discovered is

T_{mx} = m + n*L---------- (3)

When CA-ESB is used, the total vertices (processes/ services) to be discovered only pertains to a particular location,

T_{mx} (CA) = m + n---------- (4)

So, Search complexity without using CA-ESB

T_{mx} = O (n*L)

Search Complexity when CA-ESB is used

T_{mx} (CA) = O (n)

From (3) and (4), the time complexity of searching of services in the ESB is reduced by L times if the CA-ESB is used.

VI. CASE STUDY

We consider an application for an Insurance company who plans to start its business operations in several countries across the globe. The application consists of different processes, which are being used by different users located in different places. Some processes are common which are being used by all users who are accessing the application irrespective of where they are located. While some processes are specific to a country. Let us consider that the application consist the following main processes

- Policy Creation
- Policy Maintenance
- Policy Claim
- Policy Termination

The flow of events (or sub processes) of the first processes "Policy Creation" is given in a tabular form as Table III of appendix. The processes are categorized as common processes (marked as 'C') and regional processes (marked as 'R'). Here, the processes are considered for two locations namely L1 and L2.

The main process ("Policy Creation") consists of 10 sub processes that are labelled according to the rule defined in section V.A. CA-Graph is generated for process "Policy Creation" as in Figure 3.

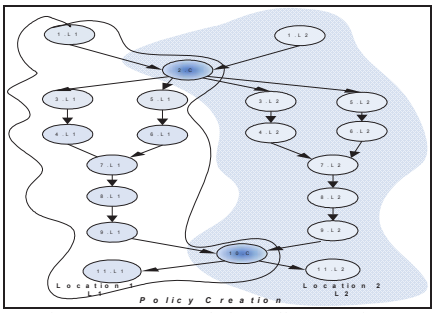


Figure 3: CA –Graph for Policy Creation

Details are given below

- It consists of 2 common processes i.e. m=2
- It consists of 9 regional processes i.e. n = 9
- The common processes are 2.C and 10.C labelled in the graph as vertices.
- The regional processes are (1.L1, 1.L2), (3.L1, 3.L2)…….(9.L1, 9.L2) and (11.L1, 11.L2)
- CA-graph is generated for location L1 and L2, where common processes are shared by both the locations.
- There is an Exclusive-OR flow from process 2 to the process 3 or 5
- The complexity metrics for the graph (m = 2, n=9 and L =2)
- Total processes = m + nL = 2 + 9x2 = 20
  C_{mx} = (m+nL) x (m+nL -1)
  = 20x19 = 380,
- Using CA-ESB,
  Total processes = m + n = 2 + 9 = 11
  C_{mx} (CA) = (m+n) x (m+n -1)
  = 11x10
  = 110 (Using equation (1) and (2))
- The search metrics for the graph (m=2, n =9 and L =2)
  T_{mx} = 20 and T_{mx} (CA) = 11
  (Using equation (3) and (4))

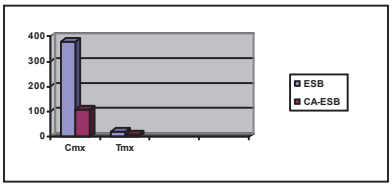


Figure 4: Performance metrics comparison for *Policy Creation*

Figure 4 shows a performance metrics chart that depicts the fact that the performance of CA-ESB is improved both in terms of complexity metrics(C_{mx}) and search metrics(T_{mx}) as compare to traditional ESB for the process *Policy Creation* which is distributed in two location(L=2). With the increase of the value of L, we will get more increase in performance of CA-ESB as compare to a traditional ESB.

VII. CONCLUSION

Presently, global delivery model is gaining significance where applications are being developed in an integrated manner for different users spread over geographically different locations forming a cloud. The core processes remain same, with several region specific processes catering to different because of variations of languages, currency, business policies, etc. In SOA architecture, ESB is responsible for publishing and discovery of all services. We propose a new variation of ESB named Context Aware ESB (CA-ESB) that will be very useful for publishing and discovery of services in a global development scenario. CA-ESB is able to sense location context, selectively discover relevant services for a region and finally dynamically choreograph them with the core services such that the whole application behaves uniquely for each different location context. We demonstrate using an algorithm and a set of metrics that the efficiency of a global software development scenario improves to a significant extent by using CA-ESB. This framework attempts to provide solution for the problem of efficient dynamic coordination of geographically distributed services.

REFERENCES

[1] Jianwei Yin, Hanwei Chen,Shuiguang Deng,and Zhaohui Wu ,A Dependable ESB Framework for Service Integration, IEEE transaction, March/April 2009 (vol. 13 no. 2), pp. 26-34, http://www2.computer.org/portal/web/csdl/doi/10.1109/MIC.2009.26

[2] Gulnoza Ziyaeva, Eunmi Choi and Dugki Min , Content Based Intelligent Routing and Message Processing in Enterprise Service Bus , International Conference on Convergence and Hybrid Information Technology 2008(ICCIT08) , Nov 11-13 ,2008 ,Busan , Korea ,

[3] Gerald Weber, Technology-Independent Modeling of Service Interaction, IEEE Proceedings of the 2008 12th Enterprise Distributed Object Computing Conference Workshops (EDOC08) , Pages 35-42 , 15-19 September 2008, Munchen, Germany

[4] Deng Bo Ding Kun Zhang Xiaoyi , A High Performance Enterprise Service Bus Platform for Complex Event Processing , IEEE 2008 Seventh International Conference on Grid and Cooperative Computing(GCC 2008),October 24–26,2008. Shenzhen, China

[5] de Almeida , D.R. de Souza Baptista , C. da Silva, E.R. Campelo , C.E.C. de Figueiredo , H.F. Lacerda ,. A context-aware system based on service-oriented architecture, 20th International Conference on Advanced Information Networking and Applications(AINA 2006), April 18 - 20 , 2006 ,Vienna, Austria.

[6] Design a SOA solution using Reference Architecture www.ibm.com/developerworks/library/ar-archtemp/ -

[7] Ananya Kanjilal, Goutam Kanjilal, Swapan Bhattacharya, "Integration of Design in Distributed Development using D-Scenario Graph", Proceedings of IEEE International Conference on Global Software Engineering, page 141-150, Bangalore, India, Aug 17-20, 2008.

[8] S.H. Chang, H. J. La, J. S. Bae, W. Y. Jeon, S. D. Kim, "Design of a Dynamic Composition Handler for ESB-based Services" Proceedings of ICEBE 2007, page 287-294 ,Hong Kong , 24-26 Oct ,2007

[9] Vaculin, R.; Neruda, R.; Sycara, K., "Towards Extending Service Discovery with Automated Composition Capabilities" Proceedings of ECOWS 2008 , page 3-12 ,Dublin, December 12 ,2008

[10] Carenini, A.; Cerizza, D.; Comerio, M.; Della Valle, E.; De Paoli, F.; Maurino, A.; Palmonari, M.; Turati, A.; "GLUE2: A Web Service Discovery Engine with Non-Functional Properties" Proceedings of ECOWS 2008 , page 21-30 ,Dublin, December 12 ,2008

[11]http://www.bestcrossmark.com/Downloads/BESTCROSS-DistributedDeliveryModel_DDM_-WhitePaper-1400.

Appendix

TABLE III: PROCESS TABLE FOR THE *POLICY CREATION*

Sub Process No	Sub Process Name for Policy Creation	Pre-Process	Post Process	Type of Process (R/C)	Process Flow Type	Total Process
1.	Application Entry	-	2	R	N	1.L1 1.L2
2.	Underwriting	1	3 or 5	C	E	2.C
3.	Underwriting 1	2	4	R	N	3.L1, 3.L2
4.	Process Error	3,6	7	R	N	4.L1 , 4.L24
5.	Accept More Details	2	6	R	N	5.L1 5.L2
6.	Underwriting 2	5	4	R	N	6.L1 6.L2
7	Process Error	3,6	7	R	N	7.L1 7.L2
8	Premium Calculation	4	8	R	N	8.L1 8.L2
9	Premium Payment	7	9	R	N	9.L1 9.L2
10	Policy Issuance	8	10	C	N	10.C
11	Certificate Generation	9	-	R	N	11.L1 11.L2

Web System to Aid Project Management

Rogéria Cristiane Gratão de Souza
rogeria@ibilce.unesp.br

Antonio Marcos Neves Esteca
amesteca@hotmail.com

Adriana Barbosa Santos
adriana@ibilce.unesp.br

Carlos Roberto Valêncio
valencio@ibilce.unesp.br

Marcelo Takeshi Honda
hondsm@gmail.com

Computer Science and Statistics Departament
UNESP – São Paulo State University
São José do Rio Preto, São Paulo, Brazil

Abstract - This work describes a new web system to aid project management that was created to correct the principal deficiencies identified in systems having a common purpose which are at present available, as well as to follow the guidelines that are proposed in the Project Management Body of Knowledge (PMBoK) and the quality characteristics described in the ISO/IEC 9126 norm. As from the adopted methodology, the system was structured to attend the real necessities of project managers and also to contribute towards obtaining quality results from the projects. The validation of the proposed solution was done with the collaboration of professionals that used the functions available in it for a period of 15 days. Results attested to the quality and adequacy of the developed system.

Keywords - project management; web-based tool; software quality

I. INTRODUCTION

Tough competition existing since the beginning of commercial activities, allied to changes provoked by technological advances, market integration and globalization have increasingly required more efforts from enterprises to find solutions which will guarantee their survival and growth [1]. Due to this, project management has become more important and an indispensible resource to increase competitive potential since it details tools and techniques that enables the organizations to increase their capacity to stimulate internal communication, plan activities, estimate and monitor time and costs, as well as to guarantee the quality of the jobs done, all of which contribute to objectives being successfully fulfilled [2, 3].

In conceptual terms, a project can be understood as being a unique process consisting of a group of coordinated and controlled activities having dates for starting and finishing, to create a product, service or exclusive result, which will satisfy customers [4, 5]. Managing projects involve complexity and celerity to make decisions. Due to this, different systems to aid project management have been developed to supply relevant information to managers in a fast, safe, and consistent manner.

Nevertheless, such systems as are available on the market have a restricted scope of the indispensable managing aspects contemplated by Project Management Body of Knowledge (PMBoK) [5], one of the most reputed guides to project managing. In this context, the objective of this paper is to present a web System to Aid Project Managing (SAPM), and to show that this system is a new solution capable of correcting deficiencies identified in systems that have the same purpose and are available in the market, follow the proposed guidelines in the PMBoK [5] and the software quality characteristics described in the ISO/IEC 9126 [6], besides complying with general requirements from project coordinators and managers from small and medium sized enterprises.

The rest of this paper is organized as follows: Section II shows a general panorama about the tools to aid project management available in the market; Section III shows the methodology adopted to develop the work; Section IV shows the principal results that were obtained; lastly, Section V shows final considerations and future works.

II. SYSTEMS TO AID PROJECT MANAGEMENT

Currently, various software systems are available on the market to supply relevant informations to project managers in a fast, safe and consistent manner [7, 8, 9]. Nevertheless, these systems are functionally restricted in relation to the real project management necessities as described in the PMBoK. For example, the system described in Ref. [7] only aids risks management area, thus making it necessary to use other tools to manage the rest of the areas. A similar situation occurs in other systems described in Ref. [8] and [9] which supply automated aid for specific contexts.

With a view to presenting an overall panorama of the scope of the software systems that aid project managing, Ref. [10] shows a comparative analysis between four desktop systems and five web systems, which were assessed as to their alignment with the PMBoK guidelines and the ISO/IEC 9126 software quality norm. The authors verified that the assessed systems had similar functions, but did not supply aid for all of the project management necessities since they all prioritize some areas in detriment to others. As to the quality criteria of the analyzed software, important limitations were noted. Therefore, it can be concluded that, although various

This research was partially supported by The State of São Paulo Research Foundation under projects number 2008/07094-2 and 2010/13478-8.

computational systems have been developed, it can be seen that there is a lack of a wider scope of resources capable of offering an effective environment for activities related to project management. Moreover, it is notorious that existing systems do not have, even indirectly, a resource that instinctively guides the user without him realizing that he is obeying the guidelines stipulated in the PMBoK.

In this scenario, there is a need to develop a new system to aid project management activities based on a quality criteria that will cover real present market necessities.

III. METHODS

The methodological process was divided into three stages:

Analyses and Projects – At this stage, the identification of requirements to be considered by the SAPM was first done. Having the requirements, architecture in layers was structured for the system, improving the organization of the information, as well as supporting incremental development of the system and to guarantee its modularity, mantainability and extensionability [1, 11, 12, 13].

In a recent work, Ref. [10] presents detailed results of this stage, with a specification of identified requirements and an illustration and description of the developed software architecture (see Appendix A).

Implementation – At this stage, an initial version of the system was constructed which reflected the contemplated requirements of the defined software architecture. The construction of the system only used free resources to guarantee an easy access to implemented resources: PHP, JavaScript and HTML languages; MySQL database management system; Apache web server.

Validation – At this stage, the system was assessed by professionals involved in project management activities, which sought to refine or confirm requirements identified initially as well as enhance the established architectural design [14, 15]. The validation was done between August and November, 2010, when the system was personally presented to nineteen project managers from fourteen Brazilian enterprises from different sectors: education, administration, consultancy, civil engineering and information technology. At the end of each presentation, the assessors received a form to grade the scope and adequacy of the system using a scale of 0 to 10 points. Moreover, there was an open space on the form so that the assessors could indicate the strong and weak points of the system. The filled in form was returned via email after the assessors had remote experienced the system for 15 days.

IV. RESULTS

The results which were obtained from the implementation and validation stages are detailed below.

A. Implementation

The objective of this stage was to obtain a usable satisfactory web system for later validation by acting project managing professionals.

The system considers the functions shown in Table I, which are organized according to the nine knowledge areas as defined in the PMBoK guide: integration, scope, time, costs, quality, human resources, communication, risks and procurements. Besides that, two complementary functions that are not directly related to any of the areas, but which were considered essential to managing activities, were incorporated to the proposed solution.

Figure 1 (a) shows the SAPM project's portfolio, from which an identified user can select an existing project to edit or register a new project. Figure 1(b) shows the screen which is exhibited by the system after a project from the portfolio has been selected, evidencing the alignment of the SAPM's principal menu to the PMBok guide, as it has the options organized according to the guide's knowledge areas, thus contributing towards the agile execution of managing activities based on solid concepts.

Besides registering and organizing information, the SAPM is capable of generating and presenting different types of diagrams and graphic indicators which aid decision making processes for project coordinators and managers. Figure 2 shows one of the diagrams generated by the system, Gantt chart, which was configured to exhibit, in different colors, activities having different status, to facilitate the understanding of the general panorama of the project. Figure 3 shows graphic indicators that refer to human resources performance in previous projects which aids coordinators and managers with their process of selecting people to make up the project teams.

B. Validation

The SAPM validation stage allowed refining the previously identified functional requirements, as well as the architecture of the project, leading the system to be adequate for the identified present necessities in the Brazilian market. Ratifying this market demand, during the validation there was a strong interest in acquiring the system with some assessors showing interest in continuing to use the SAPM in their actual projects.

The main evaluated topics in the questionaire utilized in validation were: usability - the degree of ease to access and use system resources; interface quality – the users' apreciation of the system's interface; graph quality – the appearance and representativity of the data; reports quality – the degree of users satisfaction with the type of report information generated by the system; operational efficiency – referring to degree of operational stability.

The histograms in Figure 4 reveal that the general result of the evaluation of the system by the participants was satisfactory. The vast majority of the assessments were in the range of 8 to 10 points. The left asymmetry of the data

TABLE I. FUNCTIONS CONTEMPLATED BY THE SAPM

Knowledge areas	
1. Integration	**6. Human resources**
1.1 Manage project documents	6.1 List project team members
1.2 Register events during project execution	6.2 Control personnel availability and prevent conflicting allocations
1.3 Register learnt lessons	6.3 Register and present team member performance graphs
2. Scope	**7. Communications**
2.1 Define the scope of the project	7.1 Enable online meetings between those involved in the project
2.2 Create a Work Breakdown Structure (WBS)	7.2 Production and distribution of custom made reports and warnings
2.3 Create the WBS dictionary	
3. Time	**8. Risks**
3.1 Include activities	8.1 Management risks
3.1.1 Identify activities	8.1.1 Identify risks related to the project
3.1.2 Estimate starting and ending dates of the activities	8.1.2 Estimate risk impacts
3.1.3 Allow definition of parent-activities (made up of other activities)	8.1.3 Estimate the probability of risks happening
3.2 Register activity progress updates	8.1.4 Identify events related to the probability of an increase of risks
3.3 Register finished activities	8.1.5 Monitor the probability of an increase of risks
3.4 Warn project delays	8.1.6 Create alerts about the occurrence of risks
3.5 Generate Gantt and activity network diagrams	8.1.7 Identify mitigation and contingency plans for each risk
	8.1.8 Prioritize risks based on the probability of them happening and the impact
4. Costs	**9. Procurements**
4.1 Produce cost estimates by activity	9.1 List suppliers
4.2 Create project budgets	9.2 Plan purchases
4.3 Generate project budget graphs	9.3 Register and present supplier delivery performances
5. Quality	**Remaining requirements**
5.1 Register management quality plan	Links with other software systems
5.2 Register audits to be done	Existence of access levels
5.3 Produce audit reports	

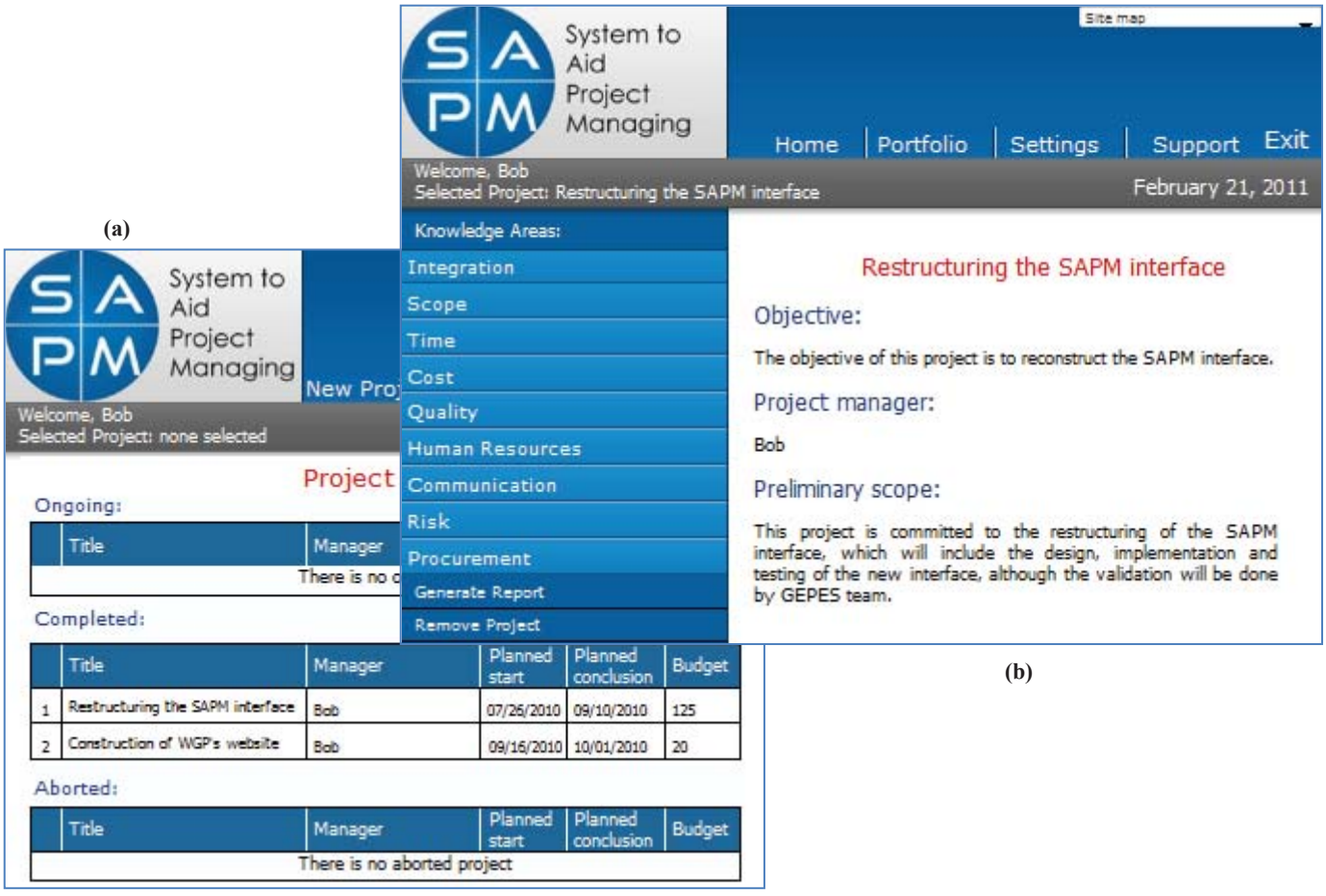


Figure 1. Project's portfolio **(a)** and main menu SAPM **(b)**.

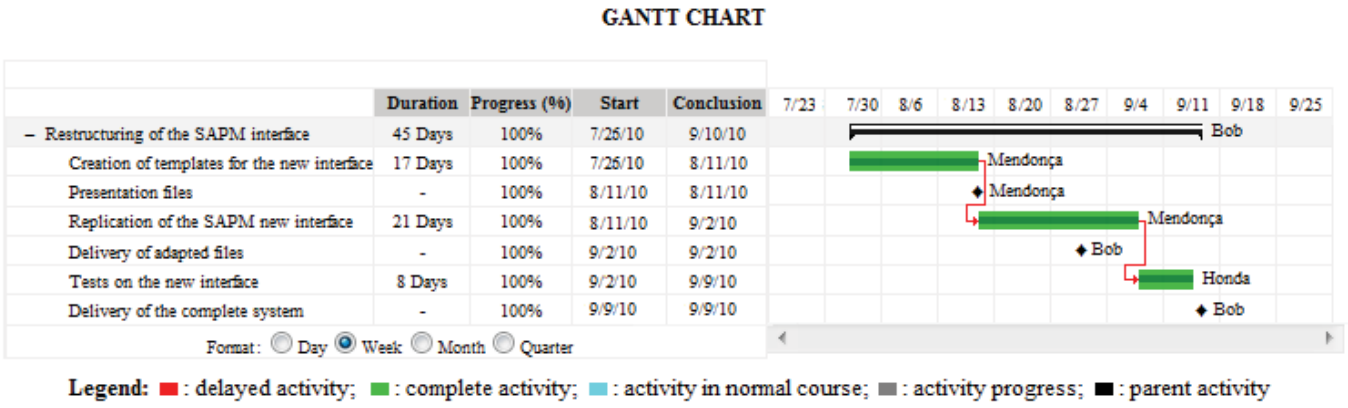


Figure 2. Gantt chart.

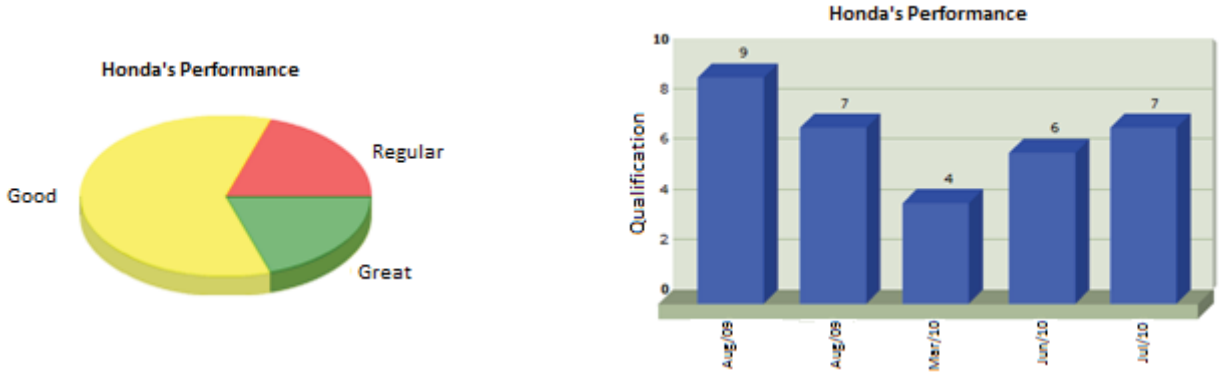


Figure 3. Human resources performance graphs.

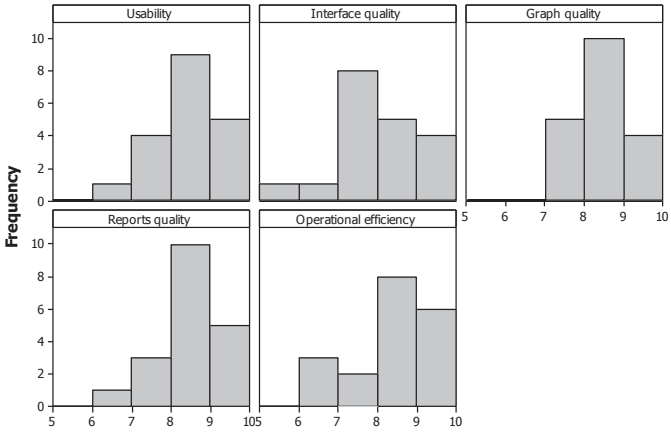


Figure 4. SAPM general assessment result.

distribution and the absence of points below 5 reinforce the high degree of satisfaction felt by the SAPM assessors.

The principal system differentials pointed out by the assessors in relation to existing options were: learnt lessons recorder; storing and organizing project documents; automatic generation of the Work Breakdown Structure (WBS); automatic detection of project activity delays; quality of generated graphs; agenda and record of auditing quality results; control of human resources and project materials availability; availability of environments for online meetings; automatic risks priority; creation of a historical of suppliers and human resources performances.

Besides mentioned differentials, the participants also contributed with some suggestions for future system improvements, among which are: create a correspondence between the chronogram and the WBS; consider holidays and vacations in the estimates; include resources that will allow estimating costs based on statistical analysis; generate graphs and diagrams to aid quality managing, such as the Pareto and Ishikawa diagrams; make available individual calendars for human resources; create mechanisms to aid quantitative risk analyses.

As a fundamental part of the methodological context, the validation exceptionally contributed towards the refinement of functional requirements for Brazilian enterprises, leading to the inclusion of new functions to the system to better serve today's necessities of small and medium sized enterprises.

To highlight SAPM's wider scope than those of other web systems on the market that have the same purpose, Table II shows an adaption of the comparative analysis presented in Ref. [10], considering only the five web systems and the SAPM. To attribute qualifications, disc representation indicating the frequency of each analyzed topic in each system was favored: greater the frequency, the larger the portion in black. This comparative analysis sought to highlight the wider scope of the SAPM as to its alignment to the PMBoK and the

TABLE II. COMPARATIVE ANALYSIS OF SUPPORT SYSTEMS FOR PROJECT MANAGING AND THE SAPM (ADAPTED FROM REF. [10])

	Criteria	Systems					
		Dot Project	ACE Project	Cooper	Project Open	Net Project	SAPM
Analysis based on PMBoK guide	1. Register of learnt lessons	◔	◔	○	○	○	●
	2. Storing documents related to the project	○	●	●	●	●	●
	3. Create WBS	○	○	○	○	●	◕
	4. Plan / control costs	◕	○	◑	◑	◕	◑
	5. Plan / control quality	○	○	○	◔	○	◕
	6. Plan / control manpower	◕	◑	◑	◕	◕	●
	7. Control changes	○	○	○	○	◕	◑
	8. Control / managing risks	○	○	○	○	◕	●
	9. Control purchases	◔	○	◔	◔	◔	●
	10. Communication between those involved	◕	●	◑	●	◑	◕
	11. Control project's progress	◑	◑	◑	◑	◑	◕
	12. Control scope	◔	◔	◔	◔	◔	●
	13. Control time	◑	◕	◕	◑	●	◕
	14. Emit reports	●	◑	◑	●	◑	◕
	15. Construct graphics	◔	◔	◔	◔	◑	◕
Analysis based on ISO/IEC 9126	1. Functionality	◑	◑	◑	◕	●	◕
	2. Reliability	◕	◑	◕	◑	◑	◕
	3. Usability	◑	◕	◕	◑	●	●
	4. Efficiency	●	◑	●	◑	●	●
	5. Maintainability	◔	○	○	○	○	●
	6. Portability	◕	◑	◑	◕	◑	◑

Legend: ○ - 0% ; ◔ - 12,5% ; ◔ - 25% ; ◔ - 37,5% ; ◑ - 50% ; ◑ - 62,5% ; ◕ - 75% ; ◕ - 87,5% ; ● - 100%

quality criteria of the ISO/IEC 9126.

V. FINAL CONSIDERATIONS AND FUTURES WORKS

This paper presented the SAPM which had its relevancy attested by market professionals who are involved with management projects in small and medium sized Brazilian enterprises. According to validation results, it may be concluded that the SAPM, due to the conceptual referential used in its realization, can increase efficiency and efficacy in the conduction of projects.

As it was projected to incorporate the guidelines proposed in the PMBoK plus the software quality characteristics described in ISO/IEC 9126, as well as to value the satisfaction of clients who are, mostly, well versed in the present scenario of web systems to aid project management, it was possible to make the SAPM capable of fulfilling the necessities of project managers and coordinators from said aforementioned enterprises in their daily activities.

Despite the limitations identified during the validation, was highlighted the contribution of the proposed system to improve the quality of project results.

Future works will seek to cover such limitations and iteratively widen this system, following the exploring evolutionary model [1, 11], which will lead the more refined and complete versions of the system.

REFERENCES

[1] R.S. Pressman, *Software Engineering: A Practitioner's Approach*, 6th ed., Philadelphia: McGraw-Hill, 2005.

[2] D.I. Cleland and L.R. Ireland, *Project Manager's Portable Handbook*, New York: McGraw-Hill, 2002.

[3] A. Murphy and A. Ledwith, "Project management tools and techniques in high-technology SMEs," *Management Research News*, v. 30, n. 2, p. 153-166, 2007.

[4] International Standard Organization. ISO 10006: *Quality management - Guidelines to quality in project management*, s.1.p. ISO, 1997.

[5] PMBoK, *A Guide to the Project Management Body of Knowledge*, 4th ed., Project Management Institute – PMI, 2008.

[6] International Standard Organization. ISO/IEC 9126: Software engineering – Product quality – Part 1: Quality model, 2001.

[7] L. M. Fontoura and R. T. Price, "Systematic Approach to Risk Management in Software Projects through Process Tailoring", in *Proceedings of the 20th International Conference on Software Engineering and Knowledge Engineering (SEKE)*, United States of America, pp.179-184, 2008.

[8] A. Tosun, A. Bener, and E. Kocaguneli "BITS: Issue Tracking and Project Management Tool in Healthcare Software Development", in *Proceedings of the 21th International Conference on Software Engineering and Knowledge Engineering (SEKE)*, United States of America, pp.526-529, 2009.

[9] R. Hewett and J. Coffey. "XProM: A Collaborative Knowledge-Based Project Management Tool," in Intelligent Problem Solving. Methodologies and Approaches, vol. 1821. R. Logananthara, G. Palm, M. Ali, Ed. Heidelberg: Springer Berlin, 2000, pp.3-40.

[10] A. M. N. Esteca, R. C. G. Souza, and A. B. Santos, "Software Architecture for Web-based Project Management System", in *Proc. Of the 16th International Conference on Distributed Multimedia Systems (DMS),* United States of America, pp. 11-16 , 2010.

[11] I. Sommerville, *Software Enginnering,* 9th ed., Essex: Addison-Wesley, 2007.

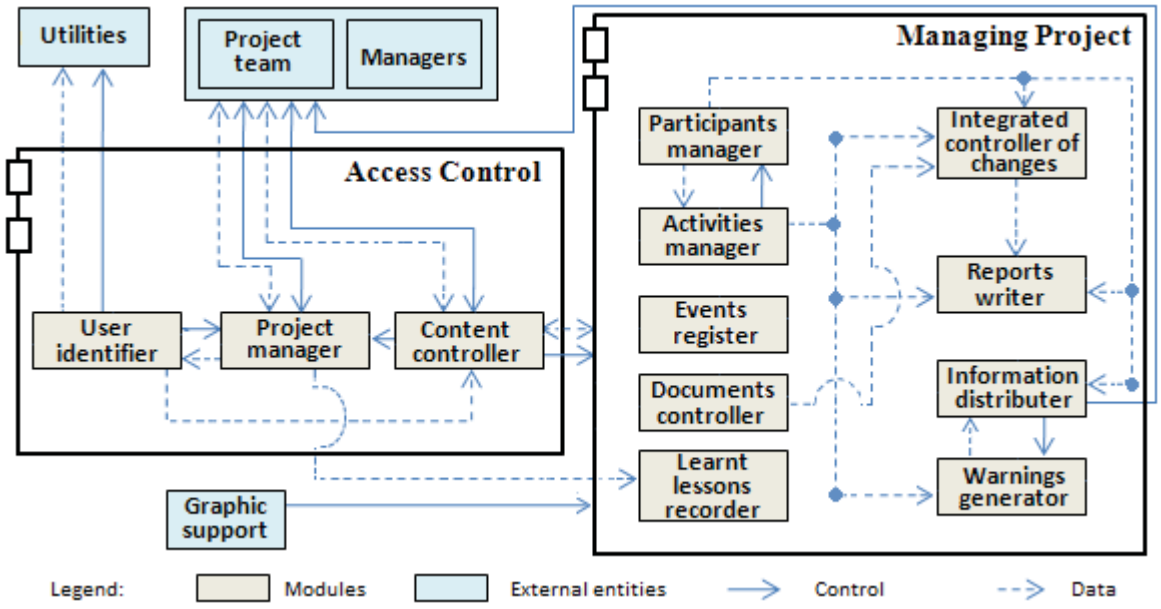
[12] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice,* 2nd ed., Boston: Addison-Wesley Professional, 2003.

[13] J. Bosch, W.M. Gentleman, C. Hofmeister, and J. Kuusela, "Software Architecture: System Design, Development and Maintenance," in *IFIP 17th World Computer Congress - TC2 Stream / 3rd IEEE/IFIP Conference on Software Architecture (WICSA3)*, Canada, 2002.

[14] V. S. Gordon and J. M. Bieman. "Rapid Prototyping: lessons learned," *IEEE Software*, v. 12, n. 1, p. 85-95, 1995.

[15] S. Raghavan, G. Zelesnik, and G. Ford. *Lecture Notes on Requirements Elicitation,* Pittsburgh: Software Engineering Institute, 1994.

APPENDIX A: SAPM ARCHITECTURAL DIAGRAM (ADAPTED FROM REF. [10])



A Composite Project Effort Estimation Approach in an Enterprise Software Development Project

Cagatay Catal
Information Technologies Institute
TUBITAK-BILGEM
Kocaeli, Turkey
cagatay.catal@bte.tubitak.gov.tr

Mehmet S. Aktas
Data to Insight Center
Pervasive Technology Institute
Indiana University, Bloomington, IN, USA
maktas@indiana.edu

***Abstract*—Software effort estimation research has been ongoing for almost 40 years. Over the years, several classes of effort estimation techniques have been introduced. Some of these techniques include model-based, expertise-based, learning-oriented, regression-based, and dynamics-based effort estimations. However, none of these techniques is best for all situations. In this study, we propose a composite technique to estimate the development efforts in a recent enterprise software development project. This paper shows how we specify and document software requirements and how we use these requirements in a novel effort estimation approach. The proposed approach is particularly effective for agile development projects.**

Keywords- Effort estimation; use case points (UCP); IFPUG; function point; cost estimation; business process management

I. INTRODUCTION

The ability to estimate a software project cost and development effort accurately is still an elusive goal, even though software engineers have been working on effort estimation models for decades. Research on software effort estimation started in the 1960s and is still a very active research area [1].

Software project managers try to control the project duration and the budget by estimating the effort and the cost at several stages of software development, such as system analysis and requirement analysis. At each stage, managers update the project plans depending on the estimation results. Furthermore, estimation can be used to approve or reject a project proposal [1]. Several purposes of effort estimation techniques are shown as follows [2]:

- Budgeting
- Tradeoff and risk analysis
- Project planning and control
- Software improvement investment analysis.

We use cost estimation and effort estimation terms interchangeably throughout the paper. Cost estimation research is not only limited to the estimation of total project cost but also to the estimation of effort necessary to perform a software process assessment and to become ISO-9001 certified for software organizations [1]. Therefore, we can state that cost estimation research has a wide application area for software engineering.

Several research findings showed that the cost and effort overruns are still beyond acceptable levels. For example, the Standish Group's Chaos Report [3] showed that an average cost overrun for software projects is around 89%. Another example, Molkken and Jorgensen [4] reported that effort overruns are lower (30-40%) than the effort overruns reported by consultancy companies (89%). However, this overrun percentage (30-40%) still drastically affects the project duration and cost. Also, they stated that expert judgment-based estimation is the most frequently used approach, because there is no empirical evidence that formal approaches are better than the expert-based techniques. There exist several significant models and techniques in literature for estimation. A good categorization for these techniques is shown as follows [2]:

- Model-based techniques
  - COCOMO
  - SEER-SEM
- Expertise-based techniques
  - Delphi technique
  - Work Breakdown Structure (WBS)
- Learning-oriented techniques
  - Case studies
  - Neural networks
- Dynamics-based techniques
  - Dynamics-based
  - System dynamics approach
- Regression-based techniques
  - Ordinary Least Squares method
  - Robust regression
- Composite techniques

Authors in Reference [2] reported that no method should be preferred over all others and discussed that a better approach may be to use several techniques. In a recent project aimed at developing a Portal, Document Management System and Business Process Management (BPM) software for Turkish Energy Markets Regulation Authority, we used a composite effort estimation technique to estimate the development effort, because different work packages had different types of modeling requirements in the agile development context. Our project had three work packages that are shown as follows:

1. *Document Management System:* This software is developed based on an open source content management system known as Alfresco.
2. *Portal Server:* This software is developed by utilizing and customizing an open source portal server known as Liferay.
3. *Business Process Management:* This software is developed with an open source business process management engine known as jBPM.

All of the software packages will work on a Linux distribution called Pardus that was developed by The Scientific and Technological Research Council of Turkey (TUBITAK). During the requirement analysis phase, we created use case diagrams and use case elaboration tables for the requirements related to Work Package 1 and Work Package 2.

In the analysis phase of the software development cycle, we spent three months identifying the business processes for Work Package 3. We used simple flowcharts to represent the workflows for each business process. In turn, this helped us to better communicate with the customer. In addition, we used sketch diagrams for graphical user interfaces (GUI) for each business process. We discussed each diagram with the customer in iterative meetings. The customer was very satisfied during the requirement analysis phase because they could easily share their ideas during the preparation of workflows and user interfaces in the analysis meetings.

For Work Package 1 and 2, we used use case modeling for the requirement analysis. Thus, we chose the Use Case Points (UCP) effort estimation approach to do cost estimation for these work packages. For Work Package 3, we used the Function Point Analysis technique (IFPUG), since we had to communicate with the customer through sketch diagrams and flow charts for each business processes. Therefore, our effort estimation approach is a composite form of UCP and IFPUG technique. In turn, this enabled a suitable software estimation effort for agile software development methodology.

In this paper, we explain how we specified and documented software requirements in this project. We show how we used these requirements in our composite project effort estimation approach. The rest of the paper is organized as follows. Section 2 presents related work and Section 3 explains our methodology for the requirement analysis. Section 4 presents our effort estimation approach and Section 5 explains our conclusions and plans for future work.

II. BACKGROUND

In this section, we explain the UCP and IPFUG methods that we utilized to estimate development efforts for our enterprise software development project. In addition, we discuss previous research conducted on composite effort estimation techniques. In [5], three stages of cost estimation are suggested for the traditional software development projects: Macro estimation (feasibility phase), detailed estimation (requirement analysis phase), and enhanced estimation (design phase). In this paper, our approach falls into detailed estimation, which takes place in the requirement analysis phase.

A. Use Case Points

Gustav Karner of Objectory developed the Use Case Points (UCP) estimation technique as his PhD thesis at the University of Linköping in 1993. Karner used several small projects while developing this technique and therefore, in-depth research that may be applied to many projects will improve the applicability of this approach. This technique is used when an object-oriented programming paradigm is chosen for implementation. Anda reported that UCP is better than expert estimation [6]. Due to space limitations, we omit the steps to calculate the development efforts.

B. IFPUG Method

Function Point (FP) measures the functionality of software systems. It was first proposed by Allan Albrecht of IBM in 1979. Function Point Analysis (FPA) evolved into the IFPUG method in 1990. The FPA is maintained by the International Function Point User Group. IFPUG is more suitable for data intensive applications but not appropriate for real-time and embedded systems. The number of inputs, outputs, enquiries, external internal files, and internal logical files in a program are taken into account to calculate the FP value. Therefore, we need to draft some graphical user interfaces for the program. Due to space limitations, we omit the details of this method.

C. Composite Techniques

Composite techniques combine two or more effort estimation methods [2]. Some researchers [7, 8, and 9] reported that the combination of effort estimation techniques and expert judgment improves the performance of estimates [10]. The Cost Estimation Benchmarking Risk Analysis (COBRA) [7] combined algorithmic and experiential approaches. According to a recent systematic review that investigated 204 papers, only 5 papers used a combination of different methods [11]. Our effort estimation methodology is different than these studies, since we used two parametric estimation models separately for each work package. Our study shows that two or more estimation techniques can be used in a single project successfully if the modeling requirements for each work package are different.

In the following sections, we give several examples to show how we documented the software requirements and combined them to estimate the total development effort. While the UCP technique was very useful for work package 1 and 2, IFPUG helped us to calculate the Business Process Management

(BPM) work package's effort. We did not prepare use cases for the BPM work package. Instead of use case modeling, we drew flowcharts and graphical user interfaces for business processes. In addition, all of the requirements were documented textually.

III. REQUIREMENT ANALYSIS

Requirements related to the work package 1 and 2 were documented by using a use case template. This template has the following sections: *Use case name, use case number, author, last modified by, creation date, last modification date, actors, summary, preconditions, post conditions, normal flow, alternative paths, notes, included use cases*. Each use case had a unique number for the traceability matrix. While 12 use cases were created for work package 1, 35 use cases were described for work package 2. However, use cases in work package 2 were much simpler than the use cases in work package 1 because they could be easily implemented by customizing the open source portal software. Use case modeling is a well-known modeling technique, and therefore we omit the details.

We preferred simple flowcharts instead of use case diagrams to document the business processes in work package 3. The customer could easily participate in the process modeling efforts because flowcharts are very simple to understand and adapt. In addition, we created sketch graphical user interfaces (GUI) for each business process. While Appendix A of the Software Requirement Specification (SRS) document includes business processes, Appendix B consists of sketch GUIs. Sketch diagrams helped our customer to visualize what the final product would look like.

Three types of business processes were identified: *Operational Processes (core business processes), Supporting Processes, and Strategic Management Processes*. 289 business processes were defined for this project. Appendix A includes 636 pages of documentation. Each requirement was documented textually by referencing the relevant diagram shown in Appendix A and user interface shown in Appendix B. A simple textual requirement for work package 3 is shown as follows:

> 3.1 *Business Process Management (BPM) software shall automate the business process BP-1 shown in Appendix A with the user interface UI-1 shown in Appendix B.*

During the requirement analysis phase, we prepared user interface prototypes; in later stages of the project, we are planning to use functional prototypes to get early feedback from the customer. Van Lamsweerde [12] emphasizes the advantages of prototyping:

> *"Prototyping as an elicitation vehicle has a built-in advantage: experimenting with some concrete flavor of what the system might look like helps us understand the implications of some requirements, clarify others, turn inadequate requirements into adequate ones, and elicit requirements that are hidden in the stakeholder's mind."*

He also expresses the limitations of prototyping and states that prototyping restricts itself to a few non-functional requirements, such as usability requirements. However, requirements such as performance, cost, interoperability, reliability and real-time constraints are ignored [12]. While there are several limitations to prototyping, prototypes are successfully used in software engineering to help the client understand some of the software features, and therefore we are using prototyping during our software development life cycle.

IV. EFFORT ESTIMATION

We applied the UCP technique to estimate the effort needed to complete for work package 1 and 2. To simplify the calculation process, we preferred the Enterprise Architect (EA) software design tool because it includes an easy-to-use UCP calculation module. Estimated values for work package 1 (Document Management System) are shown in Table I. According to these values, we need 2600 hours (or 325 days) to complete this work package. Because three developers worked on this project, we estimated that it will take 5,5 months to complete this software.

After this effort estimation was completed in June 2010, we saw that we would be able to finish this work package by the end of year 2010, in keeping with the initial plans. Estimated values for work package 2 are shown in Table II. We estimated that we need 4060 hours (or 507,5 days) to complete this work package. We will use four developers on this work package, and therefore estimated that we need 6.5 months to complete this document management system software. This estimation also showed that we will be able to finish the project on time, as we estimated at the beginning of the project proposal.

We assumed that each Use Case Point (UCP) is implemented by spending 20 hours. For the work package 3, we used the IFPUG technique because we did not prepare use cases. First of all, we calculated the Function Point (FP) value using the IFPUG technique. Later, we estimated lines of code (LOC) by using gearing factor information. The Quantitative Software Management (QSM) Company provides gearing factor information for many programming languages [13]. Because this is a Java Project, we chose value 55 for *LOC / FP* factor. A value of 55 shows that we should multiply value FP by value 55 to estimate the LOC. After LOC is estimated, we used COCOMO (Constructive Cost Model) to estimate the development effort. We have four developers in this work package.

Our effort estimation based on Basic COCOMO model showed that we need 24 persons * months to complete this work package, and therefore we need six months. The software project's complexity was chosen as semi-detached in the Basic COCOMO model. As we have seen in this effort estimation study, sometimes it may be more effective to use more than one technique in effort estimation and no one technique is best for all cases. We are planning to make an enhanced estimation after our design phase has been completed.

TABLE I. EFFORT VALUES FOR WORK PACKAGE 1

Element	Value
Estimation Date	03.06.2010
Number of Use Cases	12
Unadjusted Use Case Points (UUCP)	148.00
Technical Complexity Factor (TCF)	1.17
Environmental Complexity Factor (ECF)	0.75
Use Case Points (UUCP * TCF * ECF) = UCP	130.00
Hour per UCP (HRS)	20.00
Total Duration (HRS * UCP)	2600.00
Total Cost	78000.00

TABLE II. EFFORT VALUES FOR WORK PACKAGE 2

Element	Value
Estimation Date	03.06.2010
Number of Use Cases	33
Unadjusted Use Case Points (UUCP)	231.00
Technical Complexity Factor (TCF)	1.17
Environmental Complexity Factor (ECF)	0.75
Use Case Points (UUCP * TCF * ECF) = UCP	203.00
Hour per UCP (HRS)	20.00
Total Duration (HRS * UCP)	4060.00
Total Cost	121800.00

V. CONCLUSION AND FUTURE WORKS

This paper presented a case study of using Karner's Use Case Points (UCP) and a traditional Function Point Analysis (IFPUG) to estimate the development effort for a Turkish IT project. UCP is used for two work packages, and IFPUG is used for a third work package. We applied different estimation methods for two work packages in this project because each of them has different modeling needs. Although UCP and IFPUG seem to be competing techniques, we successfully integrated them in this enterprise software development project.

In addition, we explained how we documented software requirement specifications in project documents and effectively used these specifications in our approach. The documentation approach used in this study can be useful for IT companies that develop business process management software.

Effort estimation results are consistent with the initial predictions based on expert judgment that we've made during the project proposal. After design phase has been completed, we are planning to check the effort estimation results again. The main benefit of this approach is that we could put together use case points and function points into one effort estimation model. This approach can be used in agile development projects in an effective manner.

For future work, we are planning to apply both estimation methods to the same work package and compare the results with the actual effort observed when the work packages were implemented. In addition, we will compare the proposed approach with state-of-the-art estimation methods that were mentioned in the introduction.

REFERENCES

[1] K. Strike, K. El Emam, and N. Madhavji, "Software cost estimation with incomplete data", IEEE Trans. Softw. Eng., vol. 27, no. 10, pp. 890-908, Oct. 2001.

[2] B. Boehm, C. Abts, and S. Chulani, "Software development cost estimation approaches – A survey", Ann. Softw. Eng., vol. 10, pp. 177-205, Jan. 2000.

[3] G. Standish, The Chaos Report, The Standish Group, 1994.

[4] K. Molkken, and M. Jørgensen, "A review of surveys on software effort estimation". In Proceedings of the 2003 international Symposium on Empirical Software Engineering (September 30 - October 01, 2003). International Symposium on Empirical Software Engineering. IEEE Computer Society, Washington, DC, pp. 223, 2003.

[5] L.M. Laird, and M. C. Brennan, Software Measurement and Estimation: a Practical Approach. IEEE Computer Society, 2007.

[6] B. Anda, "Comparing effort estimates based on use case points with expert estimate", IEEE Computer Society, 2007.

[7] L. C. Briand, K. El Emam, and F. Bomarius, "COBRA: a hybrid method for software cost estimation, benchmarking, and risk assessment", In *Proceedings of the 20th international Conference on Software Engineering* (Kyoto, Japan, April 19 - 25, 1998). International Conference on Software Engineering. IEEE Computer Society, Washington, DC, pp. 390-399, 1998.

[8] M. Jørgensen, "An empirical evaluation of the MK II FPA estimation model", Norwegian Informatics Conference, Voss, Norway, pp. 7-18, 1997.

[9] I. Myrtveit, E. Stensrud, "A Controlled experiment to assess the benefits of estimating with analogy and regression models," *IEEE Transactions on Software Engineering*, vol. 25, no. 4, pp. 510-525, July/Aug. 1999

[10] M. Jorgensen, G. Kirkeboen, D. Sjoberg., B. Anda and L. Brathall., "Human judgement in effort estimation of software projects", International Conference on Software Engineering, Limerick, Ireland, 2000.

[11] M. Jorgensen, M. Shepperd, "A Systematic review of software development cost estimation studies," *IEEE Transactions on Software Engineering*, vol. 33, no. 1, pp. 33-53, Jan. 2007.

[12] A. Van Lamsweerde, Requirements Engineering,. John Wiley, 2009.

[13] QSM, http://www.qsm.com/?q=resources/function-point-languages-table/index.html

Project Risk Management Using Event Calculus

Andreas Gregoriades, Vicky Papadopoulou Lesta, Petros Petrides
Dept. of Computer Science and Engineering,
European University Cyprus.
6, Diogenous Str., Engomi, P.O. Box: 22006, 1516 Nicosia-Cyprus
{a.gregoriades, v.papadopoulou}@euc.ac.cy,
petros1@eclatent.com

Abstract - Risks are unavoidable in systems engineering projects due to emerging changes during the lifetime of projects. Changes in activities and peoples' roles are usually not free from conflicts, which in some cases if not dealt adequately, increase the project failure risks and could bring the whole project to a standstill. Herein, we present a method that examines the impact of change to project's duration which constitutes one of the most critical risks in project management. The method proposed utilizes two main criteria namely, the degree of dependency among activities and actors, and the temporal costs associated with the change. The proposed methodology is realised in Event Calculus (EC) and elaborated with an example.

Keywords: Event Calculus, Project Risk Managment, Conflict Impact Analysis.

I. INTRODUCTION

Despite sophisticated tool support for project management, projects still continue to suffer by budget overruns and a failure to meet user requirements. Various approaches have been proposed as solutions. An example is the capability maturity model integration (CMMI) [6] that defines layers of control to help ensure quality and efficient procedures that yield an improved project development process. In particular, CMMI's change control, limits project's scope creep, while, the management review, enables intermediate quality checks on the system. These have shown to be effective in improving project performance. One indicator of project quality is process flexibility, that allows rapid and cost-effective modifications of new needs [7]. However, achieving a high level of project flexibility has often been slow, inflexible, and time-consuming, even with the aid of sophisticated tools and methodologies, which oppose the goal of efficient production. Therefore, it is necessary to examine higher levels of project development flexibility by evaluating risks while still meeting budgetary and temporal restrictions. These include, role management and requirements change during a typical system development life cycle (SDLC). Requirements change analysis is translated into change impact analysis that investigates the effect of change to dependent requirements, activities and people involved. However, in most of the times changes to requirements or roles during the SDLC are not free from conflicts. If these are not resolved they could lead to inconsistencies in the process that could result in product failure. Conflict analysis falls under the change management process that is composed of three conceptual phases, namely: dependencies specification, traceability methods, conflict analysis and resolution. The method proposed herein, demonstrates the application of EC to identify conflicts in project given specifications of project's schedule, tasks and modifications. Resolving conflicts is of critical importance since they can bring a project to a standstill. The proposed method can monitor the task completion of each team member, and check if the project schedule falls within agreed time limits. Changes to requirements are also considered to assess the extra time introduced.

A. Problem Definition

Project risk management encompasses knowledge, techniques, and tools necessary to manage the risks that emerge during projects' life cycle. It is a methodological approach to minimizing uncertainty that could jeopardize achieving agreed output upon specified time frame with defined resources. An important phase during project risk management is the definition of activities. These may be related to each other via dependencies, i.e., action A_i must be implemented before A_j is completed. Given these interdependencies, risks emerge with the introduction of change to project plan that could lead to conflicts. The potential of conflicts in systems engineering projects is usually high due to the involvement of individuals from different backgrounds, specializations and locations, working together to achieve complex tasks. The objective of risk management is to provide the system designers/managers with early detection of risky situations as these are manifested by conflicts. Output from this process aids system designers in adopting appropriate countermeasures to resolve or at least mitigate emergent risks.

B. Contribution

During the development of a project, it is critical that some tasks are completed at specific times. However, actions or inactions from project members could cause critical delays. Hence, this could lead to failure to complete critical tasks that lay on the critical path. Therefore, changes in initial team structure or project specification could cause extensive overheads to the project's schedule due to conflicts. These changes could create risks of the following types: critical requirements, people and estimation risks.

Herein, we propose a Project Risk Management tool that evaluates the effect of project changes and in essence provides valuable feedback regarding the project's estimation risk. During project execution, activities of each member are recorded in the tool that detects possible conflicts that emerge due to changes in team structure. The tool resolves conflicts by suggesting appropriate modifications in the project implementation plan to improve the likelihood for guarantying success. The tool is realized using EC. In particular, using appropriate event specifications and axioms, the tool identifies conflicts upon changes to project properties. Information that is used during the analysis includes: events relating to functional project requirements, temporal specification of project characteristics i.e. start, end date of the project, roles, capabilities and actions of project members and modifications to project specifications. Inference rules are used to identify

conflicts caused by changes or actions that do not adhere to the project plan. Conflicts resolutions are provided when feasible.

C. Related Work

Work by Giorgini et al [5] describes a conflict detection method, composed of three phases namely: modeling, extensional description and verification. During the modeling phase the designer draws the extensional requirements models where every node and edge corresponds to a fact in the formal framework. Then, the reasoning system completes the extensional description of the system using rules and verifies its consistency using constraints. If any inconsistency is detected by the reasoning system, the designer revises the requirements model to avoid or at least mitigate detected conflicts and repeats the formal analysis step. This method uses the Datalog solver which is based on Boolean algebra to detect conflicts. In contrast to this approach the methods described herein is not indented to work solidly on requirements analysis, but also during project implementation where changes are more expensive to be realized and alterations to project schedule more important. EC was also utilized in [5] for policy and specification and analysis. The proposed method transforms policy and system behavior into formal notation which is based on EC. Work by [8] and [4] also use EC as a specialised first-order logic for formalising policy specification and accordingly identify conflicts and propose resolutions in network and systems management. Chomicki [9], similarly use constraints which are policies that prevent a specified action from being performed in a given situation.

II. BACKGROUND

A. Event Calculus

EC allows specification of system behaviour using notations, such as, state charts, which can then be automatically translated into a logic program representation. EC supports deductive, inductive and abductive reasoning. Abductive reasoning proof for EC can be used to detect the existence of potential conflicts in partial specifications and generate explanations for the conditions under which such conflicts may arise [4]. EC is a "logical language" from which actions and their effects can be represented using predicates. Therefore, in system engineering projects, actions that took place at particular time using certain resources to produce specific output, can be defined, explained and proved using EC. Consider the following example: Maria was hired as an account in a company on May 11 2001, Maria was promoted to head accountant on May 11 2002, Maria left her position on 23 April 2004. Based on these statements we can make the following assumptions:

A) Maria was an accountant after May 11 2001.

B) Maria ended his accountancy post on May 11 2002,

C) Maria was a chief accountant from May 11 2002 until 23 April 2004.

If the events however were mentioned in the following manner: "Maria is hired as an accountant in a company on May 11 2001, Maria left the company as a chief accountant on 23 April 2004" then the assumptions cannot hold; There is a timing gap between the ranking of the person, therefore the promotion cannot be assumed, as other events could have happened such as maternity leave. Solution to this problem is divided into two phases: the theorem problem solving, which uses theorems that have been proven in order to give the solution and the problem specification and execution part, which uses logic to infere solutions. In logic programming the implications are treated as a base for goal reduction procedures.

B. Logic Programming & Prolog

Logic programming uses mathematical propositions expressed in computational terms. Prolog is a logic-based programming language developed by Alain Colmerauer, that use the following constructs: an *atom* that describes a general-purpose meaning, a number which can be floating or integer, a variable denoted by a string consisting of letters, finally, compound terms composed of "factor" which is an atom and the arguments. Prolog works by making *deductions* and *derivations* from facts and rules stored in a *database*. The essence of Prolog programming is writing crisp, compact rules. The deductions and derivations, instigated by user-entered queries, are products of Prolog's built-in inference mechanism called *backtracking* which is part of abductive reasoning.

III. METHODOLOGY

The Project Risk management tool is built in EC terms using events and rules. Events related to project implementation, such as the beginning date of the project, actions of members and changes to project specifications are specified using EC constructs. Inference rules are used to backtrack the queries to identify conflicts caused by changes in project plan or actions that do not satisfy project constraints. Conflict resolutions strategies are presented when feasible.

A. Describing Events and Relationships with EC

Consider the following scenario: Two teams are working on systems engineering project involving the development of a car simulation project, which includes both hardware and software implementation. The project requires parallel development of both software and hardware. This procedure has the advantage of allowing various parts of the project to be progressed simultaneously. Testing can be done after completion of each phase. Team one consist of Martha, Tim, and Thomas. Martha has similar expertise acquired from similar training with Tim and can replace him if necessary. Tim is responsible for the gas system, Martha for the pipeline system and Thomas for the steering system. Give these descriptions, the following events hold:

- E1 is an event at which Tim is trained for the gas system. E1 has time 10 March 2004.
- E2 is an event at which Martha is trained for the pipeline system. E2 has time 5 March 2004.

Events are recorded in a database. Note that updates of the database are additive; events are added in the database and no deletions are allowed. In order to delete an event that expresses a relationship, a new statement needs to be set that ends the relationship. Temporal events are treated without any particular order, hence, older events can be added after adding recent events. The chronological order of the events can be expressed

using the <, >, = symbols. For instance, if an event E1 occurred before an event E2 then it can be described as E1<E2. Events can also have duration, a starting point and an ending point.

1) Representation

Given the systems engineering example, an event concerning the ranking of a person in the project, is expressed in EC as follows:

X has rank Y for a period E, provided that X has done something at a period that is equal to E.

This means that after an event happened, variable X has rank Y for a period E, which is actually equal to the time following E. In this example let's assume that Tim started working on the project on the 21st of April 2004. Therefore E1 is an event with timing: 21 of April 2004, at which X has rank Y (Tim is working on project) for the period after E, that is after the 21st of April. The events E1, E2 are in the past and their time period will finish when the person is assigned on a different rank. Events are expressed in EC using the following form: (*i*) Time (Event, Time), to express that event Event happened at time Time. (*ii*) Trained(Martha, pipeline system), to express the person Person was trained for activity A and (*iii*) Rank (Person, Time, trained A), to express that the person Person at time Time has been trained for activity A. So, for example, events E1, E2 can be expressed as follows: Trained(Tim, gas system). The *Hold* predicate is used in order to express the time periods of the relationships. The *Holds(p)* predicate states that a relationship associated with P holds for a time period p. Likewise the statement,

Rank(Tim, working on gas before (E2))

can now be written as

Holds(before (E2 rank (Tim working on gas))

which means that "Tim holds the rank of working on gas for a time period which is before E2, or in other words which last before E2 starts."

Additionally, the following predicates state:

- Initiates: represent the start of an event.
- Terminates: represent the termination of an event.
- Broken: represent identical or incompatible relationships.

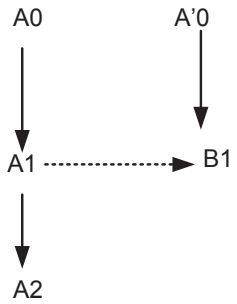
Therefore, Holds(before(e u)) IF Terminates(e u), implies that "An event E holds before time period u, if the event E is terminated at time period u".

IV. Example Application of EC in a Systems Engineering project

Application of the approach is demonstrated with an example from the systems engineering domain. In particular, we elaborate on the approach through an example drawn from the development of a vehicle subsystem. The process commences with the population of the database with the temporal project activities relating to the functional system requirements and their dependencies. Next, a second database of *actors specializations* is created. Finally, the "*event planner*" database is specified that holds the tasks of each actors as scheduled by the project plan. New events can be added in the database during project execution and while monitoring the project's progress. During project execution changes can be instantiated and subsequently evaluated using queries relating to activities or roles in the project schedule. The tool provides the user with the impact and feasibility of each change.

A. Project Specification

Consider the following model of a project specification depicting *activities* A0, A1, A'0, B1 and their dependencies. The diagram is read as follows: Activity A2 should be implemented before A1 and activity A1 should be implemented before A0, and B1 should be implemented before A'0. The dashed line means that Activity A1 can be replaced by activity B1.



B. Actors

Participants in the project, called *actors* are involved in the development of the project by undertaking activities, provided they are trained prior to engaging with the activity. Therefore, the training of actors P1 and P2 could be set as follows:

- Actor P1 is trained for the implementation of A0, A1, A'0, B1 and A2.
- Actor P2 is trained for the implementation of A0, A1, A'0 and B1.

C. Events

The following events E0-E3, that took place during the development of the project are recorded in the database:

- E0 is the event that defines the project start date. Date is 10 Feb 2010.
- E1 is the event stating that actor P1 completes activity A0. Date is 11 Feb 2010.
- E2 is the event stating that actor P1 completes activity A1. Date is 16 Feb 2010.
- E3 is the event stating that actor P1 leaves from project for 4 days. Date is Feb 21 2010.

Training of each actor is recorded in the database in the following manner:

- Rank(P1,(before E0),trained A0)) ^ Rank(P1,(before E0),trained A1)) ^Rank(P1,(before E0),trained A'0)) ^ Rank(P1,(before E0),trained B1)) ^ Rank(P1,(before E0),trained A2)).

which is translated to:

"P1 will be trained for activities, A0,A'0,A1,B1,A2 at time period before E0".

Also, we record:

- Rank(P2,(before E0),trained A0)) ^ Rank(P2,(before E0),trained A1)) ^
- Rank(P2,(before E0),trained A'0)) ^ Rank(P2,(before E0),trained B1)) .

which is translated to:

"P2 will be trained for activities, A0,A'0,A1,B1 at time period before E0".

D. Conflicts Identifications and Resolution

After recording all events related to the project, we can make queries concerning the temporal feasibility of the project. In particular, we can make queries of the following form:

- Rank_of (P,Y,T): to get the rank of actor P at time T. The system will report the actions Y performed at that date.

Moreover, to find the rank of actor P, on the 16th of Feb 2011 we make the following query: "Rank_of((P1,Y,(16 Feb 2010))".The system will search the database for the rank of actor P1 and will return the actions Y based on which the rank of P1 on the specified date is satisfied. Hence the output would be: "implements activity A0". When a project manager is interested to change an activity in the project plan, the following query could be used to get the dependencies of an action A:

- Activities_of(A, Y)

Similarly, in order to inquire about the training of actor P, on activity Y, for time T we can use the following statement:

- Rank(P,T,trained Y):.

Therefore, to check whether a change in activity A'0 and B1 is free from conflicts, the following queries are made:

- Rank(P1,(before E0),trained A'0)),
- Rank(P1,(before E0),trained B1))

In this case the system responds with a "TRUE" clause which stipulates that we can safely assume the change of P1 without any conflict.

E. Examples of Queries

1) Example1

At time period E0, if there is a request for a change in activities A1 to B1, then this will result in no conflict: This however is the best case scenario since the project has just began and none of the activities have been implemented. In this case , the following predicate holds:

Holds (before E0 (P1 implements A0)) FALSE

Which means that the predicate "actor P1 has implemented activity A0 before event E0" does not hold.

2) Example 2

Similarly, if there is request for a change to project activities, on the 21st of Feb, E4, then this will result in a conflict, since no actor would be available to take over the change. Hence,

- Initiates (after E3 and before E4(P1 implements A0)) FALSE
- Initiates (after E3 and before E4(P2 implements A0)) FALSE

To resolve this conflict: Actor P2 can replace actor P1 and implement the required parts. A conflict however, may occur, when the activity A2 needs to be implemented at a time prior to actor's P1 completion of training :Rank(P2,(before E0),trained A2)) FALSE. This means that "Actor P2 is NOT trained for activity A2".

V. CONCLUSIONS AND FUTURE WORK

The work presented herein addresses an important problem in project risk management, namely, the analysis of conflicts and their resolution. The method described elaborates on the pressing need for timely delivery of projects within agreed time limits. The complexity and uncertainty of modern systems however makes this a challenging task. The statistics of project failures alone highlights the complexity of the problem. The method described provides the mechanism for an effective mitigation of project overrun risk that emerge with changes in project requirements, activities and roles using EC. This enables automated inference of project's state given instantiations of events that define changes to schedule, requirements or structure of the project plan Preliminary results from this work are encouraging. However, the full extent of the method will be realized with its thorough validation. On the same vein, project cost is also another burden to project managers that effectively could bring projects to standstill if not managed adequately.

VI. REFERENCES

[1] R. Kowalski., "Legislation as Logic Programs", Springer-Verlag, 1992.

[2] D. Kowalski, Kuehner, "Linear Resolution with Selection Function, Springer-Verlag, 1983.

[3] R. Kowalski, "Predicate Logic as Programming Language", EEE Computer Society Press, 1986.

[4] A. Bandara,E. Lupu, E. Lupu and A. Russo, "Using Event Calculus to Formalise Policy Specification and Analysis", Policies for Distributed Systems and Networks, 2003.

[5] P. Giorgini, F. Massacci, J. Mylopoulos, N. Zannone, "Detecting Conflicts of Interest", Proceedings of the 14th IEEE International Requirements Engineering Conference, 2006.

[6] D. Ahern, A. Clouse and R. Turner, CMMI distilled, Addison-Wesley, 2003.

[7] B. Hugher, Cottetell M. "Software Project Management", McGrahill, 2011.

[8] M.Charalambides, P. Flegkas, G. Pavlou, A. Bandara, E. Lupu, A. Russo, N. Dulay, M. Sloman, J. Rubio-Loyola. "Policy Conflict Analysis for Quality of Service Management", 6th IEEE International Workshop on Policies for Distributed Systems and Networks, pp. 99-108, 2005.

[9] Jan Chomicki, Jorge Lobo, Shamim A. Naqvi, "A Logic Programming Approach to Conflict Resolution in Policy Management", Proceedings of the 17th International Conference Principles of Knowledge Representation and Reasoning, pp 121-132, 2000.

The Impact of Software Development Team Dynamics on the Knowledge Management Process

Shuib Basri
Computer and Information Science Department
Universiti Teknologi PETRONAS
Tronoh, Perak, Malaysia
shuib_basri@petronas.com.my

Rory V. O'Connor
Lero, The Irish Software Engineering Research Centre
Dublin City University
Dublin, Ireland
roconnor@computing.dcu.ie

***Abstract*—The influence of software team dynamics on well-organized software development knowledge process could prevent software development organizations from suffering from the knowledge atrophy problem. To explore this, we have studied several team dynamics factors that influence the Knowledge Management Processes (KMP) in Very Small Entities (VSEs) [1]. A survey was conducted in a variety of VSEs and through statistical and qualitative content analysis for the research data, results indicate that small teams, informal team process and structure have an important influence on the level of team dynamics in the software development process**

Keywords-component; software process, knowledge

I. INTRODUCTION *(HEADING 1)*

Software development is a complex activity and depends strongly on human commitment for its implementation. Furthermore since software development projects involve knowledge intensive exchanges and collaborations, the influence of team dynamics on the organization of software development knowledge could assist software companies to become more innovative and efficient. Hence KMP is more effective in an organization if the development teams have a good team culture with ability to share knowledge, collaborative relationship and personal responsible in creating and sharing knowledge [2]. In addition KMP is also reshaped by the attitudes and behaviour of team in order to ensure that both personal and organizational knowledge are always available [3]. The issues of limited resources; especially in cost and people almost always become an issue and can have an impact on the KMP in VSEs [4]. Therefore it is our belief that better understanding the influence of team dynamics in software projects could assist small companies to mitigate VSEs KMP against the knowledge atrophy problem.

II. BACKGROUND

A. Very Small Entities (VSEs)

The definition of "*Small*" and "*Very Small*" companies is challengingly ambiguous, as there is no commonly accepted definition of the terms. In Europe, for instance, 85% of the Information Technology (IT) sector's companies have 1-10 employees. In the context of indigenous Irish software firms 1.9% (10 companies), out of a total of 630 employed more than 100 people whilst 61% of the total employed 10 or fewer, with the average size of indigenous Irish software firms being about 16 employees [5]. The term "*Very Small Entity*" (VSE) had been defined by the ISO/IEC JTC1/SC7 Working Group 24 "*an entity (enterprise, organization, department or project) having up to 25 people*" [6]. Furthermore the issues of limited resources in VSEs always become a constraint in producing a competitive product in today's dynamic software business. [7] states that micro enterprise including VSEs whose have limited resources, particularly in financial and human resources, are practicing unique processes in managing their business. These unique characteristics have influenced VSEs in their business style and companies' process infrastructures compare to large companies' [7]. In addition due to the small number of peoples involved company's activities, most of the management processes are performed through an informal way and less documented.

B. Teams and Knowledge Management

According to [8] software development is a combination of two basic processes; social process and technological process. [9] argues that software production is more effected by social process rather than technological process. People are not only claimed as the greatest asset in a software organization [10] but also critical to software development success [11]. Software is always developed in a group rather on the individual basis [8] and the basis of every software project is a team [11]. [12] argue that the dynamic performance software project which involved many processes is always depends on team especially in quality of communication within team and between teams. They added that the communication can be applied in many ways not only in verbal but also in term of documentation form such as version control, guidelines, reports and many more. Moreover the communication also has a related impact with the team proximity [7]. They add that the increase distance from one team to another could effected the team dynamics in which it will interrupt team communication, coordination, mutual support, effort and cohesion [13]. Therefore in order to be success in KMP, organization must have a solid support from the software development and management team. The development and management team must be able to work together, share the knowledge and able to communication one another effectively. This is because the essence of software development is good relationship, effective communication and

high esteem of teamwork among software development and management team.

C. Teams dynamics

Team dynamics effect how team reacts, behaves or performs and the effects of team dynamics are often very complex [15]. There are various forces could influence team dynamics including nature of the task, the organizational context and team composition. In her dissertation [14] on dynamics of successful software team identified four characteristics of team dynamics; positive, negative, internal and external team dynamics. Positive team dynamics is the positive forces that can lead a team be a high performing successful team. [16] states the present of social relationship in a team could increase team productivity and could enhance social and interpersonal skill [17]. [18] argues that social interaction skill dimension can divide a team member to extrovert or introvert. Extroverts' team member is a people oriented, sociable person, who enjoys interaction with others. Meanwhile introvert person is a type of person who like to work alone and with less social interaction. Meanwhile, [19] believes that the positive mode of leadership (such as well focus directive, well plan and others) in software organization could enhance the positive team dynamics. Negative team dynamics is a negative force that could lead the decrease of team performance and preventing people from contributes with their full potential [14]. According to [10], from management point of view, in software development organization people are required three types of needs that have to be fulfilled and satisfied; social, self-esteem and self-realization needs. Social needs are related to social interaction and communication. The lack or ignorance of these needs will give a negative impact on the organization because people may feel unsecured, have low job satisfaction and decrease their motivation [20]. These will stop them from giving full commitment and cooperate in their work as a team member. Internal team dynamics are referring to the forces that exist within the team itself [14]. Team member also will not cooperate if they do not feel that that are a part of the team [21]. While internal social interaction between people could build team cohesion that will enhance team performance. [29]. External team dynamics are referring to the present of external forces that beyond the team control and could impact the team performance [14]. According to [23] the intrinsic and extrinsic factors in projects may motivate team. Intrinsic factors are the internal factors that consist in the task and team activity itself. Extrinsic factors are external factors that influence team from the outside such as reward and recognition, feedback from the organization and customer, team member pressure and the working environments. Moreover a better working environment also could enhance job satisfaction among team member [24].

III. Study Methodology

For this study we have developed and distributed a survey questionnaire to software VSEs (involved in software product development) in Dublin, Ireland. The survey questionnaires (which followed a GQM approach) were consisted of quantitative and qualitative questions. In order to get a quick replied, we regularly contacted the respondents via email and phone. Each received and completed questionnaire were compiled and analysis. The close-ended questionnaire were grouped according the issue and analyze using a statistical analysis. Meanwhile, on the open ended data, we analyze and categories the data according to the category that this study intends to understand. In summary we adopted the qualitative contents analysis approach in analyzing the open-ended answer [23]. At the end, we have merged the both analysis result in order to gain more understanding and validate the results. We have received a total of 70 filled questionnaires and have conducted 15 interviews for this study, In order to produce details analysis results, we have divided the survey respondents into 2 main group namely the Micro VSE (M) (1-9 employees) and Larger VSE (L) (10-25 employees) [1].

IV. Findings and Discussion

A. Teams dynamics and Structure

In this section, we explore the respondents' opinions on the companies' software development team status and study people working relationship and team environment in the companies.

TABLE I. Team Dynamics

Grp		Clear Roles	Appropriate Size	Diverse Skill Range
M	Mean	3.60	3.20	3.60
L	Mean	3.60	3.40	4.00
Avg	Mean	3.60	3.30	3.80

Table 1 indicates that the respondents' strongly agree that the development teams in their companies have a high level of team dynamics. The results shows that the team have a great working and social relationships, willing to share opinion and idea, having a good interpersonal skill and working closely each other. Further, other results (data not shown here due to lack of space) show that even though VSEs having a small team and a flat structure but staff are clear about their roles, they have enough manpower and skill to do all the development tasks. Meanwhile from the qualitative analysis, indicated that all respondents claimed that their development teams are efficient and effective. They claimed that their development team are having all important criteria such as high skills, motivated, dynamic, socialize and good teamwork, open communication, able to meet project deadline and budget, active in sharing and involved in strategic planning. These points are illustrated in the following extracts from interviews: "*They get on well as a social group and communicate regularly and openly. Also the projects we manage are normally 1 to 2 man projects and hence easily manage in an ad-hoc manner by two people that get on and communicate well." 2) "We practice clear communication and we are active in informal knowledge sharing. Beside that our environment is a family culture and, following specific strategic planning... We also actively use communication tools.*"

Beside that the result on employee turnover rate question has strengthen the above finding regarding team environment in the VSEs. The result in this question shows that the companies do not have any serious problem with the staff turnover. They claimed that the company environment, management and working styles and team relationships that

satisfied the employees have motivated people to stay longer in company. The following interview quotations which best explain the details of this situation: *"We handle many varying projects of different sizes and complexities and have a very loose/informal and friendly atmosphere. This means the work is challenging and rarely gets boring while it also being enjoyable here." "We have 14 employees. Last one who resigned in was 3 years ago. The reason people stay is we operate in relaxed and informal environment.".*

In overall team environment issue give an indicator that all the above parts or processes are much related and depended to the organization team environment, process and culture in the organization

B. Communication

The results from the analysis as shown in table 2 indicate that the companies are practicing regular informal meetings (e.g stand-up meeting, online meeting) and practicing informal formal communication in their business operations. However the results also show that organizations have clear communication process and channel. Moreover the results also indicated that that employee size has influence the formal communication process level in their VSEs daily business operations. This has been shown in comparison results between the L-VSEs and M –VSEs for this issue.

TABLE II. COMMUNICATION PROCESS

Grp	Staff Knowledge	Project Exp. & Lesson Learned	Experience Doc	Progress & Procedure
M	2.20	2.20	2.20	2.20
L	2.80	3.20	2.80	2.60
Avg	2.50	2.70	2.50	2.40

In relation to the communication process in VSEs, the analysis on the open-ended question indicated that 90% of respondents are agreed that in development projects they regularly receive feedback from the project stakeholders. However the result showed that this process been done either in face to face, informal discussion, online communication, informal internal feedback or 'on the job training' process. The interview extracts following illustrates how the process has happened: *"Online communication, informal feedback, internal discussion, and informal communication" "We sit in one office so I talk to them all the time"*

C. Learning and Sharing

All respondents' are agreed that their development team sharing and learning activities are active in the organization. This was shown from the research result which obtained more than 3.00 point in mean. This represents an indicator that in VSEs companies, they always utilize the knowledge and experience within the organization in performing their tasks. This analysis also found out that there are no big differences in term of company size in utilizing existing knowledge and experience in company.

Related to above data the open-ended question indicated that the learning and sharing activities in VSEs are being done either informal self-learning or informal knowledge sharing among the development team. This has shown how the employees enhance their skills which resulted in 90% of the respondents agreed that no formal training were given to the staff in enhancing their skills. The interview extracts below reflect the above points: *"Informally through ad-hoc conversations and some code review", "Ensuring that no single member of staff has any exclusive knowledge by using a mentoring/buddy system."*

D. Documentation Process

Our data indicates that the documentation process has been done in informal process. In details it showed that people's knowledge, experience and activities are not documented properly or have been done personally. This was showed on the total mean score which presents that all respondents do not practice a formal documentation process in their documentation activities. Our data also indicates that number of employees working in the companies give an influence to the documentation formality process in VSEs.

In relation, the qualitative answers have highlighted that only business procedure and technical issues are being documented properly and organized. This could be identified in question on documentation process where 50% of the respondents claimed they felt that they are regularly update their document regularly especially on a specific works and procedures. Moreover the analysis results also showed that small team size issue is an obstacle to VSEs from performs seriously documenting their activities as shown by below interview extracts: *"We documented it electronically, and having an equal decision on it". "We are too small to do proper documentation process"*

The result in this part of analysis demonstrates a pattern and indication that in VSEs documentations process are done in two ways; (1) the specific documentation process which is related to business and technical process and (2) informal documentation process which are inclined toward informal, personal and online documentation.

E. KM Process and Commitment

The questions on this part emphasize particularly on KM process and commitment in the software development projects. The results from the analysis as shown indicate that the respondents were agreed that the level of KM process and commitment in VSEs are very significant. This could be identified with the average mean score for each question is relatively high. Our data indicates that in principle respondents are agreed they are having a clear KM strategy and a good leadership in their organization is important in organization software development knowledge as reflected in the mean score results for these two questions. However our results indicate that activities related to KM within VSEs have not been performed properly. It is indicated in average total mean row that gained less than satisfied agreement level. Our data showed that the management is very supportive in the KMP and peoples in the organization are always communicate, share and having good relationship among them. This issue could be identified in open-ended answer related to which indicates KMP were done informally through sharing activities and informal documentation such as personal or impromptu process as the interview extracts below show: *"We are doing more on*

self-learning and sharing among us", "Regular sharing process, internal sharing and team work".

In addition to the above analysis, the analysis of the knowledge loss issue have indicate that the informal process environment in VSEs helps the companies to mitigate knowledge loss problems from happened. The analysis in this part showed 90% of the respondents claimed that they do not face knowledge loss problem in their company due to the informal process. These interview extracts illustrate this situation: *"Ensuring that no single member of staff has any exclusive knowledge by using a mentoring/buddy system.", "Not a problem since we are using the same technology and process in all our project…. We occasionally sharing and transferring knowledge among each other".*

V. CONCLUSION AND FUTURE WORK

The analysis has indicated that VSEs have a clear KMP in their organization. The results also show the knowledge atrophy problem is not a serious problem in VSEs. From the analysis we found that due to small team size which creates a flat work structure, direct and active communication, close relationship and open environment have created positive team dynamics environments in respondents' organization. These situations also have encouraged software development teams to share and create knowledge in organization. In addition the analysis in the first stage (qualitative) have indicated that management style in VSEs which is more informal and macro, and working style which more autonomous have helps to create team dynamics environments. This situation help VSEs enhance their KMP and mitigate several factors which lead to knowledge atrophy problems. This is shown from the analyses which have indicated that in VSEs knowledge sharing level is high; staff turnover rate is low, high levels of knowledge exploration, continuous guidance from the senior staff and active communication in exchanging idea or knowledge among staff. Meanwhile in second stage data analysis process indicates that 90% from our research respondents believed that informal process environment in their organization has helped the development team to become more dynamic and this situation has assisted them in KMP beside mitigated knowledge atrophy problem from happened. In addition, the second stage data analysis result also shows that 80% of respondents claimed that their software development activities are not affected by the knowledge atrophy problem. They claimed that by, having frequent guidance and mentoring activities, being active in knowledge sharing and proactive coaching could mitigate this problem from occurring.

REFERENCES

[1] Laporte, C.Y., Alexandre, S., and O'Connor, R. A Software Engineering Lifecycle Standard for Very Small Enterprises, R.V.O'Connor et al (Eds) Proceedings of EuroSPI Springer-, CCIS Vol. 16, 2008.

[2] Plessis, M., , "Knowledge management: what makes complex implementations successful?", Journal of Knowledge Management, Vol. 11, No. 2, , pp. 91-101, 2007.

[3] S Basri and O' Connor, RV. "Evaluation of Knowledge Management Process in Very Small Software Companies : A Survey, Proceeding of Knowledge Management" 5th International (KMICe 2010) Conference, 25-27 May, Kuala Terengganu, Terengganu, 2010

[4] Sapovadia, V. Rajlal K., " Micro Finance: The Pillars of a Tool to Socio-Economic Development. Development Gateway", 2006. Available at SSRN: http://ssrn.com/abstract=955062.

[5] Coleman, G. and O'Connor, R. V., "The influence of managerial experience and style on software development process". International Journal of Technology, Policy and Management. Vol. 8, No. 1, 2008.

[6] ISO/IEC DTR 29110-1, "Software Engineering - Lifecycle Profiles for Very Small Entities (VSE) -- Part 1: VSE profiles Overview". Geneva: International Organization for Standardization (ISO), 2011

[7] S Basri and O' Connor, RV "Understanding the Perception of Very Small Software Companies towards the Adoption of Process Standards", Systems, Software and Services Process Improvement, Communications in Computer and Information Science, Volume 99, 153-164, , Springer-Verlag, 2010

[8] Rosen, C.C.H., "The Influence of Intra Team relationships on the systems Development Process: A theoretical Framework of Intra-Group Dynamics.", 17th Workshop of the Psychology off Programming Interest Group, Sussex University, 2005

[9] Sawyer S. and Guinan P. J., , 'Software development: processes and performance', IBM Systems Journal, Vol. 37, Issue. 4, 1998

[10] Sommerville, I., 2011, Software Engineering, 9th Edition, Pearson, NY

[11] Cohen, S. G. and Bailey, D. E., , "What Makes Teams Work: Group effective Research from The Shop Floor to the Executive Suite", Journal of Management, Vol. 23 No. 3, pp 234-256. 1997.

[12] Hall, T., Beecham, S., Verner, J. and Wilson, D.,. "The Impact of Staff turnover on Software Project: The Importance of Understanding What makes Software Practitioners Tick", Proceedings of ACM SIGMIS CPR, ACM New York, pp. 30-39, 2008.

[13] Basri, S. "Software Process Improvement in Very Small Entities", PhD Thesis, Dublin City University, Ireland. 2010,

[14] McCarty, B., "Dynamics of a successful Team. What are the enablers and barriers to High Performing Successful Teams?" MSc Dissertation, Dublin City University. 2005.

[15] Scarnati, J. T."On becoming a team player", Team Performance Management, Vol. 7, Issue.1/2, pp. 5 – 10, 2001.

[16] Triplett. N., "The Dynamogenic Factors in Pace making and Competition". American Journal of Psychology, Vol. 9, Issue. 4, pp. 507, 1998.

[17] Katzenbach, J.R., and Smith D.K., "The Wisdom of Team. Creating the High Performance Organization". Harvard Business Scholl Press, Boston, MA. 1993.

[18] Gorla, N and Lam, Y.W., "Who Should Work With Whom? Building Effective Software Project Teams", Communications of the ACM ,Vol. 47, Issue. 6, pp. 123. 2004.

[19] Singh, S. K., "Role of leadership in knowledge management: A study", Journal of Knowledge Management, Vol.12, Issue. 4, pp.3 – 15, 2008.

[20] Sarma, A. and Van der Hoek, A., 2004, "A Need Hierarchy for Teams", www.ics.uci.edu/asarma/maslow.pdf

[21] Furumo, K. and Pearson, J.M., "An Empirical Investigation of how Trust, Cohesion and Performance Vary in Virtual and Face to Face Teams". System Sciences, Proceedings of the 39th Annual Hawaii International Conference, Vol. 1, pp. 26c- 26c ., 2006

[22] Levi, D. 2001, Group Dynamics for Teams, Sage Publications.

[23] Kirkman, B.L., Rosen. B, Tesluk, P. E. and Gibson, C.B., "The impact of team empowerment on virtual team performance: The moderating role of face-to-face Interaction", Academy of Management Journal, Vol. 47, No 2, pp. 175-192. 2004

[24] Javed, T., Maqsood, M. and Durrani Q., 'A Survey to Examine the effect of Team Communication on Job satisfaction in Software Industry', ACM SIGSOFT Software Engineering Notes, Vol. 29, No. 2, pp. 6.

Quick Acquisition of Topic-based Information/Knowledge from News Site Databases

Hao Han
National Institute of Informatics (NII), Japan
han@nii.ac.jp

***Abstract*—Web news is an important resource of information/knowledge. We can analyze news to observe the difference in various topics (e.g. economy, health, and culture) and trends in the past years. However, the collection of topic-based Web news is considered as a long-period process usually. In this paper, an effective and efficient Web-based knowledge acquisition approach is proposed to extract topic-based Web news full contents from the news site databases directly. This approach is applicable to the general news sites, and the experimental results show that it can extract the topic-based Web information/knowledge from news site databases automatically, quickly and accurately.**

***Index Terms*—Web-Based Tools, Knowledge Acquisition, News, Extraction, Database**

I. Introduction

Nowadays, fresh news contents on a variety of topics are being created and made available on the Web at breathtaking speed. We can analyze them to acquire the desired information/knowledge. For example, if we want to compare the monthly topics of each country in the past years from CNN, we need to collect the CNN news about each country and analyze these news contents to learn the desired information for personal use (*not anti-copyright republication*).

However, the process of the news pages collection consumes much time. Usually, the Web pages crawlers are used to collect the Web pages. They are executed at regular intervals, and the collection process has to last for a long period of time if we want to collect the news pages of long period. We do not think each one of the collected Web pages is usable because there are many non-news Web pages, such as the blog pages, advertisement pages and even similar pages with different URLs. Sometimes, we just want to collect the news with specific topics such as the news on ”soccer” or ”whaling”, and the other collected news are undesired. Furthermore, the news sites are crawled to find as many news pages as possible, but actually, it is difficult to acquire the old news pages because the latest news are shown prior to the old news. Besides, in a news page, there are advertisement, related stories and other undesired parts usually. In order to recognize and extract the parts of news contents from the news pages, the extraction patterns are generated based on the layout of news pages. Web page layout is the style of graphic design in which text or pictures are set out on a Web page. The different news sites use the different news pages layout, and each news site uses more than one layout usually. It is necessary to generate many news contents extraction patterns manually or automatically for each news site. It is a costly work. Moreover, the news sites update the layout of news pages irregularly. If the news sites update the layout of news pages, the corresponding analysis has to be done again. Therefore, it is not easy to extract news contents on the specific topics from news sites quickly, and the current methods of news pages collection and news contents extraction can not work efficiently.

In this paper, we propose an approach to extract the topic-based Web information/knowledge from news site databases quickly. Usually, the news sites provide site-side news search engines for the users. These engines are affiliated to the news sites and can access the news databases of news sites directly. We use these news search engines to search for the news by giving the keywords of specific topics, and extract the page URLs and titles of the matched news from the search result pages automatically. Then we use an efficient extraction algorithm to extract the full contents of the news without Web page layout analysis. We can designate the target news sites, publication dates/periods (e.g. last week, this month, from 2008 to 2010) and topics. A topic is a discrete piece of content that is about a subject, such as a series of countries, sports, companies and etc. Our approach is applicable to the general news sites and can extract a large number of news including the old ones published some years ago. *Our main purpose is to provide a practical and easy-to-use Web-based information/knowledge acquisition tool for news-oriented research*.

The organization of the rest of this paper is as follows. In Section 2 we give the motivation of our research and an overview of the related work. In Section 3, 4 and 5, we explain our topic-based Web information/knowledge extraction approach in detail. We test our approach and give an evaluation in Section 6. Finally, we conclude our approach and give the future work in Section 7.

II. Motivation and Related Work

In order to realize the analysis and comparison of Web news in many major topics, it is necessary to collect the news with specific topics from one or many designated news sites over a long period of time. Some related work has been done on Web news *collection* or *extraction*. For the news pages collection, the Web pages crawlers are often used. They are executed to collect the news pages from news sites and the collection process costs much time. Several collection approaches and systems have been proposed. More and more news sites distribute news by RSS. Generally, news sites classify the

news into different categories and publish them by RSS feeds. However, different news site uses different categories and RSS feeds just comprise the latest news. For example, CNN provides RSS feeds by fields such as science, sports, business and etc, while AllAfrica (allafrica.com) offers RSS feeds grouped by countries/regions. AllInOneNews (www.allinonenews.com) is a news search system based on automatic extraction of search results from search engines [9]. It passes each user query to the existing search engines of news sites, collects their search results for presentation to the user. However, the users of this system can not select the target news sites, and just collect the results from the first search result page. Google News (news.google.com) provides the news search service and distributes the news search results by RSS or Atom. If we use the default or advanced search of Google News, we can select the target news sites, but the publication date/period selection is weak. If we use the archive search, we can not select the target news sites. If we use the search result RSS feeds, only the results from the first search result page can be collected. These methods/systems can not satisfy the flexible and quick collection of news pages very well. Moreover, these methods can not realize the comprehensive analysis or comparison of news because they can not extract the full contents of each news. They can not easily answer the questions like ”which countries had an argument over whaling during the last years and whether the other countries were attracted to discuss it as the arguments went on”.

For the news contents extraction, a number of approaches have been proposed to analyze the layout of the news pages with the purpose of manual or semi-automatic example-based information extraction pattern learning, and to extract the news contents from the general news pages ultimately. Reis et al. gave a calculation of the edit distance between two given trees for the automatic Web news contents extraction [2]. Fukumoto et al. gave the focus on subject shift and presented a method for extracting key paragraphs from documents that discuss the same event [3]. It uses the results of event tracking which starts from a few sample documents and finds all subsequent documents. However, if a news site uses too many different layout in the news pages, the learning procedure costs too much time and the precision becomes low. Zheng et al. presented a news page as a visual block tree and derived a composite visual feature set by extracting a series of visual features, then generated the wrapper for a news site by machine learning [10]. However, it uses manually labeled data for training and the extraction result may be inaccurate if the training set is not large enough. Webstemmer [7] is a Web crawler that automatically extracts main text of a news site without having banners, advertisements and navigation links mixed up. It analyzes the layout of each page in a certain web site and figures out where the main text is located. All the analysis can be done automatically with little human intervention. However, this approach runs slowly at contents parsing and extraction, and sometimes news titles are missing. TSReC [6] provides a hybrid method for news contents extraction. It uses tag sequence and tree matching to detect the parts of news contents from a target news site. However, for these methods, if the news sites change the layout of news pages, the analysis of layout or tag sequence has to be done again. As the layout-independent extraction approaches, TidyRead (www.tidyread.com) and Readability (lab.arc90.com/experiments/readability) render Web pages with better readability as an-easy-to-read manner by extracting the context text and removing the cluttered materials. They run as plug-in or bookmarklet of Web browser. However, the extraction result is a part of Web page containing the HTML tags. It also contains some other non-news elements such as the related links. Wang et al. proposed a wrapper to realize the news extraction by using a very small number of training pages based on machine learning processes from news sites [8]. The algorithm is based on the calculation of the rectangle sizes and word numbers of news title and contents. However, these approaches still need to set the values of some parameters manually, and could not be proved to extract the news successfully or automatically if news sites update the page layouts. Full-Text RSS (echodittolabs.org/fulltextrss) only returns the news contents when the supplied RSS has a summary or description of some kind.

These news contents extraction methods are still not widely used, mostly because of the need for high human intervention or maintenance, and the low quality of the extraction results. Most of them have to analyze the news pages from a news site before they extract the news contents from this news site. If we select the different target news sites, topics and publication dates, the analysis of layout needs to be done again. It is costly and inefficient. Compared to these developed work, we use the news search engines affiliated to the news sites instead of the often used Web crawlers. We can get a large number of news from the news site databases, not only the latest news but also the old news. The target news sites, topics and publication dates are selective. Furthermore, we do not need to delete the non-news pages or other undesired news pages from the search results because all the news extracted from search result pages satisfy our designated topics. Meanwhile, we propose an algorithm special for the news contents extraction. It is applicable to the general news pages, and we do not need to analyze different kinds of news pages to generate the corresponding extraction patterns for each news site. The full contents of news are quickly extracted from the matched news pages for the further analysis.

III. Overview

Our approach is made up of two parts mainly as shown in Fig. 1: news pages collection and news full contents extraction. Firstly, we collect the topic-based news pages. We create a submitting emulator to emulate the submitting process of search engine of the target news site. We get the search keywords of a specific topic and send them to the emulator one by one, then extract news titles and URLs of news pages from the continuous search result pages. Secondly, we extract the news contents from the news pages. We propose an extraction

algorithm special for news pages, which can extract the news contents from a news page only by using the news title.

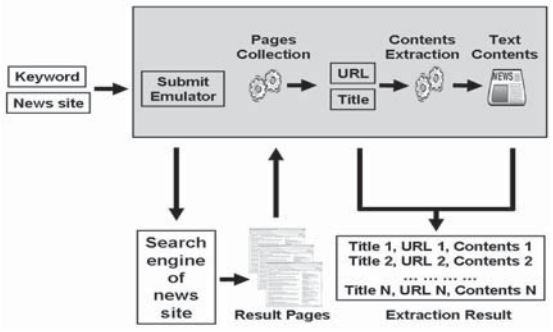


Fig. 1. Overview of our system

IV. News Pages Collection

We collect news pages from news site search engines. Although many Web sites, such as Amazon and YouTube, open their search engine services by Web service APIs, most of the news sites, such as CNN and BBC, do not provide Web services for their news search engines. We have to extract the partial information, such as news titles, news page URLs and publication dates, from the news search result pages. As shown in Fig. 2, we generate a submitting emulator for designated news site, and send the search keywords to the submitting emulator to receive the search result pages. Then, we analyze the search result page to extract the news titles and URLs.

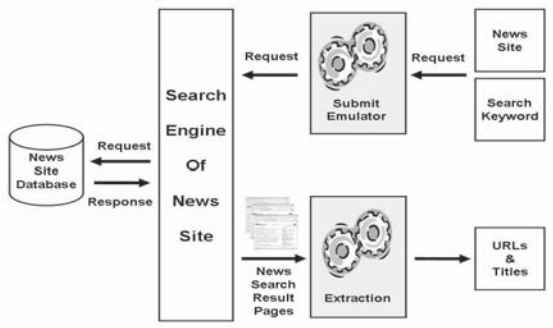


Fig. 2. Overview of news pages collection

A. Submitting Emulator

Usually, in a news Web site, there is a site-side search engine used to get the requests from users and return the search result pages. The users enter the query keywords into a form-input field by keyboard and click the submit button by mouse to send the query. For the request submitting, there are POST method and GET method, and some news Web sites use the encrypted codes or randomly generated codes. In order to get the search result pages from all kinds of news sites automatically, we use HtmlUnit (htmlunit.sourceforge.net) to emulate the submitting operation instead of URL templating mechanism.

We need to get the start Web page which comprises the form-input field and submit button of search engine. Usually, this start Web page is the top page of news site or a search result page of news site. Then we analyze the HTML document of this Web page to find the <form> nodes. If a form comprises *a text input field* and *a button next to this text input field*, and *the server-side form handler of this form is within this news site*, we think it is a possible form which includes the necessary form-input field and submit button. If we find more than one possible form in this Web page, we choose the first one as our final selection because the target form is at the top side of Web page usually.

We generate a submitting emulator and send the search keyword to it. Submitting emulator uses HtmlUnit to emulate the submitting process (input the search keyword into text field and click the button to complete the actual submit). Finally, we get the response page (search result page) from submitting emulator. All the processes of submitting emulator generation in our approach are completed *automatically* after the start Web page is given.

B. News Title and News Page URL Extraction

After we get the search result page, we need to extract the news titles and news page URLs. There are links to the advertisement pages, video pages, external non-news pages and other irrelevant information besides the page links to matched news. Fortunately, there are some similar features in the news search result pages of most news sites, which can be used to extract the news titles and links to the news pages.

- Each search result set contains the similar information at the similar position such as news page link, news title, headline and publication date.
- The news title is contained inside the news page link as text value.
- Matched news are listed in a column and spread over multi-pages.
- All the news search result pages from a news site search engine have the same Web page layout.

We extract all the link nodes from the HTML document of search result page, and find out the news page link nodes. Through our analysis of search result pages of many news sites, we find that the news page link node has some common features in its text and path (XPath expression). We calculate the possibility value of link node by the following steps. Usually, a larger possibility value represents the corresponding link node is more possible to be a news page link node.

1) We split the text value of link node into word list WL using whitespace as the delimiter, and get the length of WL as L_1 ($L_1 \geq 1$).
2) We calculate the occurrence time of search keyword in WL as L_2 ($L_2 \geq 0$).
3) We get the path of link node including the ID and class value. We calculate the occurrence time of "news" and "search" and "result" in path respectively. We count these three values up to get the sum value L_3 ($L_3 \geq 0$).
4) We calculate the possibility value of each link as P by using the following formula.

$$P = L_1 \times (L_2 + \alpha) \times (L_3 + \beta) \quad (1)$$

where, $\alpha = 0$ if $L_2 > 0$, and $\alpha = 1$ if $L_2 = 0$. Similarly, $\beta = 0$ if $L_3 > 0$, and $\beta = 1$ if $L_3 = 0$. We can not make certain that the search keyword must occur in the news title because the search range contains not only the news title but also the news contents, which is not visibile in the search result page. Also, the value of L_3 may be 0 in many news sites.We use α and β to avoid the possible occurrence of *P=0*. They work well and do not bring the negatives to the actual extraction in our experiments.

In some news search result pages, the link node with the largest possibility value is not a link node of the news page always (e.g. a link to contextual advertising or blog). Usually, the news page links are listed in the similar structures (XPath) and not mixed with other non-news links. We use the following steps to detect the news page link nodes range and find out a news page link node.

1) We count up the possibility value P_N of all the link nodes, and get the root mean square RP as a threshold.

$$RP = \sqrt{\frac{\sum P_N^2}{|N|}} \quad (2)$$

where, $|N|$ is the sum of link nodes.

2) For each Node N, we calculate P_N as follows.

$$P_N = \begin{cases} P_N & \text{(N is a link node)} \\ \sum_{n \in Child_N} P_n & (P_n > \text{RP}) \end{cases} \quad (3)$$

where, $Child_N$ is a set of child nodes of the node N. Fig. 3 shows an example of calculation of P.

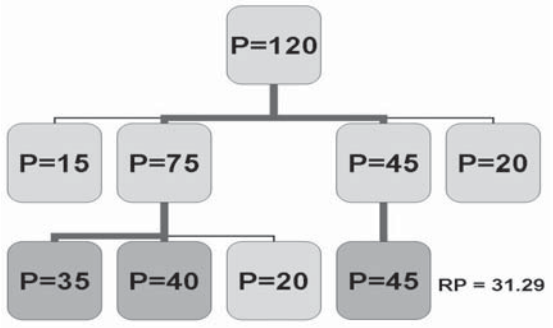


Fig. 3. Calculation of P

3) We select the child node whose value is the largest among the sibling nodes from root node to leaf nodes as shown in Fig. 4.

We think the final selected child node is the link node of a news page, and use the path of this node to extract the list of nodes with the similar paths (like WIKE [5]). Each node of list represents a news page link node, and the text value from each node is the news title.

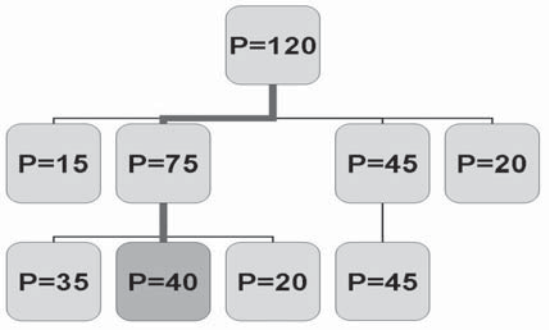


Fig. 4. Selection (largest P)

If the paths of these nodes show that these nodes are listed in a column, we think they are the final extraction results and represent the news page links because most news sites show the search results in a column, not in a row. Otherwise, we think the search result page does not comprise the matched news and shows the message like ”No Results Found” because the search engine does not find the corresponding news about the search keyword in news database.

The news search results are spread over multi-pages and we need to extract the page number links for our continuous query and extraction. The extraction of these links has to satisfy the following rules.

1) The text values of links are a series of numbers such as 1,2,3....
2) The *href* attribute values of links have the similar length.
3) The links are listed in a row.

We use the paths of news page link nodes to extract the news page link nodes directly from the second result page. If we search for the news for many search keywords continually in a news site, we use these paths to extract the news page links and page number links directly.

C. Publication Date Extraction

The publication date is necessary if we collect the news of a specific period. For example, we need to extract the news of baseball of the last 5 years if we want to find out which team was the annual focus of attention in the last 5 years.

Different news sites choose the different formats of date information such as ”March 7, 2011”, ”Mar. 7, 11” and ”2011-3-7”. Table I shows the most used format patterns of date information. We use these patterns to find the publication date in search results. Usually, a news site displays the publication date at the same position in a search result and uses a same pattern for all the publication dates in all search results. After we find a publication date in a search result, we can use the similar paths and same pattern to extract other news publication dates from search results easily.

V. News Full Contents Extraction

We extract the full contents of news from our collected news pages. The news pages from different news sites use the different page layout and news sites update their news page

TABLE I
PUBLICATION DATE FORMAT PATTERN

Pattern	Component	Format	Example
YY	Year	2-digit number	11
YYYY	Year	4-digit number	2011
MMM	Month	Text	Mar, March
MM	Month	Number	3, 03
DD	Day	Number	7, 07
Mark	Delimiter	Character	',' '-' '/' ' '

layout irregularly. We propose our news contents extraction algorithm, which is independent of the layout of news page and applicable to the general news pages. We detect the position of a news title in the news page and extract the body of news (paragraphs of news contents).

The first process detects position of a news title in the obtained news page. The news title is a piece of important information for the recognition of the news contents from the full text of news page. If we correctly locate the position of the title in a news page, the position of news contents text would be found easily because the contents text is a list of paragraphs closely preceded by the title usually. In addition, for a news, the contents describe the same topic of news title in detail, and the words constituting the title would occur in the news contents frequently usually.

The second process detects a part of the news body and extracts the whole body. Since body of a news is usually preceded by its title, the process tries to find the news body in some "contents ranges" at first, and, if it cannot find out the body in the range, it tries to find the body in a "reserve range". "Contents range" and "reserve range" are parts which might include the news body. We gave a detailed description of extraction algorithm in [4].

VI. EVALUATION

In this section, we give the experiments to test our algorithms and analyze the experimental results to evaluate our approach. We use the news sites listed in Table II as our test bed. These news sites are the popular on-line news publishers, including the global and domestic news sites.

A. Experiment 1

We selected the countries/regions and their leaders as our test topics. There are 242 countries/regions in the world and most of them have the leaders [1]. We used these country/region names and leader names as our search keywords. We collected the news page URLs and titles from the 10 (No.1-No.10, if the total number of pages is less than 10, we got them all) result pages of each keyword. As shown in Table II, *Page Collection* is the average execution time of extracting the news page URLs and titles from *one result page* (including the submitting emulation process), and *Contents Extraction* is the average execution time of extracting the news contents from *one news page*. The submitting emulation of two news sites (Washington Post and Xinhuanet) failed, and the corresponding *Contents Extraction* values are calculated by extracting news contents from manually collected/saved result pages. We selected 500 news page URLs randomly and checked them one by one manually, and found that 17 news pages could not be obtained (the server responded the message like "page not found").

TABLE II
LIST OF NEWS SITES AND EXECUTION TIME

Country/Region	News Site	Page Collection (second)	Contents Extraction (millisecond)
United States	CNN	14.3	6.02
	New York Times	4.6	0.63
	Washington Post	×	(*6.62*)
United Kingdom	BBC	3.6	3.19
Africa	All Africa	7.3	2.98
China	Xinhuanet	×	(*2.94*)
France	France 24	8.6	3.67
Japan	Mainichi Daily News	9.3	2.40
South Korea	Chosun Ilbo	6.6	0.52

B. Experiment 2 (precision $\simeq$ 97.0%)

We sent the keywords used in Experiment 1 to submitting emulator one by one, and extracted 96,095 news titles and page URLs of matched news (published from January 1, 2003 to December 31, 2007) from news database of CNN. Our computer (CPU: Intel Pentium M 1.30GHz, Memory: 1.0GB RAM, Network: 54.0Mbps Wireless) uses about 20 hours to complete this extraction process. We selected 200 news page URLs randomly and checked them one by one manually. The experiment results are listed in Table III. We found that 2 news pages could not be obtained as the reasons described in Experiment 1. Among the rest 198 news pages, the news article contents of 192 news pages are extracted correctly. In the 6 extraction failures, some parts of news article contents are not extracted.

TABLE III
EXTRACTION RESULT OF EXPERIMENT 2

Sum	Extracted	Success	Failure	Precision
200	198	192	6	96%

C. Experiment 3 (precision $\simeq$ 97.4%)

We have crawled and extracted more than 1.8 million news contents from 38 famous news sites [4] since 2007. We select 2500 news articles randomly and check them one by one manually. The experiment result is listed in Table IV.

TABLE IV
EXTRACTION RESULT OF EXPERIMENT 3

Sum	Success	Failure	Precision
2500	2434	66	97.4%

D. Analysis and Evaluation

We use the first and second experiments to test our submitting emulator and news page collection algorithm. It proves that our approach can extract the news titles and URLs of news pages of a long period from news site databases easily and

quickly. Our approach is applicable to the general news site search engines and does not need the methods like machine learning or extraction pattern matching, which cost much time when news sites change the layout of search result pages. However, some news sites use the external JavaScript files comprising the complicated JavaScript functions to realize the request submitting, or even the minor syntax errors occur in Web pages where the search keywords are inputted. Although the most of the current Web browsers, such as Firefox and Internet Explorer, can run smoothly on these Web pages, our submitting emulator still can not emulate this kind of submitting processes. We think it is a bug of HtmlUnit and wish the new version would solve this problem in the future. Furthermore, the emulation processes of some news sites run slowly. For example, the emulation of submitting search keyword to CNN news search engine costs about 10 seconds. Moreover, some old news pages are not obtained though their URLs and news titles are shown in news search result pages.

We use the second and third experiments to test our news full contents extraction algorithm. We use a large number of news to test our algorithm and the experimental results prove that our extraction algorithm is highly accurate over a long period of time. Although the news sites change the layout of news pages irregularly, our extraction method works well and the precision of extraction is over 97%. However, in some news pages, a paragraph, usually the outline of news, shows in different style compared to other paragraphs. This kind of paragraph looks like a non-news part such as an advertisement in text format, and is omitted in the extraction. Moreover, some news contents are too short to recognize from the news pages. For example, a news flash about baseball game result, which contains just a short paragraph of ten words, maybe can not be extracted correctly.

Compared with other developed extraction systems, our extraction approach has the following strong points.

1) Our extraction system is constructed easily, even for the users who know little about the information extraction technologies. The extraction processes run automatically, such as submitting emulator generation, news pages collection and news contents extraction. It needs little maintenance during the long period extraction. We do not need to analyze the layout of search result pages and news pages of news sites since our extraction algorithm is independent of the layout of Web pages. It does not need to reconfigure extraction even though the news sites change the layout of news pages.
2) Our extraction system supports the designation of news collection/extraction range, such as the target news site, news topic and publication date. By analysis of extracted news contents, we can compare the viewpoint of a topic among different news sites, see monthly/yearly variation of a topic, observe co-occurrence of one or two country names, and find other useful information/knowledge.
3) Our extraction system runs quickly because of simple and efficient algorithms. For the extraction of a large number of news, a simple algorithm of low computational complexity saves a considerable amount of time. For example, the contents extraction from a CNN news page costs 6.02 milliseconds averagely (excluding reading news pages from news sites and saving the extraction results into local hard disk), which is more efficient than other developed methods.

VII. Conclusion

In this paper, we have presented an effective and efficient approach to realize the quick and automatic extraction of topic-based Web information/knowledge from news site databases by using the site-side search engines. We proposed an algorithm to extract the news titles and news page URLs from search result pages. We also proposed an algorithm to extract the news full contents from news pages. Our extraction methods are applicable to the general news sites. All the processes of extraction are completed automatically. Our experimental results on several news sites show that our extraction system works well and the proposed approach is very promising.

As future work, we will modify our algorithm to improve the accuracy rate even further, and observe difference in various topics among countries/regions to discover useful information and knowledge from news sites. Moreover, we will extend our approach to different kinds of information/knowledge sites and construct the corresponding analysis system.

VIII. Acknowledgement

We gratefully acknowledge the advice and support from Bin Liu (Yahoo! Japan), Takehiro Tokuda (Tokyo Tech) and Keizo Oyama (NII). This work was partially supported by a Grant-in-Aid for Scientific Research A (No.22240007) from the Japan Society for the Promotion of Science (JSPS).

References

[1] BBC Country Profiles. http://news.bbc.co.uk/1/hi/country_profiles/default.stm.
[2] D. de Castro Reis, P. B. Golgher, A. S. da Silva, and A. H. F. Laender. Automatic Web news extraction using tree edit distance. In *The Proceedings of the 13th International Conference on World Wide Web*, pages 502–511, 2004.
[3] F. Fukumoto and Y. Suzuki. Detecting shifts in news stories for paragraph extraction. In *The Proceedings of the 19th International Conference on Computational Linguistics*, pages 1–7, 2002.
[4] H. Han, T. Noro, and T. Tokuda. An automatic Web news article contents extraction system based on RSS feeds. *Journal of Web Engineering*, 8(3):268–284, 2009.
[5] H. Han and T. Tokuda. WIKE: A Web information/knowledge extraction system for Web service generation. In *The Proceedings of 8th International Conference on Web Engineering*, pages 354–357, 2008.
[6] Y. Li, X. Meng, Q. Li, and L. Wang. Hybrid method for automated news content extraction from the Web. In *The Proceedings of the 7th International Conference on Web Information Systems Engineering*, pages 327–338, 2006.
[7] Y. Shinyama. *Webstemmer*. http://www.unixuser.org/~euske/python/webstemmer/.
[8] J. Wang, X. He, C. Wang, J. Pei, J. Bu, C. Chen, Z. Guan, and G. Lu. News article extraction with template-independent wrapper. In *The Proceedings of the 18th International Conference on World Wide Web*, pages 1085–1086, 2009.
[9] H. Zhao, W. Meng, and C. Yu. Automatic extraction of dynamic record sections from search engine result pages. In *The Proceedings of the 32nd International Conference on Very Large Data Bases*, pages 989–1000, 2006.
[10] S. Zheng, R. Song, and J.-R. Wen. Template-independent news extraction based on visual consistency. In *The Proceedings of the 22th AAAI Conference on Artificial Intelligence*, pages 1507–1513, 2007.

Using Contextual Information to Improve Awareness in Software Development

Bruno Antunes*, Joel Cordeiro, Pedro Costa and Paulo Gomes
Centre for Informatics and Systems of the University of Coimbra
Coimbra, Portugal
{bema,jfac,pacosta,pgomes}@dei.uc.pt

Abstract

The use of contextual information is said to improve awareness in software development. But the context of a software developer is something hard to define and capture, as it represents a complex network of elements across different dimensions, that is not limited to the work developed on an IDE. We propose a software developer context model composed of four layers: personal, project, organization and domain. We describe this context model at the personal layer, present a prototype and discuss the results of an experiment conducted with a group of developers. The results show that developers consider the use of contextual information relevant to improve information retrieval, ranking and filtering, but usability plays an important role on how these improvements are perceived.

1. Introduction

The interest in the many roles of context comes from different fields such as literature, philosophy, linguistics and computer science, with each field proposing its own view of context [5]. The term context typically refers to the set of circumstances and facts that surround the center of interest, providing additional information and increasing understanding. The context-aware computing concept was first introduced by Schilit and Theimer [7], with further definitions given by Brown et al. [1] and Dey et al. [3]. In software development, the context of a developer can be viewed as a rich and complex network of elements across different dimensions that are not limited to the work developed on an IDE (Integrated Development Environment). During their work, software developers need to cope with a large amount of contextual information that is typically not captured and processed in order to enrich their work environment.

We propose a software developer context model that takes into account all the dimensions that characterize the work environment of the developer, which is described in section 2. These dimensions can be represented in a model with four layers: personal, project, organization and domain. In section 3, we present a prototype that deals with the first layer of this model, creating and maintaining the developer context model at the personal layer. The contextual information is then used to rank, filter and suggest relevant knowledge to the developer. This prototype was submitted to an experiment, using a group of developers, discussed in secton 4. From the analysis of the results we conclude that the use of contextual information is considered relevant to improve the retrieval, ranking and filtering of relevant information to the developer. But, this study also shows that usability issues may influence the way these improvements are perceived and assimilated by software developers. An overview of related work is given in section 5. Finally, section 6 concludes the work.

2. Developer Context Model

The developer context model we envision is based on a layered model made up of four layers: *personal layer*, *project layer*, *organization layer* and *domain layer*. Each one of these layers focus on different dimensions of the environment where a typical developer works. In this paper, we are focused on the personal layer.

The *personal layer* represents the context of the work a developer has at hands at any point in time, which can be defined as one or more tasks. In order to accomplish these tasks, the developer has to deal with various kinds of resources at the same time, such as source code files, specification documents, bug reports, etc. At this layer, the developer context model comprises a set of artifacts that may be relevant for the task the developer is currently executing. Associated to each artifact is a Degree of Interest (DOI), a concept introduced in [4], that represents the weight of that resource in the developer context model. The context model is captured and maintained through the analysis of the interactions of the developer with the artifacts manipulated in the IDE. An artifact affected by one of these interactions is

*Supported by FCT scholarship grant SFRH/BD/43336/2008.

included in the context model, or gets its DOI updated in case it is already present. We monitor the actions of create, open, edit, focus, close and remove, and each one results in a variation of the DOI value of the affected resource. The artifacts that are related with the affected artifact through "implements", "extends" or "uses" relations, are also added to the context model. As time passes, the DOI values are decreased. The information stored in the context model at the personal layer is used to rank, highlight and filter information taking into account the context of the developer. The way this is accomplished is explained in section 3, where we describe the prototype developed. The *project layer* focuses on the context of the project, or projects, in which the developer is involved. A software development project is an aggregation of a team, a set of resources and a combination of explicit and implicit knowledge that keeps the project running. The project layer represents the people, resources and tasks, as well as their relations, of the software development projects where the developer is included. The *organization layer* takes into account the organization context to which the developer belongs. Similarly to a project, an organization is made up of people, resources and their relations, but in a much more complex network. The *domain layer* takes into account the knowledge domain, or domains, in which the developer works. This layer goes beyond the project and organization levels and includes a set of knowledge sources that stand out of these spheres.

3. Prototype

We have developed a prototype that makes use of the developer context model to improve developer awareness. Here we will be focusing the personal layer of the context model, thus the components that support the remaining layers will not be described in detail. The prototype works in a client/server architecture. The client application is an *Eclipse* plug-in that manages the contextual information of the personal layer and integrates contextual information from other layers directly into the IDE. The server is used for mining and storing contextual information of the project layer, which is then made available for the client applications. The use of contextual information to improve information retrieval directly in the IDE, where developers perform most of their work, increases awareness and reduces the effort put on finding information that would be hidden and dispersed otherwise.

The *browsing* functionality is provided by allowing the developer to navigate in a graph (see section 1 of figure 1), where nodes represent resources (source code, documents, tasks and developers) and edges represent relations between those resources. When the developer expands a node the system shows all the information related with that node. This information includes the entities that are statically related with the node, for instance all the classes that implement an interface, as well as the contextual information that is retrieved from the server. This way, the developer can easily gather information about what artifacts are likely to be related with that artifact, what tasks affected that artifact in the past, and what developers may be of help if extra information is needed. This navigation allows the developer to explore the source code using its static structure while obtaining contextual information about the entities being explored. While navigating in the graph, the context model is taken into account to give more relevance to resources that are closer to the context of the developer. The nodes that are more relevant to the developer stand out from the remaining, because their background color is more intense. The related entities retrieved from the server are ranked and filtered using the developer context model, with only the top five entities being shown in the graph. Also, the weight of the relation is reflected in the width of the edge that bonds the node and a related entity. Stronger bonds have a bigger width, showing the developer that a specific entity may be more relevant than others. The *search* functionality allows the developer to easily search the server repository for artifacts, tasks, and resources, through an integrated and easily accessible interface (see section 2 of figure 1). The search results can be organized by type, package and package hierarchy. Because of the different visualization options available, the search result foreground color is used to represent its relevance, starting with black for top results and fading to light grey as relevance drops. Any search result can be open in the graph, allowing the developer to situate it in the source code structure and understand how does it relate with other entities. The context model is used to rank search results taking into account the query score and the proximity to the developer context. The *suggestion* functionality is used to proactively suggest relevant information to the

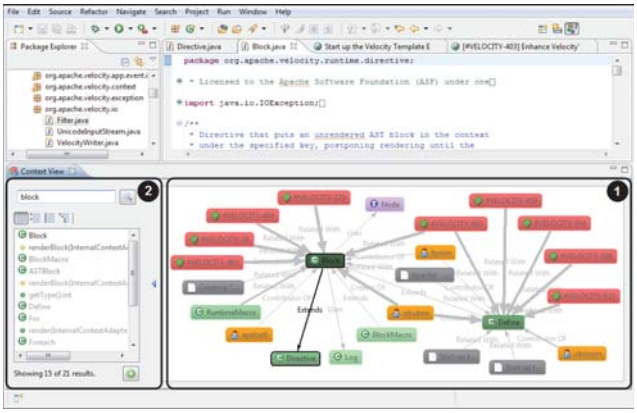


Figure 1. A screenshot of the Eclipse plug-in.

developer. When the developer opens or gives focus to an artifact, the graph automatically opens and expands that artifact, providing suggestions and hints about which entities may be relevant in that context (see section 1 of figure 1). This all happens without the need of the developer to explicitly search for that information. Also, when the developer creates a new entity, the system automatically searches for entities that have a similar name to the one being created, making them accessible through the graph.

4. Experimentation

We have created an experiment to evaluate the acceptance of our prototype among developers. The experiment was executed by a team of 6 developers working in a start-up company that operates in the area of information technology and services. The experience of the developers with *Java* is very varied, ranging from 2 years up to 8, and they have been using *Eclipse* for different periods of time, ranging from 1 year up to 5. We wanted to create a scenario that would encourage the developers to use the prototype, with a complexity that allowed the experiment to take about one hour. We have decided to use a small sized open-source project, completely unknown to all the developers, on which the developers would perform a set of tasks. The project selected was *Apache Velocity*, a Java based template engine, and we asked developers to perform 3 tasks that required them to explore and change the project's source code.

The experimentation environment comprised an *Eclipse* instance running our plug-in and set up with a workspace containing the source code of the selected project. The contextual information extracted in the project layer of the context model (see section 2) was available from a central server. In this server we have indexed the artifacts, tasks and resources provided by the project management tools used in the project, such as the Version Control System (*Subversion*), the Issue Tracking System (*Jira*) and the Collaborative System (*Wiki*). By the end of the experimentation, developers were asked to fill a questionnaire. The objective of the questionnaire was to perceive the opinion of the developers on the relevance and usability of the prototype, as well as the relevance they attach to specific features. We also wanted to know what they liked the most, and the least, and what suggestions they could give us to improve the prototype.

Concerning relevance and usability, the overall results show that developers have considered the application relevant, but usability has been rated as almost neutral (see section Relevance and Usability Questions of table 1). When we analyze the different components of the prototype (search, browsing and suggestion) individually, results are more or less homogeneous, with all of them considered relevant, but having lower scores on usability. From these results we conclude that the application is considered relevant, but usability is a concern that must be addressed in order to improve the experience of the developers.

Table 1. Questionnaire results, ranging from 1 (Very Irrelevant) to 5 (Very Relevant).

RELEVANCE AND USABILITY QUESTIONS	**AVG**	**SD**
Overall, how would you rate the relevance of the application?	3.83	0.48
Overall, how would you rate the usability of the application?	3.50	0.87
How would you rate the relevance of the search component?	4.50	0.87
How would you rate the usability of the search component?	3.67	1.02
How would you rate the relevance of the browsing component?	4.17	0.86
How would you rate the usability of the browsing component?	3.00	0.58
How would you rate the relevance of the suggestion component?	4.17	0.48
How would you rate the usability of the suggestion component?	3.67	1.02
SEARCH FEATURES	**AVG**	**SD**
Sorting of search results based on their relevance to the context of the developer.	4.50	0.87
Coloring of search results based on their relevance.	3.33	1.00
Clustering of search results based on category.	4.00	0.58
Clustering of search results based on package.	3.33	1.02
Clustering of search results based on package hierarchy.	3.67	0.77
BROWSING FEATURES	**AVG**	**SD**
Information about related resources.	4.33	1.02
Information about related developers.	3.00	1.15
Information about related tasks.	3.17	0.86
Color depth of nodes based on their relevance for the developer context.	3.17	1.40
Width of relations based on the relevance of the link.	3.33	0.77
Information filtering by relation type.	3.83	0.48
SUGGESTION FEATURES	**AVG**	**SD**
Proactive suggestion of relevant resources when an artifact is activated.	4.00	0.58
Proactive suggestion of relevant resources when a new artifact is created.	4.00	1.15

The main features provided by the search component were rated as shown in section Search Features of table 1. The use of the context model of the developer to sort the search results was considered very relevant, which shows that developers consider the use of contextual information as being important to improve the process of finding the information they need. The features related with search result clustering were generally rated with close to neutral values, excepting the one based on the package hierarchy, which was considered more relevant.

In relation to the browsing features (see section Browsing Features of table 1), the availability of information about related resources was considered relevant, while information about related developers and tasks was rated as neutral. This can be explained by the fact that information about developers and tasks was not important in the context of this experimentation. We believe that in the context of a real software development project, this information can be valuable, which should be confirmed in future field tests. The use of node colors and relation widths to highlight information based on the developer context model was considered almost neutral, which show that these features must be improved. Although we have implemented these techniques to reduce the information overload, the way they were experi-

enced did not achieve the result we have expected. Finally, some relevance was given to the information filtering features, showing again that the overload of information is a real issue that must be carefully addressed.

The two main features of the suggestion component were rated as relevant (see section Suggestion Features of table 1). This tendency shows that suggestion is a powerful way of retrieving relevant information to the developer, and this is the kind of feature where context plays a central role. The contextual information can be used to determine when to suggested this information and to help filter the most relevant information in that specific moment.

When asked to say what they liked most about the application, many developers referred search as a great feature, because it is an efficient way to quickly find information from different sources. Along with search, the suggestion of relevant information was viewed as important to developers. Other aspects noted as positive were the novelty of the approach and the use of a graph structure, which was considered an interesting way of exploring the source code and related information. On the other hand, the usability of the browsing component was among the things the developers liked the least about the application. Finally, several suggestions for improvements were collected, most of them centered on the browsing and suggestion features. The developers referred the graph layout and overload of information as big issues that must be addressed. To overcome this limitations they suggested that relevance of related resources should be reflected using other techniques, such as their distance to the artifact and the size of the node itself. The overload of information should be addressed with better context-based filtering options.

5. Related Work

Following the line of task management and recovery, Kersten et al. [4] and Parnin et al. [6] propose approaches for capturing the context relevant for a task from the programmer's interactions with an IDE. The focus of these works is the task and the resources present in the IDE that are more relevant for the fulfillment of that task. Our approach aims to define a context model that goes beyond the IDE and explores the knowledge provided by the different systems that support the software development process. Based on the assumption that social dependencies exist between developers implementing modules that are technically dependent, Cleidson Souza et al. created Ariadne [2], a plug-in for the Eclipse IDE that exposes those dependencies through various graph-based views. While the focus here is on finding relationships between code and developers, that are then used to improve awareness, our aim is to use a continuously updated context model of the developer that is used to rank, elicit and filter information through search, browsing and suggestion.

6. Conclusions

We have presented our approach to a software developer context model. The context model is based on a layered structure, taking into account four main dimensions of the work environment of a developer: personal, project, organization and domain. A prototype focusing the personal layer was presented, along with an experiment with a group of developers. The results confirmed our belief that context have a central role when it comes to retrieve relevant information to developers. Furthermore, we have been alerted to the fact that usability may be a serious obstacle to the success of our approach. As future work we plan to take into account the results of the study to improve the prototype.

References

[1] P. J. Brown, J. D. Bovey, and X. Chen. Context-aware applications: From the laboratory to the marketplace. *Personal Communications, IEEE*, 4:58–64, 1997.

[2] C. R. de Souza, S. Quirk, E. Trainer, and D. F. Redmiles. Supporting collaborative software development through the visualization of socio-technical dependencies. In *Proceedings of the 2007 international ACM conference on Supporting group work*, GROUP '07, pages 147–156, New York, NY, USA, 2007. ACM.

[3] A. K. Dey and G. D. Abowd. Towards a better understanding of context and context-awareness. In *CHI 2000 Workshop on the What, Who, Where, When, and How of Context-Awareness*, The Hague, The Netherlands, 2000.

[4] M. Kersten and G. C. Murphy. Using task context to improve programmer productivity. In *Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 1–11, Portland, Oregon, USA, 2006. ACM.

[5] G. K. Mostefaoui, J. Pasquier-Rocha, and P. Brezillon. Context-aware computing: A guide for the pervasive computing community. In *Proceedings of the IEEE/ACS International Conference on Pervasive Services, ICPS 2004*, pages 39–48, 2004.

[6] C. Parnin and C. Gorg. Building usage contexts during program comprehension. In *Proceedings of the 14th IEEE International Conference on Program Comprehension (ICPC'06)*, pages 13–22, 2006.

[7] B. Schilit and M. Theimer. Disseminating active map information to mobile hosts. *IEEE Network*, pages 22–32, 1994.

Evaluation of Semi-Automatic Acquisition of Semantic Descriptions of Web Services

Shahab Mokarizadeh[1], Peep Küngas[2], Mihhail Matskin[1, 3]

[1] Royal Institute of Technology (KTH), Stockholm, Sweden
[2]University of Tartu (UT), Tartu, Estonia
[3] Norwegian University of Science and Technology (NTNU), Trondheim, Norway
[1]shahabm@kth.se, [2]peep.kungas@ut.ee,[3]misha@kth.se

Abstract—**This paper introduces and evaluates a novel Web service interface annotation and matching scheme designed for processing large sets of Web service interface descriptions. The matching scheme relies on a set of rules, which exploit the structural relationships encoded in an automatically learned ontology to discover matching interface elements. The proposed scheme is evaluated on a Web services interface corpus consisting of WSDL descriptions. The experimental results show that the proposed matching scheme can be used for bootstrapping large-scale Web service interface annotation.**

Keywords-Web service annotation; schema matching; XSD; WSDL

I. INTRODUCTION

Lack of good benchmarks has hindered proper evaluation of proposed Web service discovery and composition solutions in practical settings where many Web services are available. To overcome this shortage, there has been recently some activities [3][4][5] in the field for tackling the analysis of essential properties of web services and their networks with the aim to provide feedback to web services discovery and composition problems from the search space topology perspective. Since semantic annotations enable construction of dataflow- and workflow-based networks, which are needed for analysis, there is a need for a fully automatic cost-effective web service annotation mechanism. Previous works in web service analysis suffer from following drawbacks: a) due to lack of semantic annotations in the vast majority of existing web services only pure syntactic matching has been applied [3]; b) only small sets of semantically annotated web services have been examined [5] due to the high costs related to labour-intensive manual annotation of web services [1]; and c) they rely on the assumption that a supporting reference ontology is provided [4].

In this paper, we report work in progress in developing and evaluating a semi-automatic cost-effective semantic annotation approach, which combines a previously proposed ontology learning method [6] and a cost-effective semantic annotation method [1], for bootstrapping analysis of large repositories of web services, which do not include semantic annotations. The ultimate goal of the reported work is to provide means for bootstrapping large scale annotation of Web services to enable further advances in the field relying on these annotations.

We evaluate the approach both qualitatively and quantitatively on our corpus of Web service interface descriptions in two stages. First we manually create a so called *golden* ontology and corresponding annotation heuristics, which will be used then for automated annotation as suggested in [1]. Then we automatically learn a reference ontology and corresponding annotation heuristics by using the scheme proposed in this paper for the same corpus as in the preceding stage and assess the quality of completely automatically generated annotations with respect to manually created ones. The evaluation confirms that the proposed matching scheme can be used for bootstrapping annotation of large scale of Web service interfaces in a semi-automatic way with some certain error margins.

The rest of this paper is organized as follows. In Section 2 we outline our Web service annotation and matching scheme. In Section 3, we discuss our approach to evaluate the quality and quantity of applied annotations. Experimental results of applying the annotations and matching scheme are presented in Section 4. Finally, Section 5 reviews related work, while conclusions and future work are presented in Section 6.

II. WEB SERVICE ANNOTATION AND MATCHING SCHEME

Different Web service annotation and matching schemes have been adopted to deal with integration of heterogeneous information sources. We employ our ontology learning approach [6] to first generate a reference ontology from our collection [1] of Web service corpus (ca 15000 WSDL documents collected from various repositories in the Web) and then utilize the generated reference ontology to annotate the services. In the reference ontology, instances are referring to the *terms* while classes refer to conceptual representation of the underlying *terms*. In this work, *term* refers to an XML schema basic element name or a message part name in the

[1]Available online at : http://www.soatrader.com/web-services

web WSDL document and our goal is to annotate the identified *terms*. The extracted *terms* from the WSDL documents in our collection builds up the dataset which is considered for ontology learning, annotation and matching. Our ontology learning mechanism [6] exploits frequently observed naming patterns in the given dataset and relies on morphological analysis to identify concepts and relations between and generate an ontology accordingly. In the generated ontology, instances refer to the *terms* and the conceptual classes are interrelated further using ontological properties namely: *hasProperty*, *isSynonymOf* and *isSimilarTo*. While *isSynonymOf* property conveys that the concepts on the both side of relation are lexically synonyms, the *isSimilarTo* relation expresses a weaker degree of lexical similarity between two concepts.

We employ the heuristic-based annotation mechanism reported by Küngas and Dumas [1] to annotate Web services. In this mechanism annotation heuristics are represented as rules in the following form: *entity_reference* ← *synset* (e.g. *Password* ← *{password, pwd, strPassword, authpassword*, *pass*}). The meaning of such a rule is that an XML schema element matched by any element in the *synset* is mapped to the *entity reference* (in our case a concept identifier in the automatically constructed ontology) on the left-hand side of the rule. A *synset*, or a synonym ring, is a group of labels (i.e. *terms*) that are considered semantically equivalent. We construct *synsets* from the labels of particular instances in an ontology. Thus according to the heuristic rule example – if *Password* is a concept identifier, then *password*, *pwd*, *strPassword*, *authpassword* and *pass* are labels of instances in the generated reference ontology (i.e. *terms*).

By utilizing the generated ontology and annotating respective Web service elements, we promote the process of correlating Web service inputs and outputs from pure syntactic level to ontological instance matching level. The annotation process simply annotates those extracted schema element names/ WSDL message part names (*terms*) with their respective concepts in the generated ontology. In order to match inputs and outputs of Web services through annotated elements, we rely on a set of matching rules. Here matching refers to the process of finding relationship or correspondence between instances (*terms*) in an ontology through utilization of any of following rules adopted from rules proposed by[11][12]. By definition two instances are matched if and only if one of the following matching rules is true:

Rule-1: They both belong to a same concept (e.g. {*loc, location1*} isInstanceOf *Location*).

Rule-2: They belong to lexically synonym or similar concepts (e.g. (*loc* isInstanceOf *Location*), and (*place* isInstanceOf *Place*) where *Place* isSynonymOf *Location* .

Rule-3: One of the instances belongs to a concept which subsumes the concept representing the second instance (e.g. pair of {*ContractId , Id*} where {*ContractId* isInstanceOf *ContractIdentifier*},and { *ContractIdentifier* isSubClassOf *Indentifer*} and {*id* isInstanceOf *Identifier*}

Rule-4: One of the instances belongs to a synonym or similar concept which subsumes the concept representing the second instance (e.g. pair of {*bidUId , Id*} where {*bidUId* isInstanceOf *BidUniqueCode*},{ *BidUniqueCode* isSynonymOf *ContractIdentifier*}

Rule-5: The instances belong to two concepts inter-related by other ontological properties (e.g. *Address* hasProperty *Address_PostalCode*).

III. Evaluation Approach

In our evaluation, we initially employ our annotation scheme to annotate certain elements (terms) from Web service corpus by generating reference ontology. Next, we evaluate quality and quantity of matching cases discovered using our introduced matching rules operating on the reference ontology.

Since automatic Web service matching is the target use-case for annotated Web services, we measure the quality of pair-wise matches between annotated XML schema element/ message part names. In order to be able to verify the quality of the generated reference ontology and subsequent annotations, initially we limit evaluation dataset to 2000 most frequent *terms* extracted from our collection of WSDL documents. Using this dataset we create two independent ontology resources. While the first ontology is handcrafted by a human expert (i.e. an ontology engineer), the second one is constructed automatically using our ontology learning mechanism, explained at [6] . We refer to the former case as *Golden* ontology while the latter one is called the *Generated* ontology. We acknowledge that *Golden* ontology might suffer from bias introduced by the human expert due to the lack of documentation in the underlying Web services. Next, we align the *Generated* ontology with *Golden* one, using Falcon-AO ontology matching tool [8] and harvest only aligned concepts and their underlying instances. We refer to this set of instances as aligned instances and they account for 968 cases [6]. The evaluation goal is to determine how many true matching cases between every two aligned instances we will gain ,provided that the precision and recall of the aligned concepts are ideal (however in practice there exist some limitations and error margin due to the exploited tool [8]).

We adopt Euznat and Shvaiko's[7] terminology to describe our instance matching process. The result of instance matching process is a set of correspondence elements. Each correspondence element implies that a relation holds, according to a particular matching rule, between two instances in an ontology. A correspondence element $Ont_kC_{i,j}$ is a triple $<a_i, b_j, R>$ where $i \neq j$; $i,j=1...N$; N is the number of instances; a_i, b_j refer to *i-th* and *j-th* instance in the ontology referenced by Ont_k; k is the identifier of the ontology, and finally R specifies the matching rule that reveals kind of semantic relationship holding between two instance a_i and b_j. If two instances are not matched, then we use notation of NM (*NotMatched)* instead of the matching rule. For evaluation purposes, we compare the matching rules R and R' in $Ont_{Gen}C_{i,j=}<a_i, b_j, R>$ and $Ont_{Gold}C'_{i,j}=<a_i, b_j, R'>$ where $Ont_{Gen}C_{i,j}$ denotes the correspondence element obtained in the *Generated* ontology (Ont_{Gen}) while $Ont_{Gold}C'_{i,j}$ refers to the

computed correspondence element for the same pair of instances a_i and b_j in *Golden* ontology(Ont_{Gold}).

IV. Evaluation Results

In this section, we present the experiments[2] made using the top 2000 recurrent terms to generate reference ontology, annotate Web service elements and perform Web service element (i.e. instances in the reference ontology) match-making. Automatic instance matching between aligned instances (968 items) using *Golden* and *Generated* ontology results in 2362 and 3797 correspondence elements respectively. We perform evaluation based on comparison between R to R' in $OntGoldC'_{i,j}=<a_i, b_j, R'>$ and $OntGenC_{i,j}=<a_i, b_j, R>$ for all correspondence elements, as pointed out in Section 3. The comparison results are grouped into three groups based on the exploited matching rules:

Matched in both *(R≠NM & R'≠NM)***:** This category embodies the correspondence elements which are matched in both ontologies and it covers 2837 cases (75% of those discovered by *Golden* ontology) and majority of them (2279 cases) are resulted by Rule-1. Our observation reveals that these identified correct correspondence elements belong to those instances with clear lexical semantic, and to some extent having context independent semantic or those conveying concrete concepts, for example *birthday, Social Security Number, ISBN, authCode, pwd*, etc.

Missing matches (*R=NM* & $R \neq R'$)**:** This group consists of instances that are matched by *Golden* ontology but not by *Generated* ontology and it accounts for 952 cases (25% of those discovered by *Golden* ontology). Besides to typical WordNet [15] limitations [10] one major reason is the fact that concepts in *Golden* ontology are more inter-related (well-organized through handcrafted taxonomical relationships) than those in *Generated* ontology. This leads to loss of possible matches between instances by the ground of subsumption relation between representing concepts (i.e. by Rule-3). This flaw is a direct result of inferring taxonomical relation solely relying on linguistic synthesis. For example, terms "Caller" (as person who makes phone call) and "Person" cannot be correlated, because they are neither synonym according to dictionary nor appeared together in a compound noun, whereas in *Golden* ontology they are assigned to two subsuming concepts. This situation can be alleviated by introducing a new module in our ontology organization step where additional domain resources such as domain ontologies, and taxonomies are incorporated to reorganize hierarchy of generated concepts.

Extra introduced matches (*R'=NM* and $R \neq R'$): This is the group of correspondence elements which are only discovered using *Generated* ontology but not *Golden* one and it consists of 1980 cases. Introduction of these extra matches in *Generated* ontology is due to following reasons: 1) *Golden* ontology lacks any kind of property relations (such as *isSynonymOf*, *hasProperty*, etc); hence it cannot correlate instances by ground of Rules 2,4, and 5. The matching cases resulted by Rule-5 are intuitively reflecting *PartOf* or *Having* relationships between matching instances. 2) in few cases there exist kind of lexical similarity between concept labels , which is not covered by *Golden* ontology for example for the cases when two instances convey a similar semantic but in a different context (e.g. {*taskName*, *jobName*}). In both cases we need to assign a confidence degree expressing trustworthiness of the matching (how close is the matching to ideal case based on contextual information).In this work, we rely on human assessment to distinguish correct cases (true positives) from incorrect ones (false positives). It should be noted the accuracy of these introduced matchings should be considered as lower quality matchings compared to those identified in previous group (i.e. "matched in both").

Fig. 1 shows the assessment results over introduced correspondence elements categorized based on the exploited matching rule. Accordingly, the incorrect figures refer to quantity of cases that two instances are matched while they do not have same/related semantic in practice, while correct cases point to semantically meaningful matched pairs. Accordingly, the quantity of incorrect matches resulted from Rule-4 is the highest among all (89%), while Rule-2 and Rule-5 are equally showing the least amount of incorrect matches (around 40%) and Rule-3 lies almost in the border, with 51% incorrect cases. Our observation over cases in group of Rule-3 shows that occurrence of false positive cases are mostly resulted from generic concepts. For example concept *Code* , in *Generated* ontology, is so generic that can subsumes any of *CurrencyCode*, *LicenseStatusCode*, and, *DesignCode* concepts while in *Golden* ontology, these concepts are more accurately classified by looking into their precise semantic of the term in the Web service corpus. For instance *CurrencyCode* refers to the currency abbreviation rather than a digital code like that of *LicenseStatusCode*. This deficiency can be correlated to limitation of using lexical resources as they do not cover with the same detail different domains of knowledge and also many of domain independent terms, as reported by Bergamashi and Sorrentino [2].

On the other hand, it can be seen in Fig. 1 that subsumption (Rule-3) is reasonably convincing for most of the specialized (concrete) concepts, which in our case are mainly compound nouns. For example, *StartDate* can subsumes any of *futureLaunchDate*, *flightDepartureDate* terms. However, without further investigation we cannot provide any firm theory supporting the aforementioned patterns at this point. The high number of false positives of Rule-4 are due to two

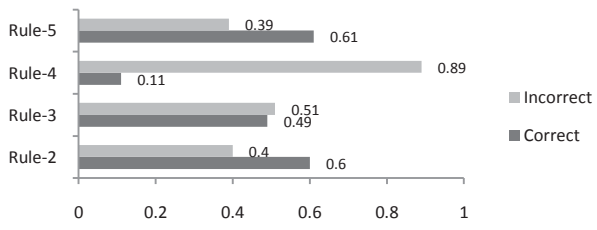


Fig.1 Quantity of "extra introduced mathces" dicovered by matching Rules 2-5 using Generated ontology

[2] Available at: http://www.isk.kth.se/~shahabm/AnnotationAnalysis

reasons: 1) it suffers from subsumption errors; and 2) the rule is based on transitivity of lexical similarity between two terms which is not always true (e.g. the middle term could possess two different meaning in WordNet in different domains). Combination of these two flaws intensifies the vulnerability of this rule. Finally, as there have been no introduced matching cases due to Rule-1, they are omitted from Fig. 1.

According to the aforementioned mentioned evaluations, it can be concluded that without further enhancement, Rule-1, Rule-2 and Rule-5 are the most promising matching rules in our matching scheme. Hence, we exploit only these rules for matching of Web service input and outputs and future analysis of annotated Web service documents.

V. Related Work

Research in knowledge acquisition methods for the purpose of (semi) automatic annotation of Web service interfaces can be roughly divided into the following categories: 1) methods which utilize linguistic resources and NLP techniques; 2) methods which rely on machine learning techniques to classify or cluster similar (or related) names and generalize them to their ontological concepts; 3) the approaches that combine the aforementioned methods with online resources.

The works in the first category, exemplified by [11][12], capture relationships between WSDL elements and transform them into ontological concepts and relationships, typically using simple lexico-syntactic patterns and WordNet dictionary. They exploit textual descriptions of Web services to improve or enrich the quality of the generated ontology.

However, the machine learning techniques, belonging to the second category, exploit NLP techniques only to normalize their input datasets and then continue with machine learning techniques. In this light, Heβ et al. [13] developed a classifier system which initially needs to be trained in order to generalize (semantic of training data) and predict semantic labels for (similar) unseen Web services. As we target a large repository of absolutely non-annotated ad-hoc Web services from different domains, applicability of such techniques is not clear. In contrast the clustering methods such as the one proposed by Dong et al. [14], which relies on the co-occurrence of parameter names as an heuristic for identifying ontological concepts, are more appealing since they do not require any training dataset.

Finally. the approaches in the third category exploit Web resources (such as search engines) as sources for knowledge acquisition and augment the Web results with machine learning and /or NLP techniques. In this direction, Segev and Sheng [9] combined TF/IDF measures with Web search results to discover proper domain concepts representing WSDL elements and then validated it using textual documentations in WSDL documents. The main obstacle of utilization of some of the aforementioned work, namely [9][11][12], is the fact that around 94% of WSDL documents in our collection lack any textual documentation [6], hence, utilization of those approaches is not feasible in our case.

VI. Conclusions and Future Work

In this work, we proposed and evaluated a scheme suitable for automated annotation of a large set of Web service corpus. The experimental results revealed that utilization of certain rules of the proposed matching scheme can be used for bootstrapping large-scale annotation of Web service interfaces in a semi-automatic way. However, we still need to enhance the quality of generated ontologies by potentially incorporating external resources such as domain ontologies or Web resources to the annotation scheme. Since the number of available Web services and therefore our corpus is expected to increase, keeping the generated reference ontology updated will be challenging. Hence, it is not cost-effective to incorporate the entire dataset for ontology learning purpose. As part of our future work, we are aiming to discover and annotate only a subset of whole dataset which its network properties exhibit closer approximation to the already observed properties in the Web service networks [3][4][5].

Acknowledgment

This research is partly funded by ERDF via the Software Technology and Applications Competence Centre (STACC) and ESF via the DoRa program.

References

[1] P. Küngas, and M. Dumas,"Cost-effective semantic annotation of XML schemas and Web service interfaces," Proc. SCC-09, 2009, pp. 372-379.

[2] S. Bergamaschi, and S. Sorrentino, "Semi-automatic compound nouns annotation for data integration systems," in Proc. SEDB-09, 2009, pp. 221-228.

[3] S.-C. Oh, D. Lee, and S. R. T. Kumara, "Effective Web service composition in diverse and large-scale service networks," IEEE Transactions on Services Computing, vol. 1, no. 1, 2008, pp. 15-32.

[4] Y. Cui, S. Kumara, J. Jung-Woon Yoo, and F. Cavdur, "Large-scale network decomposition and mathematical programming based Web service composition," in Proc. CEC-09, 2009, pp. 511-514.

[5] H. Kil, S.-C. Oh, E. Elmacioglu, W. Nam, and D. Lee, "Graph theoretic topological analysis of web service networks," Journal of World Wide Web. vol.2, no.13, 2009, pp. 321-243.

[6] S. Mokarizadeh, P. Küngas, and M. Matskin, "Ontology learning for cost-effective large-scale semantic annotation of XML schemas and Web service interfaces," in Proc. EKAW-10, 2010, pp. 401-410.

[7] J. Euzenat, and P. Shvaiko, Ontology Matching. Springer, 2007.

[8] W. Hu, and Y. Qu, "Falcon-AO: A practical ontology matching system," Journal of Web Semantics, vol.6, no.3, 2008, pp. 237-239.

[9] A. Segev, and Q. Z. Sheng, "Bootstraping ontologies for Web services," IEEE Transactions on Service Computing, preprint, 2010.

[10] A. Gangemi, N. Guarino, C. Masolo, A. Oltramari, and L. Schneider, "Sweetening ontologies with DOLCE" Proc. EKAW'02, 2002, pp.166-181.

[11] M. Sabou, C. Wroe, C. Goble, and G. Mishne, "Learning domain ontologies for Web service descriptions: An experiment in bioinformatics" Proc.14th Int.Conf.World Wide Web, 2005, pp.190-198.

[12] H. Guo et al., "Learning ontologies to improve the quality of automatic Web service matching," In Proc. ICWS-07, pp. 118-125.

[13] A. Heß, and N. Kushmerick, "Learning to attach semantic metadata to Web services," in Proc. ISWC-03, 2003, pp. 258-273.

[14] X. Dong, A. Halevy, J. Madhavan, E. Nemes, and J. Zhang, "Similarity search for web services, " in Proc. VLDB-04, 2004, pp. 372-383.

[15] G. A. Miller, "WordNet: A Lexical Database for English," Communications of the ACM, vol. 38, no. 11, 1995, pp. 39-41.

A Comparison and Analysis of Some Ontology Visualization Tools

Simon Suigen Guo
Energy Informatics Laboratory, Faculty of Engineering and Applied Science
University of Regina
Regina, Saskatchewan, S4S 0A2, Canada
Email: guosi111@uregina.ca

Christine W. Chan
Energy Informatics Laboratory, Faculty of Engineering and Applied Science
University of Regina
Regina, Saskatchewan, S4S 0A2, Canada
Email: chancw@uregina.ca

***Abstract*—This paper presents an analysis and comparison of three existing ontology visualization tools with a new ontology visualization tool developed at the Energy Informatics Laboratory of University of Regina, Canada. The new tool is called Onto3DViz, which is designed as a knowledge engineering support tool for ontology visualization. It aims to address the deficiency that existing tools have in their lack of support for dynamic knowledge visualization. The Onto3DViz is a tool that supports a new approach of visualization in that it supports: (1) dynamic knowledge visualization; and (2) complex ontology visualization in 3-dimensional (3D) computer graphics. This paper also discusses the strengths and weaknesses of the four visualization tool when they are measured against a set of assessment criteria.**

Keywords – Ontology, Ontological Visualization, Knowledge Engineering, 2D Graphics, 3D Graphics

I. INRODUCTION

The Semantic Web (SW) is the next evolutionary step for the web; it aims to provide semantics to data on the Web, enabling computers to more easily share and perform problem solving on the data [1]. SW technology can enable sharing and re-use of knowledge between Knowledge-Based Systems (KBS) in a distributed and heterogeneous environment. Ontologies can enable data on the Web to be structured semantically for machine processing, as well as to become the basis for building KBSs. An ontology is an "explicit specification of a shared conceptualization" [2]. The main benefit of an ontology is that it supports sharing and re-use of application domain knowledge across distributed and heterogeneous software systems [3]. However, the construction of ontologies is time-consuming and costly.

Software tools can help reduce the effort required to construct ontologies and knowledge bases. The general objective of our work is to provide software tool support for ontological engineering. In this paper we present our work in developing a visualization tool for ontological engineering which generates a 3D visualized image of an ontology model. The visualized 3D model can support Knowledge Engineering in two ways. First, it supports better communication between the knowledge engineer and expert, who often are from diverse backgrounds and do not share the same language or vocabulary. A visualized model does not rely on textual descriptions and can avoid difficulties in communication that arise due to language differences. Secondly, a visualized model summarizes the elicited knowledge elements in a 3D model and can support expert validation of the model.

Onto3DViz is an ontology visualization tool, which utilizes 3D graphics to visualize a knowledge model consisting of both static and dynamic knowledge components. The tool has been constructed based on the theoretical framework of Inferential Modeling Technique (IMT) [4], hence Onto3DViz simultaneously supports a developed knowledge engineering method and ontology visualization. The tool aims to address deficiencies of some existing and surveyed visualization tools, which support neither a knowledge engineering method nor dynamic knowledge modeling. Onto3DViz employs 3D graphics to represent knowledge concepts and relationships among concepts. Compared to a 2D graphic representation, 3D graphics can support visualizations consisting of complex and related information in a clear visual model that can be manipulated for flexible views. Onto3DViz was developed in Java, it accepts an input ontology that is represented in an XML-based language document, and it generates a 3D visual ontology model as output. Hence, Onto3DViz is theoretically not limited to any machine platform. This tool has been applied to visualize application ontologies. Figures 1 and 2 show an application ontology visualized in the Onto3DViz. The representations of the knowledge elements are described below:

- Class: Sphere
- Objective: Cylinder
- Instance: Box
- Task: Cone
- Relationship between concepts : Line

More details of Onto3DViz and its applications can be found in [5] and [6].

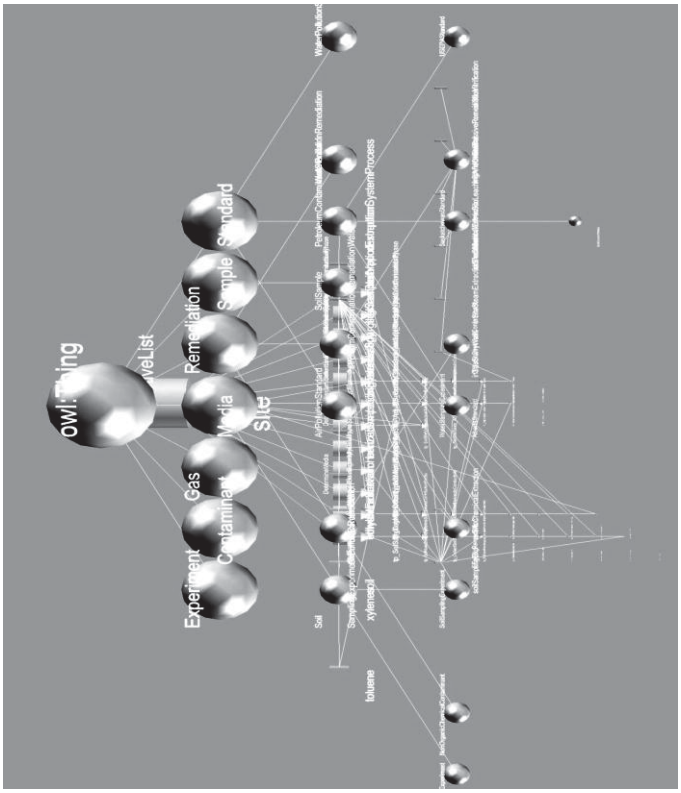


Fig.1 3D visualized model of the application ontology

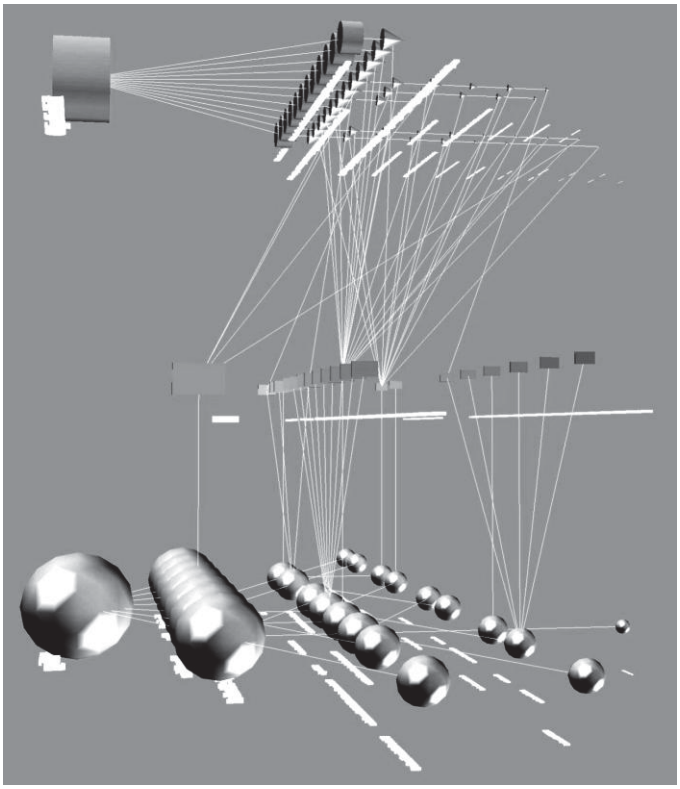


Fig. 2 The side view of the visualized application ontology

The objective of this paper is to present a comparison of Onto3DViz with three other existing ontology visualization tools: OWLViz [7], Jambalaya [8] and OntoSphere [9] against a set of selected criteria. Each of these four tools will be examined and evaluated from the knowledge engineering perspective. The advantages and disadvantages of these tools will be compared and discussed.

This paper is organized as follows. Section II presents relevant background about the field of ontology visualization tools. Section III describes evaluation of the tool. Section IV presents the result of the evaluation, and discusses some strengths and weaknesses of each tool. Section V provides some concluding thoughts and discusses directions for future works.

II. BACKGROUND

Ontology visualization tools can help to improve human understanding of the conceptual model by illustrating the model in visualized graphics. Since visualization can provide an instant picture of the knowledge structure to the knowledge engineer or domain expert, and avoid the need for tedious examination of the detail relationships among concepts expressed in an ontology language, these tools enhance the knowledge acquisition and ontology authoring process. In other words, the user is able to quickly identify the concepts and their relationships with other concepts from the visualized model instead of having to comprehend an ontology from a textual document.

A. Ontology Visualization Tools

Among the ontology visualization tools that have been developed, OWLViz, Jambalaya and OntoSphere are three popular ontology visualization tools that invoked 2D or 3D computer graphic technology. A summary of these tools is given as follows:

OWLViz is a graph type visualization plug-in for the Protégé [10] platform. It can visualize class hierarchies in a graph instead of the default tree view in Protégé. OWLViz supports user interactions, such as zooming and searching in the graph. The class hierarchy is displayed from left to right in a horizontal manner and the class at the most left is the root class.

Jambalaya is a Protégé plug-in that implements the Simple Hierarchical Multi-Perspective (SHriMP) [11] visualization technique. SHriMP represents a hierarchical structure of information as a set of nested graphs. It was originally designed for assisting programmers to visualize object-oriented software programs, and Jambalaya adapted it for representing a domain ontology. Jambalaya provides several viewing perspectives for the ontology model, thereby enhancing user browsing, exploring and interacting with 2-dimensional (2D) ontology visualization. Additionally, Jambalaya provides operations such as browsing, filtering and searching, so that the user can examine the knowledge elements of an application ontology at different levels of abstraction and details.

OntoSphere has been implemented as a Protégé plug in. By employing 3D graphics for visualization, OntoSphere extends the volume of space available for visualizing overcrowded concepts, and it is the only existing ontology visualization tool that adopts the 3D view, which is natural for humans. A main advantage of a 3D representation is that it allows users to manipulate the visualized knowledge elements of the application ontology by means of the actions of zooming, rotating and translating. Through physical manipulation of the concepts, the user can better understand a complex ontology.

B. *Inferential Modeling Technique*

The Inferential Modeling Technique is a knowledge engineering method that supports developing a domain ontology consisting of both static and dynamic knowledge of the problem domain. Static knowledge consists of concepts, attributes, individuals and the relationships among them; dynamic knowledge includes objectives, tasks, and relationships among objectives and tasks. Static and dynamic knowledge are intertwined in that a task is a process that manipulates static knowledge to achieve an objective. The details of this modeling technique can be found in [4].

III. EVALUATION METHODOLOGY

An analysis and comparison of existing visualization tools with Onto3DViz is given in this section. Although comprehensive surveys had been conducted previously, e.g. [12], these works do not include tools that support dynamic knowledge modeling. Also, previous surveys usually emphasize the graphic visualization methodology aspect, and ignore consideration of the ontological engineering methodologies implicit in the tools.

The objective of this analysis is to compare Onto3DViz with several exsisting ontology visualization tools according to a set of criteria selected for assessing the quality of ontology visualization generated by the tools. The tools chosen for comparison include OWLViz, Jambalaya, and OntoSphere, which have been introduced in section II, and Onto3DViz. The criteria for evaluation of these from tools are described below:

1. *Representation of Concept Hierarchy* – is the representation of the hierarchical information among concepts, such as inheritance between classes.
2. *Number of viewing perspectives* – refers to the different graphic layouts that each visualization tool supports. Some tools have only one layout, while others have multiple layouts.
3. *Maximum levels of sub-concepts to be rendered (Max Level)* – is the number of descendant levels that can be shown in the hierarchical representation.
4. *Optimization of rendered space* – measures if the tool can minimize graphic space used in the visualization.
5. *Zoomable* – is a feature that supports magnification of the visualization.
6. *Concept searching* – is a feature that supports searching for a concept by entering its name.
7. *Concept filtering* – is a feature that hides or displays only the selected concept
8. *Concept editing* – is a feature that supports real-time ontology editing of the visualized model
9. *Focusing* – is a feature that supports visualization of only a single concept or a branch of related concepts
10. *Static knowledge visualization* – static knowledge support
11. *Dynamic knowledge visualization* – dynamic knowledge support
12. *Additional visual aids* (such as coloring, sizing) – are additional features that enhance the visualized model.
13. *2D VS. 3D* – indicates if the visualization tool is developed using 2D or 3D graphic technology.
14. *System requirements* (such as ontology modeling platform) – refer to the pre-requisite environment for executing the tool

Some of these criteria are standard factors in assessing general ontology visualization; they include Concept Hierarchy, space, zoomable, concept searching and filtering, focusing [12]. Since IMT includes the concepts of static and dynamic knowledge visualization, hence, the static knowledge visualization and dynamic knowledge visualization are considered in this analysis. The criteria of max level and additional visual aids are used to measure the quality of ontology visualization. The factor of system requirements aims to evaluate the extent to which the tools depend on other systems. The assessment of the four tools against this set of criteria is presented in the next section.

IV. COMPARISON AND DISCUSSION

A summary of comparison of Onto3DViz against the other tools of OWLViz, Jambalaya and OntoSphere according to each criterion is presented in TABLE I, and discussion of the comparison follows.

TABLE I. Evaluation of Ontology Visualization Tools

	OWLViz	Jambalaya	OntoSphere	Onto3DViz
Representation of Concept Hierarchy	Hierarchical tree layout	Rich support for various layouts, like tree, map, etc	Concept hierarchy is not well represented	3D hierarchical tree layout
Number of viewing perspectives	1 fixed layout	Provides 5 graphic layouts	Provides four 3D scenes	1 dynamic 3D view
Max level	Maximum at 10 descendant levels	unlimited	unlimited	unlimited
Space	Limited optimization, cannot fit complex model in a screen	Good optimization, but labels are overlapping	Good optimization, no overlaps	Limited optimization, labels and nodes are overlapped in complex ontology
Zoomable	Not supported	Limited zoom	Flexible zoom	Flexible zoom
Concept searching	Supported	Supported	Supported	Not supported
Concept filtering	Supported by user defining levels of concepts	Supported by collapsing the visual nodes	Supported by user changing 3D perspective	Supported by hiding static or dynamic knowledge only
Concept editing	Supported through Protégé editor	Supported through Protégé editor	Supported through Protégé editor	Not supported
Focusing	Not supported	Support by changing graphic layouts	Support by user changing 3D perspective	Support by user manipulating 3D model
Static knowledge support	Supported	Supported	Supported	Supported
Dynamic knowledge support	Not supported	Not supported	Not supported	supported
Additional visual aids	Limited	Limited	Rich	Rich
2D VS 3D	2D	2D	3D	3D
System requirements	Requires Protégé and GraphViz plug-in	Requires Protégé	Requires Protégé and Java 3D	Requires Java 3D

OWLViz can only do basic ontology visualization in a root tree model generated from left to right on the computer screen. It combines ontology editing with visualization. Hence, the user can edit, search and filter ontology classes in OWLViz, without it being necessary to shift back to the class editor in Protégé. However, OWLViz does not support graphic animation, multi-perspectives, visualization on instance and other visual aids. It has only one view in its visualization. It is difficult for the user to navigate among the classes because it does not optimize on rendering space for the graphical objects within the size of the computer window. Also it does not support either zoom-in or zoom-out. OWLViz cannot render an ontology that has ten or more levels of sub-classes, and it does not support dynamic knowledge visualization. In order to visualize an ontology in OWLViz, a plug-in software called GraphViz [13] is required in addition to the Protégé platform. In general, OWLViz is only suitable for visualizing a simple ontology.

Jambalaya provides rich support for ontology visualization. It overcomes the limitations of 2D graphic in terms of not having enough space for rendering a large amount of ontological information, because it employs multi-perspectives viewing and animation graphics. Jambalaya also supports ontology editing and searching within the visualization screen. The user can select and edit concepts from the visualized model without switching back to the Protégé class editor. It can visualize a large scale ontology.

Although Jambalaya has optimized the space to fit all the ontology concepts into the size of a computer screen, overcrowded concepts may have their labels being overlapped with each other. In addition, the visual nodes are too small and too close to each other in some views, which make it difficult for the user to select the nodes. In terms of the *zoomable* feature, Jambalaya only has limited function because it can support only either zoom in or out, and provide either a full view or focus on one concept. The user is not able to use this zoom function to explore a segment of the visualized ontology. Jambalaya does not support *additional visual aids*, and all the visualized concepts in Jambalaya are of one size and color. Hence it is hard to see the *Representation of the Concept Hierarchy*. Jambalaya does not support *dynamic knowledge visualization*; it does not support the versions of Protégé newer than 4.0 because Protégé changed the plug-in architecture since version 4.0. Jambalaya is much more advanced than OWLViz in terms of functionalities. Though Jambalaya can be applied to explore a complex ontology, the lack of visualizing space in a 2D graphics environment is the problem that causes visual objects generated to be overcrowded and labels to overlap with each other.

OntoSphere employs 3D graphics, and is able to generate many perspectives, such as rotating, zooming, and translating the 3D model. It also supports multiple scenes, and the user can switch among four different scenes in OntoSphere to explore the 3D visualized ontology model. For example, the Concept Focus Scene provides the option that OntoSphere only visualizes one concept instead of the entire ontology. OntoSphere can support a large scale ontology, and ontology concept editing and searching. It also provides some visual aids, such as sizing and coloring of the visual objects for distinguishing different concepts in the visualized ontology model.

However, OntoSphere has a limitation when it is used to render hierarchies in the Root Focus Scene. The technique that OntoSphere uses to visualize the entire hierarchical model differs from the general top-down or left-right approaches. Instead, it visualizes all the domain concepts that around a big hyperbolic sphere, which represents the root concept. The problem with this technique is that the inheritance relationship and relationships among concepts cannot be shown. Moreover, OntoSphere does not support dynamic knowledge visualization. Since Protégé has upgraded its plug-in architecture after version 4.0 and OntoSphere did not upgrade accordingly, it is not compatible with newer versions of Protégé after 4.0.

Onto3DViz can successfully render the complex domain concepts and relationships among concepts in a 3D model, thereby creating a visualized knowledge model first formulated with the IMT during the knowledge acquisition process. The user can easily examine and manipulate the 3D model by performing the operations of zooming, rotating, and translating using the computer mouse or keyboard. The 3D model clearly shows the hierarchical structures of the static and dynamic knowledge. For dynamic knowledge, the 3D model supports representing tasks related to each objective by arranging the tasks in the correct prioritized order. Onto3DViz also supports visualizing the relationships between the static and dynamic knowledge models.

Several weaknesses of the current version of Onto3DViz are also noted. First, when the visual objects are too close to each other, the objects and labels become overlapped, which makes it difficult for users to clearly see details of the model. Secondly, Onto3DViz does not currently support a search function. If the user wants to retrieve a particular concept, he or she has to manually examine the 3D model to find the corresponding visual object. This can be difficult if the ontology is complex or consists of a large number of concepts. Thirdly, if an application ontology consists of too many hierarchical levels, it is difficult to see the lower level nodes because they are too small to show in Onto3DViz. This is because the sizes of the visual nodes are reduced quickly by successive applications of the scaling technique. The detail of the scaling technique is described in [5].

In comparison to OWLViz, Jambalaya and OntoSphere, Onto3DViz provides a new solution for dynamic knowledge visualization; this is a feature that existing tools do not support. The implementation reflects an approach which ensures the ontology visualization tool is independent of the ontology modeling platform. The weakness in adopting this approach is that Onto3DViz currently cannot support ontology editing and searching. The benefit of the approach is that Onto3DViz is not restricted by the platform; it can visualize an ontology model as long as the user inputs to the tool a valid OWL or XML file that represents the ontology.

Onto3DViz is designed as a new solution for dynamic knowledge visualization. Due to the complexities of representing both static and dynamic knowledge, it is necessary for Onto3DViz to employ 3D graphics so as to access a larger volume of space for rendering the complex information and relationships. The advantage of 3D technology over 2D technology is still controversial. According to [14], 3D representation only marginally improves upon the screen space while increasing complexity of the interaction. [15] shows that users of a 3D graphic application need to have more computer experience than users of a 2D graphic application. [16] points out that navigation in a 3D space can be difficult for a novice user. In fact, our study has revealed that comprehensive 2D visualization tools like Jambalaya also require a certain level of computer skills on the users' part for manipulating visualized models. Compared to OntoSphere and Onto3DViz, Jambalaya requires the user to perform complex steps for generating effective ontology visualizations. Moreover, the advantage of the 3D image is not shown to its full potential in the dominant 2D display technology we currently have, because the 2D display only shows a 2D projection of the 3D ontology model and the depth of the 3D graphic image cannot be adequately explored. With increasing popularity of the 3D display technology, the true benefits of 3D graphics will become more evident when the user can navigate the ontology model in a real 3D environment.

V. CONCLUSIONS AND FUTURE WORKS

This paper presents an analysis and comparison of Onto3DViz, OWLViz, Jambalaya and OntoSphere against a set of criteria. The advantages and disadvantages of Onto3DViz as compared to the other ontology visualization tools have been discussed. Onto3DViz was designed as a new ontology visualization tool the supports: (1) visualization of an ontology with static and dynamic knowledge; and (2) visualizing a complex application ontology in a new 3D computer graphic architecture. In the future, the new tool of Onto3DViz can be enhanced by implementing concept searching and editing functions, so that the user can quickly identify and edit a concept in the 3D ontology model. Moreover, assigning colors to the visual objects can also assist the user in identifying concepts. For instance, the class objects are assigned the blue colour and instance objects are assigned the yellow colour. This new feature would ensure that the different types of concepts are easily distinguishable in a visualized ontology. A coloring scheme can be added to the tree structure of generated visual nodes so that the root is the darkest shade and the descendent nodes from the root will be progressively lighter in color, to indicate the different layers of the tree structure. As well, more user controls can be implemented which will enhance user-friendliness of the tool. Integrating or interfacing Onto3DViz with ontology modeling platform, such as Protégé [10] is also possible in the future, because Onto3DViz was also develop in Java language.

ACKNOWLEDGEMENT

We are grateful for the generous support of Research Grants from Natural Sciences and Engineering Research Council (NSERC) and the Canada Research Chair Program to the first author.

REFERENCES

[1] T. Berners-Lee, J. Hendler, & O. Lassila, "The Semantic Web", *Scientific American*, May 2001, (pp. 34-43).

[2] T. Gruber, "Towards Principles for the Design of Ontologies Used for Knowledge Sharing", In *Guarino & Poli (eds): Formal Ontology in Conceptual Analysis & Knowledge Representation*, Padova, Italy: Kluwer, 1993.

[3] N. Guarino, "Formal Ontology in Information Systems". In *Guarino (ed) 1st International Conference on Formal Ontology in Information Systems (FOIS'98)*, Trento, Italy, IOS Press, Amsterdam, 1998, (pp. 3-15).

[4] C. W. Chan, "From Knowledge Modeling to Ontology Construction", *Int. Journal of Software Engineering and Knowledge Engineering*, 14(6), Dec 2004, (pp. 603-624).

[5] S. Guo & C.W. Chan, A tool for ontology visualization in 3D graphics: Onto3DViz. *The Proceedings of the 23rd Canadian Conference on Electrical and Computer Engineering*. Calgary, AB, May 2010.

[6] S. Guo & C.W. Chan, "Application of a tool for ontology visualization", *The Proceedings of the 9th IEEE International Conference on Cognitive Informatics*, Beijing, July 2010, (pp. 471-476).

[7] M. Horridge, *OWLViz*. Retrieved March 10, 2011, from CO-ODE: http://www.co-ode.org/downloads/owlviz

[8] M. Storey, M. Musen, J. Silva, C. Best, N. Ernst, R. Fergerson, & N. Noy, "Jambalaya: an interactive environment for exploring ontologies", *Workshop on Interactive Tools for Knowledge Capture (K-CAP-2001)*, Victoria B.C., Canada, (p. 239).

[9] A, Bosca, D. Bonino, & P. Pellegrino, "OntoSphere: more than a 3D ontology visualization tool" *Proceedings of SWAP 2005, the 2nd Italian Semantic Web Workshop.* Trento, Italy: CEUR Workshop Proceedings.

[10] Protégé. Retrieved March 10, 2011, from Protégé: http://protege.stanford.edu

[11] J. Wu, & M. Storey, "A Multi-Perspective Software Visualization Environment", *conference of the Centre for Advanced Studies on Collaborative Research 2000*, Mississauga, Ontario, Canada, (pp. 31-40).

[12] A. Katifori, C. Halatsis, G. Lepouras, C. Vassilakis, & E. Giannopoulou, "Ontology Visualization Methos - A Survey", *ACM Computing Surveys (CSUR), Volume 39 , Issue 4 (2007), Article No.: 10.*

[13] AT&T Research Lab. *Graphviz - Graph Visualization Software*. Retrieved March 10, 2011, from Graphviz – Graph Visualization Software: http://www.graphviz.org

[14] C. Plaisant, J. Grosjean, & B. Bederson, SpaceTree: Supporting exploration in large node link tree, design evolution and empirical evaluation. Proceedings of IEEE Symposium on Information Visualization, (pp. 57–64). Boston.

[15] M. Sebrechts, J. Cugini, S. Laskowski, J. Vasilakis, & M. Miller, "Visualization of search results: a comparative evaluation of text, 2D, and 3D interfaces", *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, (pp. 3-10)

[16] A. Cockburn, & B. McKenzie, "Evaluating the Effectiveness of Spatial Memory in 2D and 3D Physical and Virtual Environments", Proceedings *of ACM CHI'2002 Conference on Human Factors in Computing Systems. Minneapolis, Minnesota, 20--25 April 2002*, (pp. 203—210).

Knowledge management in next generation networks

Samir Atitallah
Computer sciences department
Ecole Polytechnique de Lausanne
Lausanne, Suisse
samir.atitallah@epfl.ch

Maria Sokhn
Computer sciences department
Ecole dingénieurs et d'architectes de Fribourg
Fribourg, Suisse
maria.sokhn@hefr.ch

Omar Abou Khaled
Computer sciences department
Ecole d'ingénieurs et d'architectes de Fribourg
Fribourg, Suisse
omar.aboukhaled@hefr.ch

Elena Mugellini
Computer sciences department
Ecole d'ingénieurs et d'architectes de Fribourg
Fribourg, Suisse
elena.mugellini@hefr.ch

Abstract— **As the amount of information grows, the major issue is to access these resources, distributed over the network. Since a large number of the data that we might want to retrieve is not text-based, the only way to search within this type of information is by associating metadata. To reach this goal, some projects aim to optimize the annotation scheme in order to have a standard unified ontology model to describe the data and make use of this model to search for information by concept, not only by keywords. The main goal of the paper is to design and implement the information retrieval over a peer-to-peer network to perform high level annotation of scientific talk recordings, offer granular search facilities and complex queries, and enhance the knowledge management of the recordings.**

Keywords: Knowledge management, peer-to-peer architecture, ontology, information retrieval

I. INTRODUCTION

Distribution technologies are constantly expanding. They have faculties unexplored in centralized systems. Their integration with annex systems such as research and registration services can lead to distributed solutions that facilitate and greatly improve the collection of information. In the field of technology research, it is often limited to research in "text mode" which causes a considerable loss of information. Indeed, we know that the information consists mainly of audio-visual content that is searchable only by the associated metadata and not its actual content. This restriction has raised the question of how search and registration technologies can tap into the potential pool of information. This problem has grown to develop technologies that aim to break barriers and allow search engines to index and retrieve information based on content. These search techniques of a new generation would effectively and efficiently provide relevant information in the presence of the exponential growth of distributed multimedia data volumes. Our approach focuses on developing a scalable solution that meets the needs of massively distributed data produced by a variety of services. The magnitude of the problem can be gauged by the fact that almost everything we see, read, listen, write will soon be available for computerized systems.

This paper presents a service based on a P2P architecture that aims at breaking this technological barrier by defining a distributed P2P architecture that allows search of audio-visual content using the paradigm of query by example. Considering that a video is much more explicit than the text, we attempt to introduce information about the content. The combination of research advises with optional metadata annotations of users and the context of social networking has a capacity of research results more accurate and complete. The P2P architecture is based on the ontological model for recording videoconferences, HELO [1] and its uses within the Framework CALIMERA [2]. It models the information and knowledge transmitted in a life cycle of the conference. Using HELO we aim to bridge the semantic gap by providing opportunities to make annotations to the records of granular scientific discourse, and to improve information retrieval and display of records.

II. STATE OF THE ART

The semantic approach to existing Web resources is one of the major challenges for building the Semantic Web. In the state of the art of semantic web tools, we find many that can be useful. But rare are those that use systems that exploit the power of millions of users every day seeking information on the web. Today, many applications of file sharing peer-to-peer exist. These applications do not have a major characteristic: the semantic search. Although most applications of peer-to-peer sharing offer a keyword search based on indexing data, few are actually accessing the data content and use powerful annotation systems that provide detailed information on data. In other words, there are few projects [3; 4; 5] that offer the user the opportunity to describe his desire to search in the most intuitive words and allow an application to extract the necessary semantic and make it understandable to the machine that will perform the recovery action that best suits the user. This absence is due to lack of use of metadata information generated by annotations and ontology models that describe the data content rather than a general description of the file.

Many publications and projects have been designed to manage multimedia information retrieval using a P2P architecture[6; 7; 8]. Some as VIKEF were developed in order to provide support for advanced semantic information, content

production and knowledge acquisition, processing, annotation, sharing and use by empowering information and knowledge environments for scientific and commercial communities. Matterhorn, meanwhile, produce recordings of lectures, manages existing video, provide distribution channels and provide user interfaces for students with educational videos. However, those two projects do not use the possibilities of a distributed system such as the P2P architecture. Indeed, there are projects like PHAROS [9] and Sapir [10] that use the power of P2P to allow the operation of audio-visual metadata by different peer of the architecture. However these projects do not rely on ontology dedicated. These ontologies lack of expressivity needed to model multimedia data generated by scientific events such as video recorded lectures. To resolve this problem, we propose a new model named conference HELO. HELO models the knowledge imparted in a life cycle of a scientific conference and has the advantage of being integrated into a framework designed to manage information about the conference and its recovery. This new ontology associated with a P2P distributed architecture provides an environment conducive to the operation of all the information in a conference through various peer architecture. This framework, named CALIMERA, is introduced in the next section.

III. LOGICAL ARCHITECTURE : CALIMERA

The evolution of the web in the last decades has created the need for new requirements towards intelligent information retrieval capabilities and advanced user interfaces. Nowadays, effective retrieval and usage of multimedia resources have to deal with the issues of creating efficient indexes, developing retrieval tools and improving user oriented visualization interfaces. To that end we put forward an integrated framework named CALIMERA. The framework is based on a High-level modEL for cOnference (HELO) and aims at enhancing the information management, retrieval and visualization of recorded talks of scientific conferences. This section presents the conference model and its uses within the framework: performing high level annotation of scientific talk recordings, offering granular search facilities and complex queries, and enhancing the knowledge visualization of the recordings. As a proof-of-concept we present the prototypes that have been implemented.

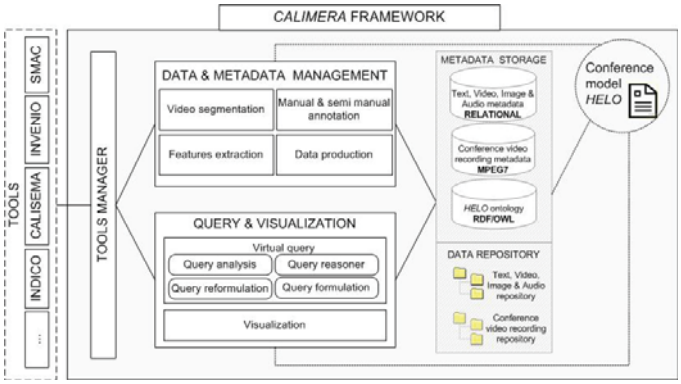


Figure 1. CALIMERA architecture global view

Figure 1 outlines the global view of the framework that is composed of the following modules:

- **Tools manager**: CALIMERA is a tool independent framework. The tool manager allows any user to integrate a tool that may be used for data meta-data management, query and visualization or both.
- **Data and metadata management module** consists of handling the conference high-level information, such as recording talks, segmenting video recordings, annotating video segments, managing the context information of these talks, etc.
- **Data and Metadata storage** integrates existing data and metadata formats such as MPEG-7 which is one of the most widely used standard for multimedia description, RDF and OWL, which are a more semantically oriented standards for multimedia description that integrates high level semantic description.
- **Query and visualization module** queries the data and metadata storage in order to return the video or the set of video sequences of recorded talks the users are seeking for.
- **Conference model**: HELO is a conference model we designed to model the conference high-level information conveyed within a conference life cycle. As cited in the introduction HELO is based on existing ontologies related to conference domain.

To have more information about this logical architecture refers to the *"Conference knowledge modeling for conference-video-recordings querying and visualization"* paper [1].

IV. PHYSICAL ARCHITECTURE : CALIMERA OVER P2P

This chapter will deal with designing a physical architecture that uses a version of the Framework CALIMERA in a distributed way to help expand opportunities and overcome the limitations of information retrieval mentioned above. This p2p architecture matches the logical architecture and allows using independent tools that produce the data and metadata by each user of the p2p network. Thus it offers the possibility to generate that information, which require resources and time cost, by every peer. That information will be accessible by anyone inside or outside the p2p network.

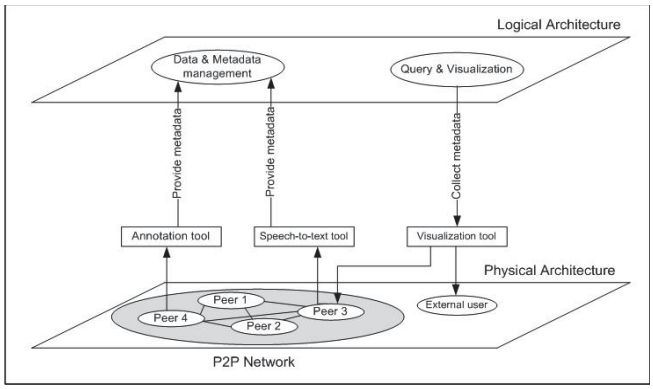


Figure 2. Logical and Physical architecture

The following chapters will address the design of the architecture through the critical inputs that they bring.

A. Data management over P2P networks

The main objective is to develop a file-sharing application for semantic information sharing conference through a network user. The research is based on the model developed ontology conference at the University of Applied Sciences in Fribourg. We also hope that the implementation of the application based on P2P architecture meets four requirements:

- The first one concerns the wide distribution of multimedia data items: each user holds his files at his end device, and geographical distances separate users. With no interest in infrastructure investment, we care for an architecture that could make use of this distribution.
- The second issue, which is vitally related to the first, concerns the dynamicity of the users and of their data. Users must remain free to access and leave the network at any time. They must have all the rights to introduce and remove data files for which they are responsible at any moment. We look for a scalable and dynamically adaptable system.
- The third one is directly related to the retrieval process: since we will be leading a semantic search we are interested to localize data items at any moment during short periods. We wish to discover the location of any item in the network in a reasonable time interval.
- Finally, the fourth issue concerns the retrieval results efficiency and effectiveness: we want to be able to resolve user complex queries. We want a system that processes and retrieves data in a semantically meaningful manner.

Our reflections toward resolving the issues above lead into the choices that we exposed earlier.

The choice of the peer-to-peer architecture eliminated the concerns about distribution. In fact, peer-to-peer takes benefit of this distribution as presented in the state of the art section. It eliminates the need of major infrastructure implementation: users share their simplest resources and contribute in the maintenance of the network. It ensures also that all peers are equal and holding the same privileges.

The choice of the structured decentralized peer-to-peer architecture using the JXTA algorithm addresses the issue of data indexing and network flexibility and scalability. Each peer has the necessary information to correctly localize any data item in the network: each peer performs his own routing calculations and directly addresses responsible nodes. This eliminates the need to dial into central indexes to localize items on the peers. On the other hand, the JXTA algorithm is designed to ensure flexibility and scalability: the distribution of the keys over the nodes adapts as peers get connected or disconnected or when new data items are introduced. Periodic updates of the network are lead by JXTA to recover from unexpected failures such as when a peer leaves the network without notifying its neighbors. By this we maintain a high level of network consistency.

To address the lack of semantic retrievals, we chose to introduce the RDF language as the data description language and the base to formulate queries. RDF triples that describe data items are distributed over the peers by mean of the JXTA keys. Research will be toward finding the triples that match the user's request; the initial results are then RDF triples that mostly describe the user's desire. Those triples will hold information about the file that they describe which by then will be the file that mostly describes the user's desire. File retrieval is then lead in a basic semantically meaningful manner.

B. Metadata management over P2P networks

The architecture is a very generic one. We kept in mind during the design all possible future ameliorations and variations in the network.

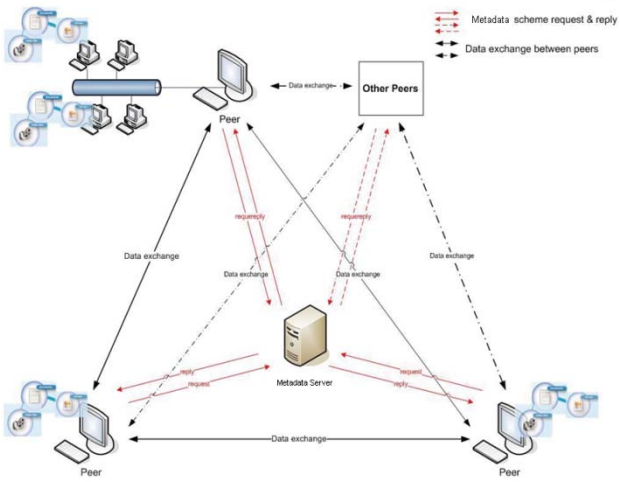


Figure 3. P2P architecture using a routing algorithm

Peers could represent simple end devices or masters of local networks for which they act as access points to the peer network. On another axis, we predicted the presence of the metadata server as necessary, since the users are widely distributed and they might annotate their data differently. They also can produce speech-to-text files metadata that could be reachable by the others peers. Consequently, when a user is willing to share a file[1] over the network, his application contacts the server and downloads the unified metadata scheme proper to the ontology used. The presence of the server is also beneficial if new ontology models are to be implemented over the same network using the same application, in that the network administrator does not have to contact all peers and install the new metadata schemes; they are learned dynamically. It is important to mention that the server has another role: since the server is always present and holding a unique fixed IP, it is responsible for creating the peer to peer network and it offers its URL as the bootstrap URL for the peers that wish to connect to the network. However, the server acts as an ordinary peer on the network when it comes to data

[1] Especially media files, but could also refer to text documents

distribution over the network: it holds no central indexes neither acts like a central storage unit. The JXTA routing algorithm, where each peer calculates his routes locally and performs retrieval actions, handles the discovery of the resources over the network[2]. Data files are kept at peers responsible of them. When a research mechanism points to a file location, a point-to-point connection is established between the requesting peer and the server peer[3] to ensure the channel for data transfer.

As was mentioned earlier in the report, RDF triples, result of the parsing of RDF/XML metadata files, are introduced to the network at three different locations[4] each, by applying a key to an attribute value at each time. Triples are stored at the peers that receive them into files.

V. PROTOTYPE

In this chapter we introduce a prototype that was developed to highlight some features of the architecture implementation. The prototype is based on the JXTA P2P Framework that is described in the next chapter.

A. P2P Networks : JXTA Platform

The information retrieval is performed over a peer-to-peer network with distributed data and a distributed searching technique based on JXTA framework. The Java JXTA platform is a series of classes and methods for managing and transmitting application and control data between JXTA compatible peer platforms. A peer is an identification of a specific instance of JXTA. The concept of peer is similar to the way a computer is named on a LAN, except that the name is not guaranteed to be unique. As a way to make sure that peers are unique, there is a peer ID. The peer ID is generated like other IDs. The concept of peer ID is used because there are multiple methods to reach a peer, so a fixed address or name is not that useful. Also, in the case of computers behind security barriers such as firewalls or NATs, the actual computer name or address is quite useless.

There are different types of peers, divided in two main categories:

- **Edge peers**: these peers are located at the edge of the network; they are the most numerous peers in the network.
- **Super peers**: these peers have specific tasks to accomplish such as keeping a topology of the network, allowing peers behind a firewall to access the network or maintaining a Data Hash Table.

B. Design

There are multiple actions that should be performed using the p2p application. In the first tab of the application interface, the user can annotate the data he wants to share, based on a unified annotation predefined model that includes the type of data and some specific information depending on the type of data being shared. In the second tab, the user can initiate a search using a keyword. The query is sent to the neighboring peers, which will then search for information related to this keyword. The results will appear in the same table, whether they come from an ordinary peer in the network or from an INVENIO[5] server, and the user can then select the document he needs to download. The third tab allows the user to check and manage the files he is sharing with the other peers on the network and the fourth tab allows him to connect to a specific peer knowing its IP address and view the list of peers on the network.

This figure shows the general design of the p2p application

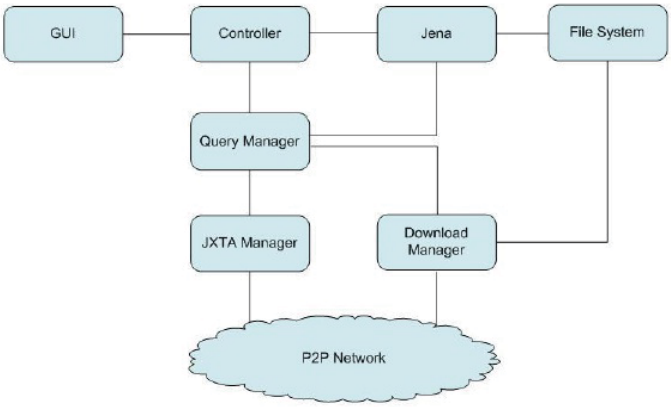


Figure 4. General design of the p2p application

- **GUI (Graphic User Interface)**: it allows the user to interact with the application by sharing and managing the files and sending search queries.
- **Controller**: it collaborates with the user interface to send search and download queries through the query manager and manage the ontology using Jena.
- **Query Manager**: this is the intermediate layer between the application layer (GUI, Controller, Jena) and the network layer (JXTA manager, Download manager)
- **JXTA Manager**: this bloc is responsible of all the connections and the discovery of new peers in the network.
- **Jena**: it is a programming toolkit that manages everything related to the annotation and the ontology.
- **Download Manager:** this bloc is responsible of uploading and downloading the shared files.

1) Communication over the network

[2] Using JXTA successor routing

[3] *Server peer* refers to any peer that holds a data file and contacted by another peer to acquire this file; could be different from the annotation server

[4] JXTA places each *objectID* at the *nodeID* that is equal or higher than it

[5] INVENIO is digital library developed by the CERN, which integrate multimedia data.

In the application developed for this project, we use decentralized network architecture. When the application is launched, the peer connects to the JXTA net peer group then sends advertisements to the peers on the network to exchange some information about their respective sockets. The classes and functions of the JXTA framework provide these functionalities.

2) Nertwork implementation

In the p2p application developed for this project, the class JXTAManager is responsible for connecting the peer to the network. It uses the NetworkManager class provided by JXTA framework and creates a multicast JXTA socket on each peer, allowing it to send and receive requests and responses. The application makes use of the Discovery Service offered by JXTA in order to periodically detect the new peers in the network and connect to them.

3) Metadata annotation and ontology management

When a user wants to share a file, he chooses the type of file he wants to share in the annotation tab of the interface. According to the type of file, the fields to annotate are shown dynamically in the tab. He can decide which fields he wants to fill but he cannot add other fields, which makes him comply with the predefined annotation model.

Once the user clicks on the "submit" button, the local ontology is updated; the name of the new file and its properties are added to the OWL file (p2pontology.owl) that should be located in the same directory as the application itself.

4) Information retrieval and download

The application developed in this project is conceived in a way to allow all types of peers to join the network. In our scenario, we have two types of peers: the "normal" peers that represent the users that might want to share files and search for information and the INVENIO servers that act like super peers. Depending on the type of the peer, the request received is handled differently.

Whether the peer is an INVENIO server or not, the procedure for sending requests and giving back an answer is almost the same. In fact, the results appear in the same table, the only thing that differs is the way the INVENIO server searches in its own information and the peers extract the information from the INVENIO responses.

To decide whether a peer is an edge peer or an INVENIO server, we only have to set the Boolean static attribute INVENIO (in the P2POntology class) to true. The figure 4 shows the UML diagram of the information retrieval class on a server INVENIO.

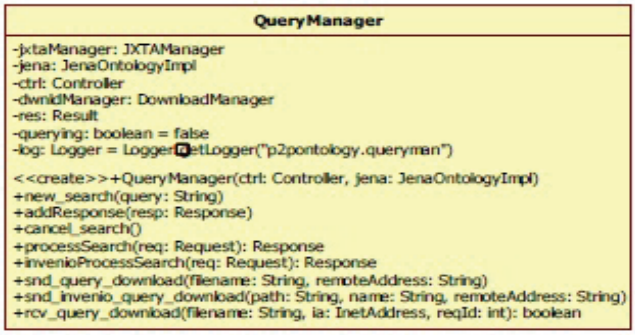


Figure 5. UML diagram of the QueryManager class

5) Jena and ontology search

When the peer is a "normal" peer in the network, using the function processSearch of the QueryManager class performs the search for information related to a keyword. This function uses the Jena Ontology API and is responsible of searching in the local ontology of the peer. Jena can manipulate different programming languages such as RDF and OWL. To find information related to a specific keyword, we search in the OWL file for a match with this keyword and, according to that, we send a response to the peer requesting the information. We will not go into further details about Jena and the ontology search since this is not the aim of this paper.

6) Invenio search

When an INVENIO server receives a search request, it handles the request just like any other peer on the network, which means that it searches for files related to the specific keyword and respond in an XML message, giving some information about the file it finds. The only difference is in the way the INVENIO server searches for the files we need.

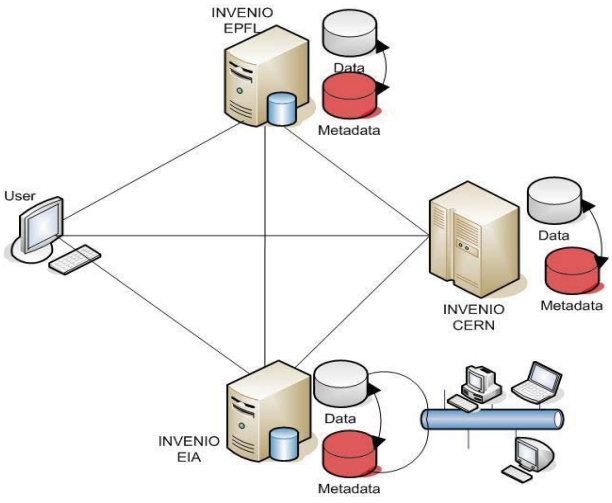


Figure 6. Network topology with different types of peers

We could think of many ways to search within the INVENIO server. Intuitively, the first solution that comes to mind is to create a module that receives the request and uses the search engine of the server to find the results. This solution seems complex for many reasons, one of them being that the programming language used in INVENIO is Python, while the peer-to-peer application is written in Java. This leads us to

think of another simpler solution in order to search within the database of INVENIO. This solution consists of sending an HTTP request to the server and parsing the page to extract the information needed to fill the results table.

When the peer requesting the information receives the response, it can proceed the same way it does with the other types of peers to download the file needed. By right clicking on the selected row, the user can choose to download the page from the server. In this case, since the peer itself might not be connected to the Internet, the user sends a request to the server asking it to provide him with this page. The server then sends another HTTP request with the corresponding web address and returns the page back to the peer.

C. Demonstator

The demonstrator shows how the user can interact with the architecture.

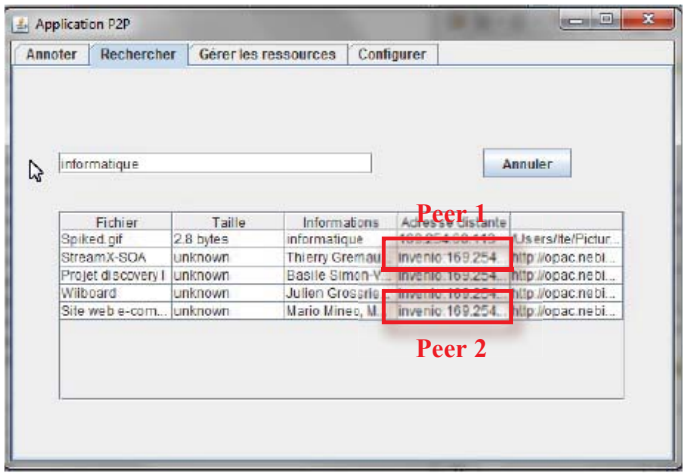


Figure 7. Table showing the results from INVENIO server and other peers

We can recognize the results provided by an INVENIO server by looking at the IP address of the peer that has them (Figure 6). In the column “information”, we can see the names of the authors of the file and in the last column we have a link to this file. Since the peer requesting the information is not necessarily connected to the internet, he can ask the INVENIO server to download the file and send it to him.

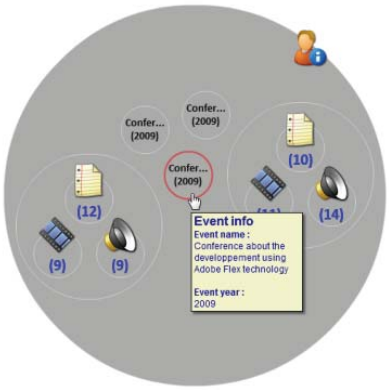


Figure 8. Visualisation of the content of a specific peer

It is also possible to visualize the content of a peer by using a graph interface like the prototype NAVIR [11]. This tool offers the possibility to see the different peers of the network and the data and metadata that each of them provide. It’s a easy way to navigate and access over the p2p network.

VI. Conculsion

In this paper we presented a peer-to-peer architecture that use the scientific conference model HELO (High-level modEL for cOnference) used to enhance the retrieval process of talk recordings (annotation, querying and visualization). We showed that this architecture use the power of P2P to allow the operation of audio-visual metadata by different peer of the architecture.

In the future, our goal is to enhance the retrieval results and to answer the user more efficiently, we could use the Bayesian filter to create users profiles depending on the history of their queries. Each user will have a specific profile as a Professor or student or researcher for example. After profiling by person, people of the same profile could be gathered into a group. This enhances the research also for that users belonging to the same group will be most probably interested with the same topics, this eases and quickens the sharing.

References

[1] Maria Sokhn, Francesco Carrino, Elena Mugellini, Omar Abou Khaled. Conference knowledge modeling for conference-video-recordings querying & visualization. Acm-Medes, Insa Lyon, October, 2009

[2] Y. Liu, D. Zhang, G. Lu, W. Y. Ma. A survey of content-based image retrieval with high-level semantics, (January 2007)

[3] Maria Sokhn. Calimera. http://calimera.project.eia-fr.ch.

[4] D. Zeinalipour-Yazti, Vana Kalogeraki, Dimitrios Gunopulos. Information Retrieval in Peer-to-Peer Systems, Computing in Science and Engineering archive, NJ, USA, July 2004

[5] Maria Sokhn, Elena Mugellini, Omar Abou Khaled, Ahmed Serhrouchni. Conference knowledge modeling for conference-video-recordings querying & visualization. Journal of Multimedia Processing and Technologies Volume 1 Number 2 June 2010

[6] Aberer, K., Klemm, F., Rajman, M., Wu, J.: An Architecture for Peer-to-Peer Information Retrieval. In: SIGIR’04, Workshop on Peer-to-Peer Information Retrieval. (2004)

[7] Cuenca-Acuna, F.M., Peery, C., Martin, R.P., Nguyen, T.D.: PlanetP: Using Gossiping to Build Content Addressable Peer-to-Peer Information Sharing Communities. In: 12th IEEE International Symposium on High Performance Distributed Computing (HPDC-12), IEEE Press (2003) 236–246

[8] Lu, J., Callan, J.: Federated search of text-based digital libraries in hierarchical peer-to-peer networks. In: Advances in Information Retrieval, 27th European Conference on IR Research (ECIR). (2005) 52–66

[9] Platform for searching of audiovisual resources across online spaces. http://www.pharos-audiovisualsearch.eu/

[10] SAPIR extends the power of web searches beyond centralized text and metadata searches to include distributed audio-visual content. http://sapir.eu

[11] Joël Dumoulin, Maria Sokhn, Elena Mugellini, Omar Abou Khaled. Multimedia information browsing and visualization. IEEE VisWeek/InfoViz, Salt Lake City, Utah, USA, October 2010

A Model for Knowledge Retrieval based on Semantic Images

H. Andres Melgar S.* †, Fabiano D. Beppler‡, Roberto C.S. Pacheco† and Jose L. Todesco†
* Sección de Ingeniería Informática, Departamento de Ingeniería,
Pontificia Universidad Católica del Perú, Lima, Perú
†Programa de Pós-Graduação em Engenharia e Gestão do Conhecimento (EGC)
Universidade Federal de Santa Catarina (UFSC), Brasil
‡ Instituto Stela, Florianópolis, Santa Catarina, Brasil

***Abstract*—This paper presents a model that aims to support the visualization of the knowledge stored in digital repositories through semantic images. In this model images contain representations of the real world that are a priori known by the target group, and which have semantic structures that allows identifying the entities of the domain represented in each region. The proposed model is supported by the framework for knowledge visualization proposed by Burkhard and describes the users' interactions with the images. The user through the images can retrieve and view the knowledge related to the entities represented in each region. A prototype was developed to demonstrate the feasibility of the model using images in the biomedical field, the Foundational Model of Anatomy and the Unified Medical Language System as domain knowledge and the Scientific Electronic Library Online database as a document repository. The use of images facilitates the dissemination of knowledge, because these compose user's world view and can easily be related with prior knowledge. Visual representations are processed quickly in the brain and require less effort than the processing of textual information.**

***Index Terms*—knowledge visualization, knowledge retrieval, semantic annotation, ontology.**

I. Introduction

The knowledge dissemination and sharing, are complex tasks for organizations as they often do not know what they have and do not possess systems to efficiently locate and retrieve the knowledge that reside in them [1]. A considerable amount of explicit knowledge is spread across multiple documents within organizations. In many cases, the ability to access efficiently (*i.e.* retrieval) and reuse this knowledge is limited.

Information visualization systems can be used to explore knowledge, to navigate through large volumes of information and to inspect data to make new discoveries [2], [3]. This kind of system is especially useful when people need some information, but they can not translate these needs in key words to search information [4].

In this context, the knowledge visualization field has been researching how the use of visual elements can help the knowledge dissemination process. The use of images that contain representations of the real world, that are part of the users world view, allows that knowledge presented by these images can be easily related to previous knowledge, thus facilitating knowledge dissemination.

This work presents a model that support knowledge visualization by using semantic images. The goal of the proposed model is to use images to construct a support structure for the knowledge visualization process. These images not only contain a visual representation of the real world, but also semantic annotations that help describe its content. The idea is that when a user views an image be able to quickly recognize which regions have associated knowledge and can to retrieve the documents related to the regions by selecting only the region of interest.

This paper is structured as follows: after this introduction, we present the literature review about knowledge visualization and semantic images annotation. Subsequently, the proposed knowledge visualization model is described. In the following sections, we present the material and methods used for model development and discussion. Finally, the last section, we present the conclusions.

II. Literature Review

A. Knowledge Visualization

The knowledge visualization can be defined as the use of visual representations to improve the transfer of knowledge between at least two persons or group of people [5], [6]. Making knowledge visible so that it can be accessed, discussed, valued, appreciated or managed is a long-standing goal in knowledge management. Because of this the knowledge visualization has recently become the focus of attention in academic and business communities [7].

The benefits provided by the visualization seem to be dependent on the fact that it acts as a frame of reference or as a temporary storage area for the processes of human cognition. Visualization enhances the memory of humans to provide an extensive set of work to analyze and reflect, and thus becomes an external facilitator of cognition [4]. According to Ware [8] there are two main theories in psychology that explain how the vision can be effectively used to realize elements and shapes. At low level, the theory of pre-attentional processing explains that some visual elements can be processed quickly. At the highest level, the Gestalt theory describes some principles used by our brain to understand an image.

The knowledge visualization systems are designed to make use of the skills that humans have to process images. The

images are pre-attentive and these are processed before the text [9]. Moreover, the use of images that contain representations of the real world, which are known a priori by the target group and are part of his world view, allows the knowledge presented to abeam of these images can easily be related to prior knowledge of individuals, facilitating learning and memory [5], [9].

Aiming to guide the knowledge visualization application within organizations, Burkhard proposed a framework based on five perspectives that respond to five key questions [5], [6]: What kind of knowledge needs to be viewed? 2) Why should knowledge be visualized? 3) Who is being targeted? 4) In what context should be viewed? and 5) How can knowledge be visualized?

B. Semantic Image Annotation

The annotation can be defined as the process of making explicit the interpretation of the document. Creating metadata by annotating documents is one of the major techniques for putting machine understandable data [10]. Metadata can be attached to a wide range of documents; can be expressed in a wide range of languages and with a wide range of vocabularies [11]; and can be performed manually, automatically or semi-automatic [11]–[13].

Ontologies have been used to annotate documents [10], [11]. Ontological structures may give additional value to semantic annotations allowing inferences and conceptual navigation [10].

The metadata associated with images can be classified as i) independent content metadata, where metadata are related to the image but not described, for example: names of authors, dates, location, etc; ii) content-dependent metadata, where metadata is related to low-level features and/or intermediate level, for example: color, texture, shape; and iii) descriptive content metadata, where metadata is related to semantic content. It has to do with relations of the entities of the image with the real world entities [14].

The descriptive content metadata can be provided at two levels of specificity: i) descriptive content associated with the full image [15], [16] and ii) image segmentation with links to the descriptive content in each segmented region [17], [18].

III. Proposed Model for Knowledge Retrieval

The proposed model was designed in order to facilitate knowledge visualization using semantic images as support structures. The conceptual model described in the figure 1, consists of four components: the semantic images, document repository, knowledge repository and visualization. To the knowledge linked to the images can be recovered efficiently (*i.e.*, semantic search), both images and documents have been previously enriched with semantic content (letters a and b in figure 1) obtained from the knowledge repository (*i.e.*, ontologies, taxonomies, thesauri).

The process starts when the user has to satisfy some information need. The user selects the image (1 in figure 1) from which the knowledge will be visualized. The selection criterion will depend directly of the user information needs,

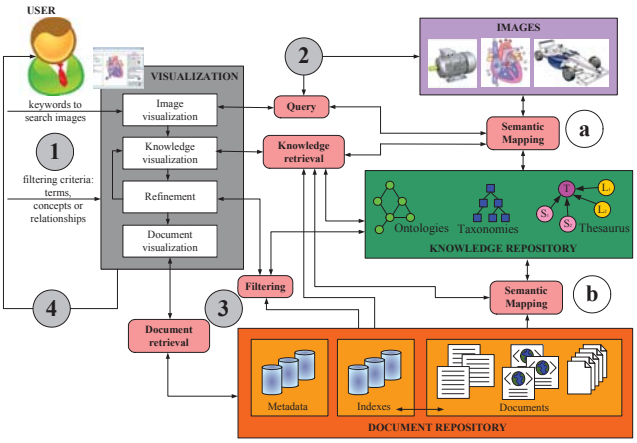


Fig. 1. The proposed model for Knowledge Visualization

taking into consideration the concepts represented in each image. For example, if the user need to visualized knowledge associated with the heart, he must select an image in which the heart is represented.

Once selected the images, the visualization process is executed (2 in figure 1). The visualization process aim to provide users the knowledge stored in document repositories. The results visualization is done over the images, so that users quickly realize the amount of documentation associated with each image region. This is reflected in the image, for example, changing the color of the regions in which concentrates most / least amount of documents or include the number of documents retrieved on each region. From the image, the user can restrict the search space (number 3 in figure 1) and is able to use the metadata provided by document repositories or concepts defined in knowledge artifacts, thus initiating a new visualization process. Next, we briefly describes each component.

A. The Semantic Images

In the model, the images are enriched with semantics structures that allows to represent graphically the domain concepts in which each of its parts is specified in an explicit and formal way. It is defined on three levels (figure 2): the *descriptive level* aims at identifying and describing the visual representations; the *structural level* provides information about the internal structure in order to make explicit the region images; the *semantic level* aims at describing the semantic mappings.

B. Document Repository

In the model, document repositories are characterized by a semantic layer that allows to formalize the information contained in documents. They are defined in four levels: the *descriptive level* aims at identifying the repository; the *metadata level* aims at describing the information structure; the *content level* provides the structures needed to store documents and metadata. The document repository also has a semantic level that aim at linking the semantic content of documents.

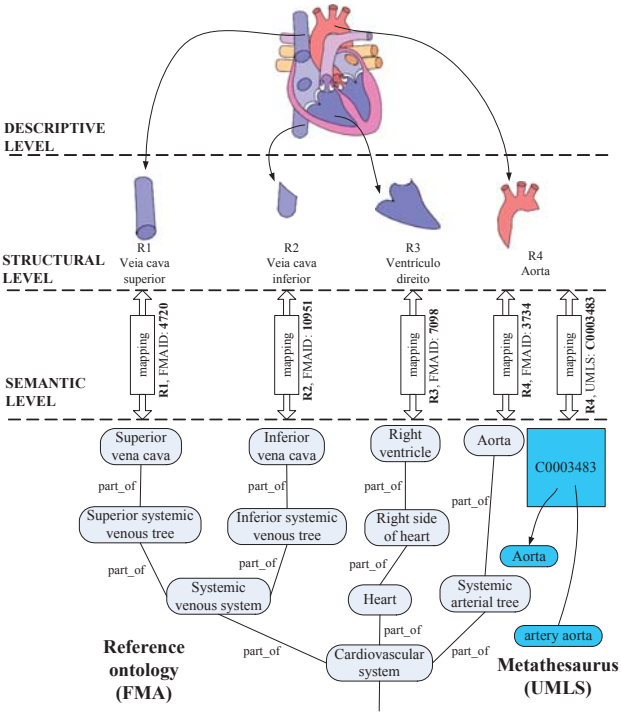


Fig. 2. Semantic images

C. Knowledge Repository

This component is composed by the artifacts used to represent the knowledge domain to perform both the semantic mapping of the images and documents as well as to make inferences, when possible, on the concepts used in the visualization processes.

D. Visualization

The visualization component is responsible for presenting the user search results. This component is based on tasks defined by Shneiderman [19] to visualize information: first obtain an overview of the data, then concentrating on items of interest and filter out irrelevant items, and finally to provide details on demand. The data overview is done using the semantic images, where the visual elements are changed in order that user can easily identify the regions where exists knowledge.

IV. Prototype Model

In order to demonstrate the feasibility of the proposed model, we developed a prototype applied to the biomedical field. The knowledge repository was composed by the Foundational Model of Anatomy ontology (FMA) [20], [21] and the Unified Medical Language System Metathesaurus (UMLS) [22]. As documents repository a copy SciELO (Scientific Electronic Library Online) database was used.

In figure 2 we can see an example of a semantic image. Four regions in this image were made explicit: the superior vena cava, the inferior vena cava, the right ventricle and the aorta. To annotate the images region, we used the FMA. Each region was linked to a FMA class incorporating to the

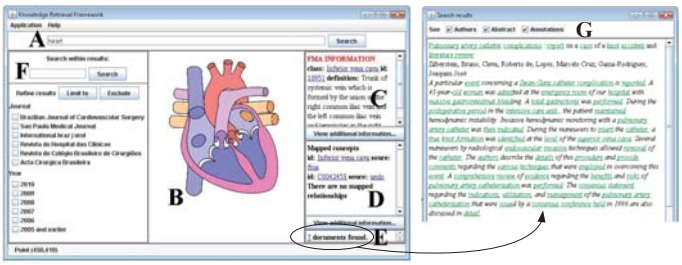


Fig. 3. Semantic annotation of documents

semantic image, all explicit knowledge in the FMA ontology. For example, when linking the R1 region (labeled with *veia cava superior*) to the class *Superior vena cava*, the model now "understands" that the region is part of the cardiovascular system (class *Cardiovascular system* in FMA ontology) which is also known as *Anterior vena cava* (synonym in English) and *Vena cava superior* (equivalent name in Spanish). Figure 2 also shows how the region R4 (labeled *aorta*) is mapped to the UMLS concept *C0003483*. Using UMLS relations, the model "knows" that *Aneurysm* is an aorta disease.

In figure 3 we can see the prototype's main interface. To search for the semantic images the user enters the search terms in the text box in the upper panel (figure 3-A). The tool recovers all the semantic images associated by processing the query terms in both the structural level (*i.e.*, text search for the name of the regions) and in the semantic level (*i.e.*, conceptual search by mappings). When the processing is done on the semantic level, the terms are processed in domain concepts. This transformation allows the query to be executed semantically, making the search language independent. For example, to search for semantic images related to heart, the user can enter search terms such as words like *corazn* (heart in Spanish), or *corao* (heart in Portuguese). In all cases, the process returns the same results.

After retrieving the semantic images, it is placed in the center pane (figure 3-B). The user can use it in two ways: to obtain knowledge related to the concepts represented in the regions or to retrieve documents that mention the concepts.

Information related to the concept appears in the top right pane (figure 3-C). In this prototype version, this information is presented using the FMA ontology. In this panel the user can view the name, identification and description of the class. Using the button "View additional information", the user can get additional information such as names in other languages or anatomical entities that constitute it. The concepts mapped in each region can be observed in the right pane in the center (figure 3-D). This information is obtained from the semantic level of the image.

When selecting an region in the image, in addition to presenting to the user information about the concept, also shows the number of documents related to the concepts mapped in each region.

V. Discussion

In the proposed model both images as the repositories of documents have been enriched with semantic content enabling

the integration of these two components. The semantic content allows that the model can to "understand" what are the concepts represented both in images and in the documents thus facilitating conceptual search. One advantage of this is the recovery of knowledge independently of the language of written documents. Due to this "understanding", the model is able to retrieve documents related to the concepts represented in the images helping the user in the search process. This retrieval mechanism can be seen in the prototype in which to retrieve documents, the user only need select one region of the image without specify the search terms as in traditional retrieval systems.

The idea of the model is that search results are presented in the images changing the color of the regions where it concentrates the largest number of documents. This model behavior is consistent with the visualization tasks defined by Shneiderman [19].

The reason for using images containing representations of the real world is based first on the humans' skills to process images quickly and secondly in the ease of individuals to relate prior knowledge associated with an image already known.

The proposed model is generic and can be used in any domain that allows the concepts representation by images. It can also be used on any document repository where these can be mapped into knowledge representations (*i.e.*, ontologies or taxonomies).

VI. Conclusion

One of the advantages offered by the model is the ease of documents retrieval using only regions of the images. In traditional IR systems, users translate their information needs in search terms, they return a list of items that match the most relevant documents according to the terms informed. In the proposed model users have to translate their information needs also in terms, but these are not used to directly seek the documents, but rather to seek the image to be used in visualization process.

Another model advantage is the component integrations by semantic mappings. The images and information repositories are all integrated by means of semantic information, this allows a single image can be used to view documents in different repositories. For example, the image of the heart, shown in previous sections, can be used to retrieve scientific articles, organizational competencies, projects, clinical studies, indicators, medical imaging, among others. This semantic integration also allows to the model to infer new information about a particular concept. When render the heart image, the model can retrieve information related to concepts that are not explicitly defined in the image, but can be inferred, such as certain diseases. In heart image when selecting the aorta, for example, and using the UMLS relations could be recovered documents related to the aneurysm, a disease that affects the aorta.

References

[1] M. Alavi and D. Leidner, "Review: Knowledge management and knowledge management systems: Conceptual foundations and research issues," *MIS Quarterly*, vol. 25, no. 1, pp. 107–136, 2001.

[2] N. Gershon, S. G. Eick, and S. Card, "Information visualization," *interactions*, vol. 5, no. 2, pp. 9–15, 1998.

[3] J. S. Yi, Y.-a. Kang, J. T. Stasko, and J. A. Jacko, "Understanding and characterizing insights: how do people gain insights using information visualization?" in *Proceedings of the 2008 conference on BEyond time and errors*. New York, NY, USA: ACM, 2008, pp. 1–6.

[4] J.-D. Fekete, J. J. Wijk, J. T. Stasko, and C. North, "The value of information visualization," in *Information Visualization: Human-Centered Issues and Perspectives*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 1–18.

[5] R. A. Burkhard, "Towards a framework and a model for knowledge visualization: synergies between information and knowledge visualization," in *Knowledge and Information Visualization*. Berlin/Heidelberg: Springer, 2005, vol. 3426, pp. 238–255.

[6] M. Eppler and R. A. Burkhard, "Visual representations in knowledge management: framework and cases," *Journal of Knowledge Management*, vol. 11, no. 4, pp. 112–122, 2007.

[7] W. Xiao-yue, "Visualization based on concept maps: An efficient way to knowledge sharing and knowledge discovery in e-science environment," in *Fourth International Conference on Fuzzy Systems and Knowledge Discovery, FSKD 2007*, M. Yan, Ed., vol. 2. Haikou, Hainan, China: IEEE Computer Society, 2009, pp. 144–147.

[8] C. Ware, *Information visualization: perception for design*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2000.

[9] R. A. Burkhard, "Learning from architects: The difference between knowledge visualization and information visualization," in *Eighth International Conference on Information Visualisation (IV'04)*. Los Alamitos, CA, USA: IEEE Computer Society, 2004, pp. 519–524.

[10] S. Steffen, M. Er, and H. Siegfried, "An annotation framework for the semantic web," in *Proceedings of the First Workshop on Multimedia Annotation*, S. Isjizaki, Ed., Tokyo, Japan, 2001.

[11] O. Corcho, "Ontology based document annotation: trends and open research problems," *International Journal of Metadata, Semantics and Ontologies*, vol. 1, no. 1, pp. 47–57, 2006.

[12] L. Reeve and H. Han, "Survey of semantic annotation platforms," pp. 1634–1638, 2005.

[13] V. Uren, P. Cimiano, J. Iria, S. Handschuh, M. Vargas-Vera, E. Motta, and F. Ciravegna, "Semantic annotation for knowledge management: Requirements and a survey of the state of the art," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 4, no. 1, pp. 14–28, 2006.

[14] A. Hanbury, "A survey of methods for image annotation," *Journal of Visual Languages & Computing*, vol. 19, no. 5, pp. 617–627, 2008.

[15] A. T. Schreiber, B. Dubbeldam, J. Wielemaker, and B. Wielinga, "Ontology-based photo annotation," *Intelligent Systems, IEEE*, vol. 16, no. 3, pp. 66–74, 2001.

[16] T. Osman, D. Thakker, G. Schaefer, and P. Lakin, "An integrative semantic framework for image annotation and retrieval," in *Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence*. IEEE Computer Society, 2007, pp. 366–373.

[17] W. Hsu, S. Antani, L. R. Long, L. Neve, and G. R. Thoma, "Spirs: A web-based image retrieval system for large biomedical databases," *International Journal of Medical Informatics*, vol. 78, no. Supplement 1, pp. S13–S24, 2009.

[18] D. Sonntag and M. Mller, "A multimodal dialogue mashup for medical image semantics," in *Proceeding of the 14th international conference on Intelligent user interfaces*. Hong Kong, China: ACM, 2010, pp. 381–384.

[19] B. Shneiderman, "The eyes have it: a task by data type taxonomy for information visualizations," in *Proceedings of the 1996 IEEE Symposium on Visual Languages*. Boulder, CO, USA: IEEE, Los Alamitos, CA, United States, 1996, pp. 336–343.

[20] C. Rosse and J. Mejino, "A reference ontology for biomedical informatics: the Foundational Model of Anatomy," *Journal of Biomedical Informatics*, vol. 36, no. 6, pp. 478–500, 2003.

[21] ——, "The foundational model of anatomy ontology," in *Anatomy Ontologies for Bioinformatics*, ser. Computational Biology. London, UK: Springer, 2008, vol. 6, pp. 59–117.

[22] O. Bodenreider, "The unified medical language system (umls): integrating biomedical terminology," *Nucleic Acids Research*, vol. 32, no. Database Issue, pp. 267–270, 2004.

Graph Grammar Based Web Data Extraction

Amin Roudaki
Computer Science Department
North Dakota State University
amin.roudaki@ndsu.edu

Jun Kong
Computer Science Department
North Dakota State University
Jun.kong@ndsu.edu

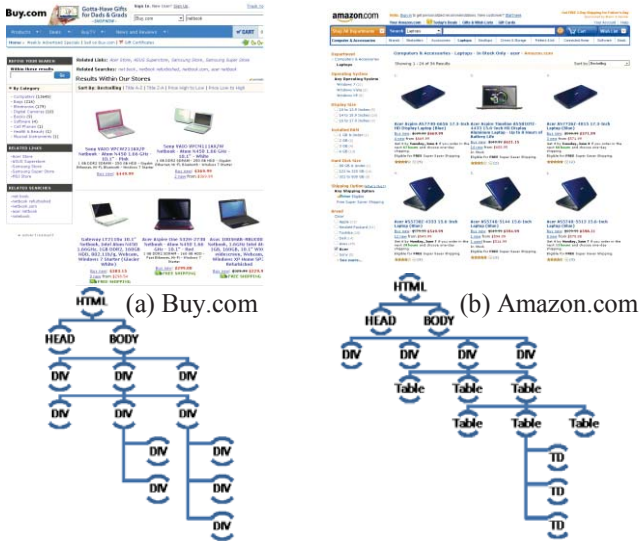
***Abstract*—Web data extraction becomes a hot topic after the invention of World Wide Web, because the large amount of information on the Web makes it challenging to retrieve useful information. Due to the diverse designs and presentations of information on different Web sites, it is hard to implement a general solution to extract data across different Web sites. This paper presents a novel method based on graph grammar to extract the same type of information from different Web sites without the need of training or adjustment. Our approach formalizes a common Web pattern as a graph grammar. Then, based on the visual layout and HTML DOM structure, a Web page is abstracted as a spatial graph that highlights the essential spatial relations between information objects. According to the defined graph grammar, a spatial parsing is performed on the spatial graph to extract structured records. We have evaluated our approach on twenty one different Web sites, and achieved the F1-score as 97.49% which shows promising performance.**

Keywords: Web Data Extraction; Graph Grammar; Wrapper

I. INTRODUCTION

Exploring useful information becomes increasingly difficult as the volume and diversity of available information rapidly grow. To efficiently discover knowledge from the vast amount of heterogeneous digital data on the Web, it is critical to extract meaningful contents from Web pages and organize extracted information in a structured format, i.e. *Web data extraction.*

HTML DOM structures could be very diverse among different Web sites. For example, some Web designers may use *table* to present tabular data while others use *table* to divide space into different grids for layout purpose. Therefore, even if two Web pages have similar layouts, their HTML source codes may be completely different. For example, Figure 1 presents two Web pages that have similar layouts but different DOM structures. Due to the diversity, it is challenging to make a wrapper applicable to different Web sites of the same category. In addition, a DOM structure is complex. For example, even the DOM structure of Google homepage includes 110 HTML tags. The complexity of DOM structures could further increase the diversity and reduce the accuracy. Recently, layout based analysis receives more and more attention [3, 14, 19]. In order to provide efficient Web browsing, Web pages that include similar information in general are presented with consistent layouts even though those pages may be implemented completely differently. Therefore, layout based analysis addresses the diversity issue to a certain degree. However, existing layout-based approaches have limited applicability. Some approaches [19] need training before they are applied to a Web site. Some approaches are optimized for a specific type of domains, and may not be easily adjusted to another domain. For example, the Visual Wrapper [3] is powerful to extract news stories while it is not applicable to other domains.



(a) Buy.com (b) Amazon.com

Figure 1. The same visual layout with different DOM structures

This paper presents a novel approach that combines layout and DOM structure analysis. Our approach extracts structured records by analyzing the screenshot of a Web page that is coherent with human's cognition of visual perception. In our approach, the screenshot of a Web page is first abstracted as a spatial graph, in which a node is an information object (such as text or image) and an edge indicates a close semantic relation between two information objects. Instead of using machine vision technique to recognize images and texts, our approach recognizes information objects based on the DOM structure. Though a DOM-structure based recognition is not as powerful as the machine vision technique, it is efficient and sufficient to recognize texts, links and images that are useful in information extraction. Based on the spatial graph, we are using the graph parsing technique to extract structured records.

The graph parsing assumes that Web designers usually follow some guidelines or patterns to present information on the Web. This is a valid assumption in practice since a consistent layout style can provide effective browsing for end users and ease the efforts of development and maintenance. This assumption has been validated by our evaluation on different Web sites and also by other researchers [15]. Those common guidelines are referred to as patterns that are visually specified through graph grammars [7] in our approach.

This work is supported in part by ND EPSCoR Grants #FAR0015845.

Since the input of data extraction is a Web page that is abstracted as a spatial graph and the output is a tree, the process of data extraction can be considered as transforming from one graph to another one that can be naturally specified through the graph grammar technology. Graph grammars provide a solid theoretic foundation to define computing in a two-dimensional space. In our approach, a graph grammar visually yet formally defines a Web pattern, and then the data extraction is implemented as a process of graph parsing that searches in a spatial graph the sub-structures consistent with the defined pattern. In order to minimize the manual effort of designing a graph grammar, we implemented a graphical interactive tool to facilitate the design of graph grammars. We have evaluated our approach on 21 Web sites to extract product information. The results are promising and the performance of our approach measured in terms of F1-Score (See Section V for further detail) is high.

II. RELATED WORK

With a clear structure to specify the layout of a Web page, the HTML source codes have been commonly analyzed to extract structured data records [1, 4, 5, 6, 8, 9, 10, 11, 12, 16, 18]. Several approaches [6, 8, 11] use the machine learning technique to automatically derive a wrapper based on a set of manually labeled training data. Though the above approaches apply different technologies to derive a wrapper, they all require a set of training data, which are manually labeled by human experts. Several approaches [1, 4, 5, 12] automatically derive a template from sample Web pages and use the extracted template to discover structured records. These approaches do not require manually labeled data, which greatly reduces the manual effort in the data extraction process. However, they require that Web pages being analyzed must follow the same template as the sample Web pages. MDR [10] and DEPTA [16] generate an HTML tag tree based on table and form related tags, e.g., *table*, *form*, *tr*, *td*, and etc. This HTML tag tree significantly reduces the complexity of the original Web page. Based on the HTML tag tree, they use the string comparison technique to divide a Web page into different regions. In each region, it identifies data records by calculating similarity between tag strings. Zhai *et. al.* [18] rendered a Web page and allowed users to select information objects in the screen shot to define a data pattern. Different from our approach, this data pattern is defined on the DOM structure, not on the visual layout. Laber *et. al.* [9] used some statistical analysis to analyze the DOM tree elements and identify the relevant information objects. All of the above methods use HTML DOM structure as the main source for data extraction. Instead, our approach, i.e. *Visual Grammar Based Extractor - VGE*, implements data extraction based on the information presentation. The visual analysis can address the issues of complexity and diverse usages of HTML DOM structures to a certain degree. By actually rendering a Web page, our approach supports dynamic information objects which are generated at run time.

Recently, the visual perception technique has been applied to extract structured data since it is independent from the detailed implementation underlying a Web page. These approaches [3, 14, 19] basically calculate the visual similarity among different Web pages to group semantically related information. [3, 19] are limited to extract news stories, and are not applicable to other domains. ViPER [14] is implemented on statistical models that emphasize on extracting repetitive data records.

The Hybrid method takes benefits from both DOM Structure and Visual Perception, and combines them together. ViNTs [17] automatically recognizes different content shapes based on the visual position of information objects. Afterward, a wrapper is generated based on an HTML structure which represents each shape. This approach still extracts information based on HTML DOM structures, though the wrapper is derived from visual analysis. Instead, our approach specifies extraction rules from both the layout and the DOM structure. ViNTs is limited to search results, while our approach is general to different domains.

III. A GRAMMAR BASED APPROACH

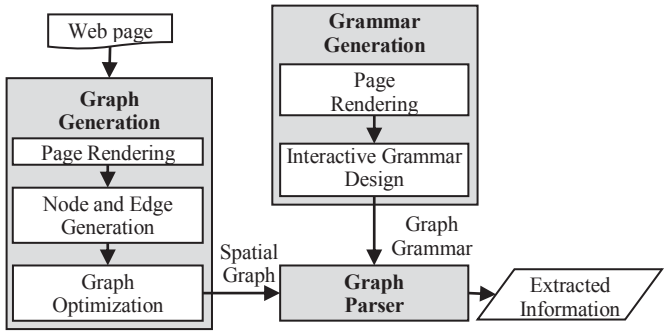


Figure 2. A Graph Grammar based Extractor

This paper presents a novel and robust approach, i.e. ***Visual Grammar based Extractor - VGE***, to extracting structured information. Our approach consists of three components as shown in Figure 2. The graph generation component abstracts a Web page as a spatial graph that simplifies the original Web page and highlights important semantic relations between recognized information objects. The graph generation proceeds in the following steps: (1) render a Web page on the screen, (2) recognize information objects and divide a Web page into different regions according to its DOM structure, (3) calculate semantic relations between recognized information objects based on the layout information, and finally (4) optimize the spatial graph. Based on the generated spatial graph, the data extraction is implemented as a graph parsing process that searches in the spatial graph sub-graphs satisfying certain spatial properties. Those spatial properties are visually defined through a graph grammar. The grammar generation component provides an interactive grammar design tool that allows end users to define a graph grammar by directly manipulating the screenshot of a Web page. This interactive grammar design tool eases the process of developing a graph grammar and improves the usability of our approach.

A. Graph Generation

HTML is a very flexible language. Information with the same presentation could be implemented in many different ways. Being coherent with the HCI principle that consistent presentations can improve the usability of an interface [13], Web designers across different Web sites commonly use similar layouts to present the same type of information.

Therefore, our approach extracts structured records by analyzing the layout of a Web page. The visual analysis can address the diversity of HTML usages and make our approach applicable to different Web sites. The process of graph generation is a critical step in our approach since it simplifies original Web pages and eliminates variations among different Web pages. The simplification only preserves essential information objects. Especially, the graph generation process removes (1) style and layout elements, which do not include any real content, (2) advertisements and (3) menus in the border areas. The simplification effectively reduces the complexity of HTML pages and removes potential noises in the data extraction. The graph generation process proceeds in three steps: Web page rendering, node and edge generation, and graph optimization.

The first step in the graph generation is to render a Web page. The visual layout of a Web page is determined by three variables, i.e. (1) the actual HTML source code that specifies the DOM structure of the page, (2) data items such as text and picture and (3) style sheets and client side scripts which are executed by a browser at run time. We can access all HTML elements, especially dynamic elements which are generated on the fly, only by actually rendering a Web page. Also, the page rendering determines the position and size of each element.

Based on the dynamic and static HTML elements and their spatial properties, the second step generates a spatial graph in which a node represents an information object for data extraction and an edge indicates a close semantic relation between the pair of connecting nodes. Contents are stored in three types of nodes, i.e. image, text and link. The contents enclosed in the <img> or <a> tags are recognized as an image node or a link node, respectively. However, it is challenging to identify a text node since one complete sentence may be separated by several HTML tags and it is necessary to consolidate those information pieces together. For example, inside the text block of a sentence, formatting and styling tags (such as <b>, <br>, <font>,) can divide the sentence into several pieces. In the graph generation, all those formatting and styling tags are removed and adjacent contents are consolidated as one single text node.

After identifying atomic information objects as nodes, it is critical to calculate semantic relations between information objects and use an edge to connect two nodes that are closely related in semantics. In a two dimensional space, an information object can have an arbitrary spatial relation with adjacent nodes. A complete spatial parsing that analyzes different spatial properties in a graph could be time consuming. Our approach first derives the semantic relation between adjacent nodes, and each close semantic relation is represented as an edge in the spatial graph. Based on the derived semantic relations, we can limit the spatial parsing to objects that have semantic relations and thus reduce the search space to speed up the parsing process. We have extensively investigated different Web sites and found that a small distance strongly indicates a close semantic relation between two objects. This observation is consistent with the Human Computer Interaction principle that closely related objects should be grouped together and placed in proximity [13]. Accordingly, we derive the semantic relation by calculating the distance between two objects. Also, an HTML DOM structure provides valuable hints for deriving semantic relations. Web designers group related objects together by using a container, such as *table* or *div*. In general, two objects belonging to two containers are not related. For example, in Figure 3, though *text* objects 4 and 5 are placed in proximity, they are not semantically related since they belong to two different containers. Our approach uses HTML DOM structures to recognize the containers, and semantic relations are limited to information objects that have one common (ancestor) container. In order to accommodate different image sizes and variations in Web pages, we propose a novel approach to calculating distance and deriving semantic relations. The size of an information object *a* is extended to a certain degree. If the extended object *a* is overlapping with at least two corners of another information object *b*, *a* has a semantic relation with *b*.

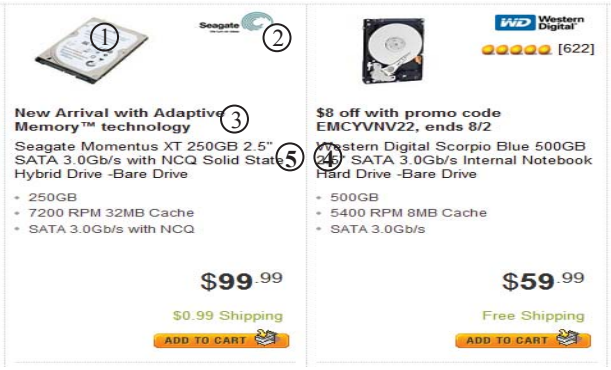


Figure 3. Different containers in a Web page

The last step in the graph generation is to optimize the generated spatial graph. In a spatial graph, some nodes may be considered as noises (such as advertisements and menus), which do not contribute to the data extraction process. Since those objects in general are placed in the border areas of a Web page, we can remove them according to their position. Another type of noise is small repetitive pictures, such as the "*Add to Cart"* icon in Figure 3.

B. Grammar Generation and Parser

The graph generation component generates a spatial graph, and the data extraction is performed on a spatial graph to search for information objects having certain spatial relations. In order to support efficient browsing, the same type of information in general is presented similarly across different Web sites. Those consistent spatial features among information objects are summarized as a Web pattern that is visually specified through a graph grammar. A graph grammar defines computation in a multi-dimensional fashion based on a set of rewriting rules, i.e. *productions*. Since the input of data extraction is a graph and the output is a tree structure that represents structured records, the data extraction is essentially a process of graph transformation that can be naturally specified through graph grammars. Furthermore, a graph grammar is powerful to handle the variations among instances of a Web pattern. This paper selects the Spatial Graph Grammar (SGG) [7] as the specifying formalism. With the capability of spatial specification in the abstract syntax, SGG provides the flexibility to define a pattern from both the edges (i.e. close semantic relations) and spatial features (e.g. directions).

Instead of designing a graph grammar from scratch, we designed an interactive design tool that allows users to design a graph grammar visually and intuitively. This interactive tool renders a sample Web page on the screen, and highlights recognized information objects in the Web page. Users can directly select one or more information objects in the Web page to make a production. This tool supports a direct manipulation interaction on the grammar design that reduces the gap between a concrete Web pattern and an abstract graph grammar. With the help of this tool, even users without much training in graph grammars may design a graph grammar.

IV. SYSTEM IMPLEMENTATION

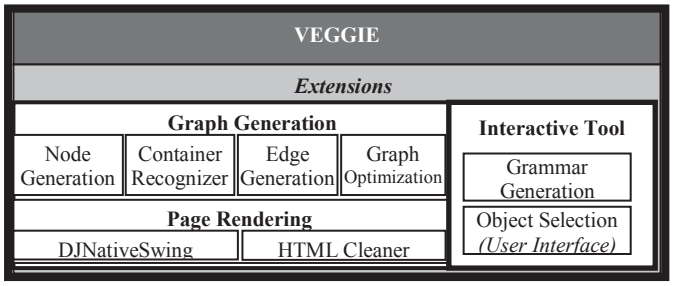


Figure 4. The VGE system architecture

We have implemented a prototype for our approach. Our prototype is built based on VEGGIE - **V**isual **E**nvironment for **G**raph **G**rammars: **I**nduction and **E**ngineering [2]. VEGGIE is a general visual programming environment, and supports the Spatial Graph Grammar specification and parsing. VEGGIE mainly consists of three independent editors (i.e., the Type Editor, the Grammar Editor, and the Graph Editor) and an SGG parser. The three editors provide GUIs for designers to visually design a graph grammar, and are closely related and seamlessly working together in VEGGIE. Grammar designers can visually create visual objects, i.e. node types, in the Type Editor, or import existing node types from a file in the form of GraphML. Then, based on these defined nodes, the designer can define productions in the Grammar Editor. The designer can visually draw or import a host graph to be analyzed by the SGG parser. As shown in Figure 4, we have extended VEGGIE with two subsystems, both implemented in Java. The first sub-system is responsible to generate a spatial graph from a Web page. The generated spatial graph is fed to the SGG parser for a spatial parsing. The second sub-system provides an interactive graphic tool to design a graph grammar.

The Graph Generation sub-system has several components. The *Page Rendering* component renders a Web page, extracts size/position information and passes the output to the *Node Generation* to generate nodes. Containment relations are identified by the *Containment Recognizer*. The *Edge Generation* derives semantic relations based on containers and spatial properties. The *Graph Optimization* component optimizes a generated spatial graph by removing noises.

The *Page Rendering* component renders a Web page based on the DJNativeSwing Browser (http://djproject.sourceforge.net/ns/). The HTML Cleaner component (http://htmlcleaner.sourceforge.net) solves syntactical problems (e.g., unclosed tags and markup errors). The HTML Cleaner returns a clean and well-structured HTML DOM tree that includes all static and dynamic elements. Based on this DOM tree, the graph generation component, including *node generation*, *container recognizer*, *edge generation* and *graph optimizer*, generates an optimized spatial graph for data extraction. The second sub-system eases the process of designing a graph grammar. It consists of two components, i.e. *object selection* and *grammar generation*.

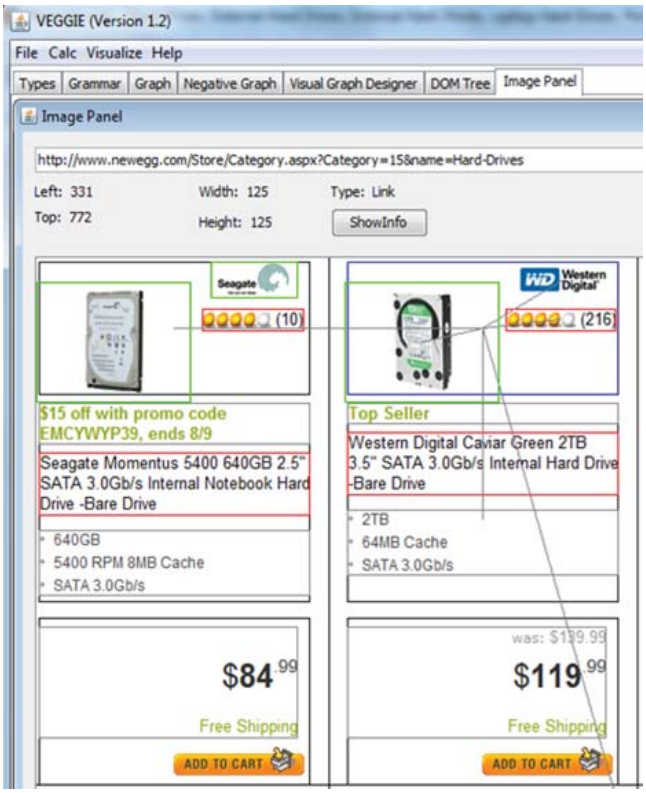


Figure 5. Browsing a Web page

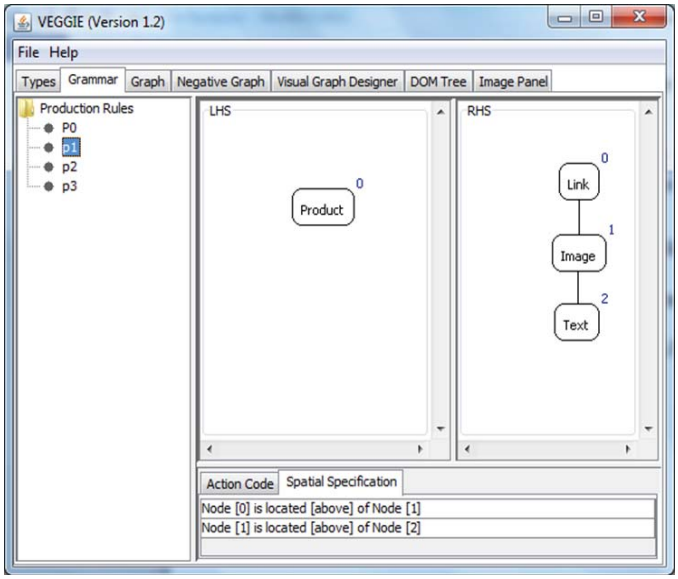


Figure 6. A grammar editor

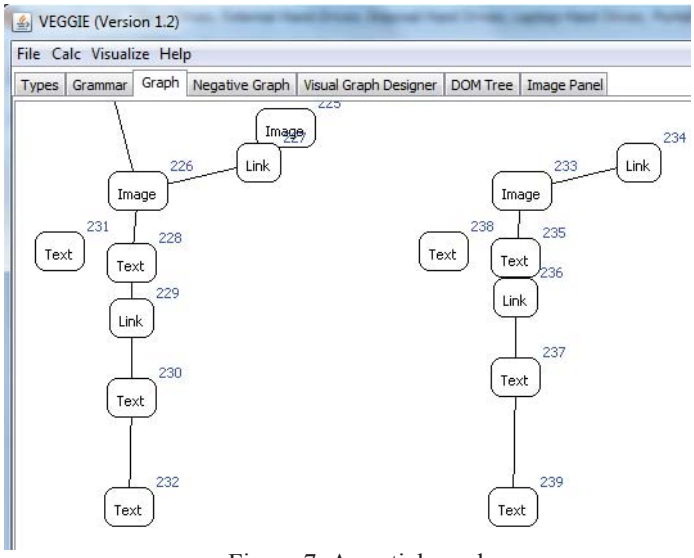


Figure 7. A spatial graph

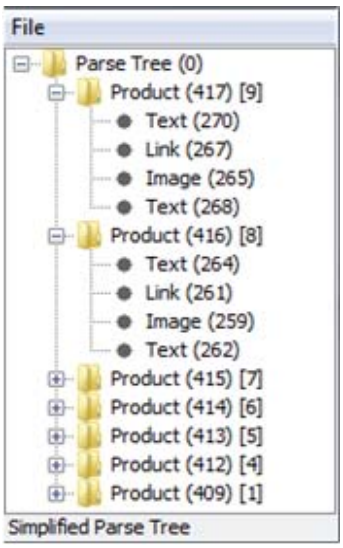


Figure 8. Parsing result

The prototype provides a complete system to support data extraction, from the specification of extraction knowledge (i.e. defining a graph grammar) to the visualization of extraction results. At first, a user uses the *Image Panel*, as presented in Figure 5, to display a Web page. In Figure 5, recognized information objects are highlighted with rectangles, and edges indicate close semantic relations. Based on the screenshot of a Web page, a user can select related information objects to intuitively design a graph grammar. After a graph grammar is defined, the user can go to the VEGGIE Type Editor or Grammar Editor (as presented in Figure 6) to elaborate the designed graph grammar. Once a graph grammar is finalized, a user can use the prototype to extract structured records that are consistent with the defined graph grammar. A user first inputs the URL of a Web page in the image panel. Then, the corresponding spatial graph is automatically generated and can be retrieved in the graph panel, as presented in Figure 7. By applying the defined graph grammar to the spatial graph, a small popup window shows up to present the extracted records, as shown in Figure 8.

V. EXPERIMENT

This section discusses the primary experiment on VGE. We first discuss the design of the experiment, and then present the results.

A. Setup

Experiment web pages: We have evaluated our approach on 21 ecommerce Web sites, which include well known Web sites, such as ebay.com, lycos.com, amazon.com, compusa.com, and etc.

Measurement: We measured the performance with the standard metrics: $recall = \frac{Ecorrect}{Ntotal}$; $precision = \frac{Ecorrect}{Etotal}$;

Where *Ntotal* is the number of data records contained in a Web page; *Ecorrect* indicates the total number of correctly extracted data records; and *Etotal* denotes the total number of data records extracted from a Web page. We also calculated the F1-Score, which is the harmonic mean of *precision* and *recall* and is defined as $\frac{2\times recall\times precision}{recall+precision}$. The F1-Score has been commonly used as a metric to evaluate the overall performance in many approaches [3, 9].

Execution Platform: We have evaluated our approach on a desktop with a Core 2 Duo CPU 2.26 GHz and 4 GB RAM, running Windows 7 Professional.

B. Evaluation

Precision/Recall/F1-Score: The evaluation results are presented in Table 1. The recall of our approach is 99.5. The high recall rate in our approach indicates that graph-grammar-based visual analysis is powerful to recognize structured records. The precision of our approach is 95.5%. Our approach may falsely recognize some records, which are mainly caused by noise. For example, if an advertisement is placed in the central area and its overall layout is similar to our selected pattern (e.g. including a link, a picture and several lines of textual description that are displayed vertically); this advertisement may be recognized as a product record. In order to improve the precision, it is critical to improve the graph generation process by removing potential noise. F1-Score shows the overall performance. Our approach has a high F1-Score of 97.49%. In summary, the results indicate our approach has a good performance in terms of both precision and recall.

Table 1. Evaluation Results

Domain Name	# of Structured Records	Our approach	
		Correct	Found
shopping.yahoo.com	15	14	14
scistore.cambridgesoft.com	13	13	14
shop.lycos.com	18	18	18
www.barnesandnoble.com	48	48	48
www.borders.com	27	28	29
www.circuitcity.com	5	5	7
www.compusa.com	18	18	21
www.drugstore.com	15	13	14
www.ebay.com	20	20	20
www.etoys.com	32	32	32
www.kidsfootlocker.com	29	29	29
www.kodak.com	20	20	20
www.newegg.com	20	20	26

www.nothingbutsoftware.com	24	24	24
www.overstock.com	18	18	18
www.powells.com	50	50	51
www.softwareoutlet.com	14	14	15
www.ubid.com	8	8	9
www.amazon.com	7	7	8
www.shopping.hp.com	5	5	5
www.qualityinks.com	24	24	26
Total	430	423	443
Recall/Precision		99.5%/95.5%	
F1-Score		97.49%	

VI. CONCLUSION

This paper presents a novel and general solution to extract data across different Web sites. Our method works based on graph grammars to extract the same type of information from different Web sites without the need of training and adjustment for different Web sites. Our approach utilizes both the visual features of a rendered Web page and the HTML DOM structure to extract structured records. We have implemented a prototype and tested the prototype on 21 Web sites. The evaluation shows promising results. Our approach has a high F1-Score as 97.49%. The evaluation results indicate our approach has a good performance in terms of both precision and recall. The main advantage of our approach lies in its ability to distinguish the most important contents from less important and noisy information and to convert the complex HTML DOM structure to a simple spatial graph. The generated spatial graph significantly reduces the complexity of the original Web page. Based on the simplified spatial graph, our approach is efficient to extract structured records through a graph parsing.

In the future work, we will identify more spatial relations between information objects, and optimize the graph generation algorithm. These optimizations may increase the quality of generated spatial graphs, which can affect both the precision and recall.

REFERENCES

[1] Arasu, A. and Garcia-Molina, H. 2003. Extracting structured data from Web pages. In *Proceedings of the 2003 ACM SIGMOD international Conference on Management of Data*. 337-348.

[2] Ates, K. and Zhang, K. 2007. Constructing VEGGIE: machine learning for context-sensitive graph grammars. In *Proceedings of the 19th IEEE international Conference on Tools with Artificial intelligence - Volume 02*. ICTAI. IEEE Computer Society, Washington, DC, 456-463.

[3] Chen, J. and Xiao, K. 2008. Perception-oriented online news extraction. In *Proceedings of the 8th ACM/IEEE-CS Joint Conference on Digital Libraries*. ACM, New York, NY, 363-366.

[4] Chuang, S. and Hsu, J. Y. 2004. Tree-structured template generation for Web pages. In *Proceedings of the 2004 IEEE/WIC/ACM international Conference on Web intelligence*. IEEE Computer Society, Washington, DC, 327-333.

[5] Crescenzi, V., Mecca, G., and Merialdo, P. 2001. RoadRunner: towards automatic data extraction from large Web sites. In *Proceedings of the 27th international Conference on Very Large Data Bases*. P. M. Apers, P. Atzeni, S. Ceri, S. Paraboschi, K. Ramamohanarao, and R. T. Snodgrass, Eds. Very Large Data Bases. Morgan Kaufmann Publishers, San Francisco, CA, 109-118.

[6] Hsu, C. and Dung, M. 1998. Generating finite-state transducers for semi-structured data extraction from the Web. *Inf. Syst.* 23, 9, 521-538.

[7] Kong, J., Zhang, K., and Zeng, X. 2006. Spatial graph grammars for graphical user interfaces. *ACM Trans. Comput.-Human Interact.* 13, 2, 268-307.

[8] Kushmerick, N., Weld, D., & Doorenbos, R. 1997. Wrapper induction for information extraction. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*. 729-737.

[9] Laber, E. S., de Souza, C. P., Jabour, I. V., de Amorim, E. C., Cardoso, E. T., Rentería, R. P., Tinoco, L. C., and Valentim, C. D. 2009. A fast and simple method for extracting relevant content from news webpages. In *Proceeding of the 18th ACM Conference on information and Knowledge Management*. ACM, New York, NY, 1685-1688.

[10] Liu, B., Grossman, R., and Zhai, Y. 2003. Mining data records in Web pages. In *Proceedings of the Ninth ACM SIGKDD international Conference on Knowledge Discovery and Data Mining*. ACM, New York, NY, 601-606.

[11] Muslea, I., Minton, S., and Knoblock, C. 1999. A hierarchical approach to wrapper induction. In *Proceedings of the Third Annual Conference on Autonomous Agents*. O. Etzioni, J. P. Müller, and J. M. Bradshaw, Eds. AGENTS '99. ACM, New York, NY, 190-197.

[12] Reis, D. C., Golgher, P. B., Silva, A. S., and Laender, A. F. 2004. Automatic web news extraction using tree edit distance. In *Proceedings of the 13th international Conference on World Wide Web*. ACM, New York, NY, 502-511.

[13] Shneiderman, B. 2009. Designing the User Interface: Strategies for Effective Human-Computer Interaction. Addison-Wesley Longman Publishing Co., Inc.

[14] Simon, K. and Lausen, G. 2005. ViPER: augmenting automatic information extraction with visual perceptions. In *Proceedings of the 14th ACM international Conference on information and Knowledge Management*. ACM, New York, NY, 381-388.

[15] Zhang, Z., He, B., and Chang, K. C.-C. 2004. Understanding Web query interfaces: best-effort parsing with hidden syntax. In *Proceedings of 2004 ACM SIGMOD International Conference on Management of Data*, 107-118.

[16] Zhao, H., Meng, W., Wu, Z., Raghavan, V., and Yu, C. 2005. Fully automatic wrapper generation for search engines. In *Proceedings of the 14th international Conference on World Wide Web*. ACM, New York, NY, 66-75.

[17] Zhai, Y. and Liu, B. 2005. Web data extraction based on partial tree alignment. In *Proceedings of the 14th international Conference on World Wide Web*. ACM, New York, NY, 76-85.

[18] Zhai, Y. and Liu, B. 2007. Extracting Web data using instance-based learning. *World Wide Web* 10, 2, 113-132.

[19] Zheng, S., Song, R., and Wen, J. 2007. Template-independent news extraction based on visual consistency. In *Proceedings of the 22nd National Conference on Artificial intelligence - Volume 2*. A. Cohn, Ed. Aaai Conference On Artificial Intelligence. AAAI Press, 1507-1512.

Cyclic Association Rules: Coupling Dimensions And Measures

Eya Ben Ahmed, Ahlem Nabli and Faïez Gargouri
University of Sfax, Higher Institute of Computer Science and Multimedia of Sfax, Tunisia
eya.benahmed@gmail.com, ahlem.nabli@fsegs.rnu.tn, faiez.gargouri@isimsf.rnu.tn

***Abstract*— On-line analytical processing (OLAP) provides tools to explore data cubes in order to extract interesting information. Nevertheless, it cannot offer any explanation of relationships that could exist within data. To achieve this goal, the association rules were performed on data cubes. We focus in this work on a particular class of association rules which is the cyclic association rules. The latter aims to discover rules that occur in user-defined intervals at regular periods. Generally, the generated patterns do not take into consideration the specificities of the multi-dimensional context *i.e.*, the measures and their aggregations. In this paper, we propose a new method of extraction of cyclic association rules from both dimensions and measures. In addition, we redefine the quality metrics of the derived patterns using the summarizability of measures through applying the suitable aggregation functions. To prove the utility of our approach, we undertake an empirical study on a real data warehouse.**

I. Introduction

Coupling of OLAP with data mining can bring explanations of the correlations that may exist between the multidimensional data. Recently, many works focused on mining association rules from data warehouses. Some of them were mainly interested in mining association rules from data cubes. A particular form of the generated association rules is the cyclic ones. In fact, cyclic association rules (CAR) are defined as rules that occur in user-defined intervals at regular periods throughout a dataset. Compared to the stampede algorithmic effort for extracting CAR ([1],[9],[6],[13], [14]), only few works study the CAR drawn from multi-dimensional context [3] which does not take enough advantage of the data cube structure. However, many research works exploit the aggregate function in order to generate association rules, such as the COUNT function or the SUM function [4]. Nevertheless, in analysis process, users generally focus on multi-dimensional data and their associations according to more elaborated measures than simple frequencies or summations. So, restricting the considered aggregate functions to the SUM and the COUNT limits the possible considered measures or eliminates the concept of the aggregation measure during the mining process. In fact, some measures can not be summarized using addition across any dimension. Such non-additive measures are frequent and experts highly need to extract association rules from both measures and dimensions. To better illustrate our contribution, we assume that the table $\mathscr{T}$ shown in table I, is defined over three dimensions, namely: the `Time` T of the transactions, the `Location` L where the transactions took place (Jordan, Yemen) , the `Range` R (let us consider both *Licensed Range* and *Model Hospital*) as well as the three measures, namely: the `sold Quantity` Q of the range R, the `Exchange Rate` ER, the `External Turnover` ET. The `sold quantity` cannot be aggregated along the time dimension, similarly, the `exchange rate` cannot be meaningfully summarized using addition across any dimension, and the `external turnover` cannot be expressively summarized through addition along all dimensions. For instance, we aim at building rules combining several dimensions and measures like " *Every odd quarter, the* `sold quantity` *of the licensed range which is at the rate of 77520 units is sold in Jordan with an* `external turnover` *reaching 750 in the respect of an* `exchange rate` *equal to 1,948%*". The interestingness measures of such a rule are articulated on the semantic aggregation of measures using the fitting aggregation function. For example, 77520 and 750 are occurrences of different measures and should not be evaluated using the same criteria. Therefore depending on the summarizability of each measure, we employ the suitable aggregation function to qualify the importance of each value on the dataset.

Time	Location	Range	Sold Quantity	Exchange Rate	External Turnover
T	*L*	*R*	*Q*	*ER*	*ET*
Q1 of 2007	Jordan	Licensed Range	77520	1,948%	750
Q2 of 2007	Jordan	Licensed Range	75000	1,947%	550
Q3 of 2007	Jordan	Licensed Range	77520	1,948%	750
Q4 of 2007	Jordan	Licensed Range	80000	1,946%	1000
Q1 of 2008	Yemen	Model Hospital	71100	1,382%	450
Q2 of 2008	Yemen	Model Hospital	80000	1,382%	800

TABLE I

TABLE $\mathscr{T}$

To the best of our knowledge, few works address the cyclic patterns mining issue from multi-dimensional context containing semi or non-additive measures. The originality of our approach is to combine dimensions and measures during the mining process. We aim to investigate such measures that cannot be aggregated using any aggregate functions, especially the COUNT or the SUM function and to redefine the quality metrics of these generated rules through the appropriate aggregation functions. In order to mine cyclic rules from by coupling dimensions and measures, we extend our approach which exclusively takes into account the measures on the mining process [2]. This paper is organized as follows. In Section 2, we give a brief overview of our chosen discretization technique of quantitative attributes, specially in the multidimensional context. Then, we represent the cyclic patterns. After that, we review the multidimensional association rule mining approaches. To close this section, we

focus on the measures summarizability concept. In Section 3, we develop the formal background, notations, and definitions of our proposal. Section 4 studies our contribution. Section 5 describes our algorithm. In Section 6, we conduct some experiments performed on real data warehouse. Finally, Section 7 gives a conclusion and future research directions.

II. Related works

In this section, we present the related works.

A. Discretization of quantitative attributes

The *equal frequency* technique divides the range of possible values into N bins, each of which holds the same number of training instances. In the multidimensional context, Palaniappan et *al.* underline that the *equal frequency* method performs respectable results and is lossless time consuming.

B. Cyclic patterns

The extraction of the CAR was introduced by Ozden et *al.* in order to better identify sales trends cyclicly. It involves the association rules mining from articles characterized by their regular variation over time, such as the daily, weekly, quarterly, or annual regular variation which is naturally cyclic. Such cycles are specified by the user to divide the data into disjoint segments. Several algorithms were proposed such as INTERLEAVED and SEQUENTIAL introduced by [9] or MTP presented by Thuan [13], [14] or the Chiang's method to combine cyclic and sequential patterns [5] or PCAR, proposed by [1].

C. Inter-dimensional association rules mining

Falling within the combination of several analysis dimensions on association rules mining and based on the quality evaluation of the generated patterns, two main pools can be distinguished: (i) Approaches using the COUNT function; (ii) Approaches using the SUM function.

1) Count-based evaluation of inter-dimensional association rules: The main idea is to compute the support and the confidence of inter-dimensional association rules according to the COUNT function. In fact, a COUNT cell of the cube stores the number of occurrences of the corresponding multi-dimensional data values. Several works have investigated this challenging solution. Firstly, Kamber et *al.* use the COUNT function [7]. Plantevit et *al.* advanced an OLAP-sequential multi-level mining [12]. Besides, Ben Ahmed et *al.* [3] study the CAR mining from data warehouses.

2) SUM-based evaluation of inter-dimensional association rules: Ben Messaoud et *al.* redefine the support and the confidence metrics using the SUM aggregation function [4]. Such contribution scrutinizes facts according to summarized values of measures more meaningful than the regular number of occurrences of facts.

Except the proposal of plantevit et *al.* [11], where an initiation to the mining process of association rules from quantitative measures is investigated, all the remainder approaches neglect the measures during the mining process of association rules. In addition, they are restricted to the COUNT function to evaluate the derived rules except the proposal of Ben Messaoud which is mainly based on the SUM function. Nevertheless, in real situations, the aggregation of some fact attributes might not be semantically meaningful along all dimensions. For that reason, the exclusive use of the COUNT or the SUM aggregation function is mostly unfitting so the generated rules are imperfectly evaluated.

D. Measures Summarizability

Mandatory to determine the appropriate aggregation operators, Lenz and Shoshani distinguish three types of measures [8], namely, *(i)* ***Additive measure*** values can be combined meaningfully along any dimension (e.g., Add the total sales or the external turnover over `location` and `time`); *(ii)* ***Semi-additive measure*** values cannot be combined along one or more of the dimensions, most often the `time` dimension (e.g., Do not sum inventory item because the quantity may be counted several times, but we can sum inventory levels across products); *(iii)* ***Non-additive measure*** values cannot be combined along any dimension, usually because of the chosen formula (e.g., Compute the item price and the cost per unit of a product).

In this paper, we use the concept of measures summarizability proposed by Lenz [8] to introduce a general process by coupling both of dimensions and measures to mine measure-based CAR with a specific evaluation of each measure depending on its appropriate aggregate functions. To do this, an algorithm based on the Apriori and incorporating dimensions and measures is introduced to extract cyclic rules on one side. On the other side, we integrate the summarizability of data cube measures in the computation of the support and the confidence of such rules to better evaluate the derived patterns.

III. Formal background and notations

In this section, we introduce our innovative basic concepts inspired of the proposal of Plantevit et *al.* that will be of use in the remainder.

A. Context partition

We consider that all is set in a multi-dimensional context. The three necessary data for cyclic mining drawn from classic context (Customer, Product, Date) become in a multi-dimensional context sets. Let us consider a relational table $\mathscr{T}$, in which transactions issued by customers are stored, defined on a set D of d dimensions and a set M of m measures. The table is partitioned into two sets: *(1) Endogen context* composed of the context dimensions $\mathscr{D}_C$ and the context measures $\mathscr{M}_C$ included in the analysis process; *(2) Exogen context* related to all the excluded dimensions and measures from the analysis. We focus on the following on the endogen context.

1) Context dimensions: The context dimensions $\mathscr{D}_C$ can be divided into three subcategories: *(i) Temporal dimension* $\mathscr{D}_T$ (date in classical context); *(ii) Reference dimensions* $\mathscr{D}_R$ (customer in classical context); *(iii) Analysis dimensions* $\mathscr{D}_A$ (product in classical context). All reference dimensions $\mathscr{D}_R$ are a conjunction of several dimensions where each dimension can have a single attribute value or a set of occurrences.

2) Context measures: The context measures $\mathscr{M}_C$can be partitioned into two subcategories: *(i) Reference measures* $\mathscr{M}_R$ (table is partitioned according to tuple values over reference measures); *(ii) Analysis measures* $\mathscr{M}_A$ (tuples over

analysis measures are those that appear in the items that constitute the cyclic patterns to be mined).

The analysis measures can be divided into four subcategories: *(1) Set of additive measures* $\mathscr{M}_{SUM}$ when the SUM function can be used for data aggregation; *(2) Set of maximal measures* $\mathscr{M}_{MAX}$ when the MAX function can be applied to aggregate the data; *(3) Set of minimum measures* $\mathscr{M}_{MIN}$ when the MIN aggregation function can be used; *(4) Set of average measures* $\mathscr{M}_{AVG}$ when the AVG aggregation function can be applied.

In our running example shown by table 1, we consider the whole table as our endogen context composed of : *(i)* context dimensions $\mathscr{D}_C$=$\{T, R, L\}$ with the temporal dimension $\mathscr{D}_T$=$\{T\}$, the reference dimension $\mathscr{D}_R$= { *L*:"Jordan" } and the analysis dimensions $\mathscr{D}_A$=$\{R\}$; and *(ii) context measures* $\mathscr{M}_C$=$\{Q, ER, ET\}$ with the analysis measures is equal to all existing measures and no reference measure is limited to a fixed value of measure and $\mathscr{M}_{MAX}$=$\{ Q\}$ and $\mathscr{M}_{MIN}$=$\{ET\}$ and $\mathscr{M}_{AVG}$=$\{ER\}$.

Definition 1: ***(Endogen Sub-cube)***: Let $\mathscr{D}' \subset \mathscr{D}$ a nonempty set of p dimensions $\{\mathscr{D}_1,...,\mathscr{D}_p\}$ extracted from the data cube *C* (p $\leq$ d) with d the cardinality of all dimensions and $\mathscr{M}' \subset \mathscr{M}$ a nonempty set of q measures $\{\mathscr{M}_1,...,\mathscr{M}_q\}$ extracted from the data cube *C* (q $\leq$ m) with m the cardinality of all measures.

The e-tuple $(\delta_1,...,\delta_t)$ is called an endogen sub-cube of data *C* according to : (i) $\mathscr{D}'$ iff $\forall$ i $\in \{1,...,p\}$, $\delta_i \neq \emptyset$ and $\delta_i \in$ Dom($\mathscr{D}_i$); (ii) $\mathscr{M}'$ iff $\forall$ j $\in \{1,...,q\}$, $\gamma_j \neq \emptyset$ and $\gamma_i \in$ Dom($\mathscr{M}_j$).

In our example, in the endogen sub-cube, each e-tuple e = (d_1,...,d_p,m_1,...,m_q) can be written in the form of a triple e = (r, a , t) where r is the restriction of $\mathscr{D}_R$ and $\mathscr{M}_R$, a is the restriction of $\mathscr{D}_A$ and $\mathscr{M}_A$ and t is the restriction of $\mathscr{D}_T$.

B. Measure-Based Uni-dimensional Cyclic Item and Inter-dimensional Cyclic Itemset

Definition 2: ***Measure-based Uni-Dimensional Cyclic Item***: Let the analysis dimensions $\mathscr{D}_A$ = $\{\mathscr{D}_1,...,\mathscr{D}_p\}$ and the analysis measures $\mathscr{M}_A$ = $\{\mathscr{M}_1,...,\mathscr{M}_q\}$ and a length of cycle *l*. A measure-based uni-dimensional cyclic item α is an item satisfying at least one of the following conditions:

- belonging to one of the analysis dimensions, namely, $\mathscr{D}_k$ and having a value of α= d_k on the date *t* and compulsorily on the date $t+l$ such that $\forall$ k $\in$ [1,p], d_k $\in$ Dom($\mathscr{D}_k$);
- belonging to one of the analysis measures, namely, $\mathscr{M}_l$ and having a value of α =m_l on the date *t* and necessarily on the date $t+l$ such that $\forall$ l $\in$ [1,q], m_l $\in$ Dom($\mathscr{M}_l$).

Typical examples of measure-based uni-dimensional cyclic item, considered in the multi-dimensional context, shown by the table III and the delimitation of the context considered previously, is α= (*Licensed Range*) because it belongs to the `Range` dimension *R*, being a part of analysis dimension and its value *Licensed Range* belongs to the `Range` domain and is repeated each odd quarter of 2007. Or α= (77500) because it belongs to the `sold quantity` measure *Q*, being a part of analysis measure and its value 77500 belongs to the `sold quantity` domain and is repeated each odd quarter of 2007.

Definition 3: ***Measure-Based Inter-dimensional Cyclic Itemset***: A measure-based inter-dimensional cyclic itemset *I* defined on $\mathscr{D}_A$ = $\{\mathscr{D}_1,...,\mathscr{D}_k\}$ or $\mathscr{M}_A$ = $\{\mathscr{M}_1,...,\mathscr{M}_j\}$ is a nonempty set of items I = $\{\alpha_1,...,\alpha_t\}$ with $\alpha_1 \in \mathscr{D}_A$ and $\forall$ π $\in$ [2, θ], α_π is a measure-based uni-dimensional cyclic item defined on $\mathscr{D}_A$ and/or $\mathscr{M}_A$ and $\forall$ π, e $\in$ [1, θ], $\alpha_\pi \neq \alpha_\theta$.

Example 1: An example of measure-based inter-dimensional cyclic itemset is *I*:[*L* = Jordan, *R* = Licensed Range,*Q* = 77500] because it is composed of three measure-based uni-dimensional cyclic items *i.e.*, α_1=(Jordan), α_2=(Licensed Range) and α_3=(77500). It is repeated quarterly.

C. Support computation of measure-based uni-dimensional cyclic item and measure-based inter-dimensional itemset using appropriate aggregate functions

Definition 4: ***The Support of measure-based uni-dimensional cyclic Item***: The support of the measure-based uni-dimensional cyclic item α, denoted $Supp(\alpha)$ is computed as follows:

- If α does not belong to one of the analysis measure: the support is equal to the number of tuples that contain the item; $Supp(\alpha) = \frac{COUNT(\alpha)}{COUNT(ALL)}$;
- If α belongs to one of the analysis measure $\mathscr{M}_l$, the support is computed in the respect of:
  1) $\mathscr{M}_l \in \mathscr{M}_{SUM}$, the support of α is the quotient of the value of α and the sum of values belonging to the current measure because the latter is aggregating using the SUM function; $Supp(\alpha) = \frac{\alpha}{SUM(\mathscr{M}_l)}$;
  2) $\mathscr{M}_l \in \mathscr{M}_{MAX}$, $Supp(\alpha) = \frac{\alpha}{MAX(\mathscr{M}_l)}$; the support of α is the quotient of the value of α and the maximum of values belonging to the current measure because the latter is aggregated using the MAX function;
  3) $\mathscr{M}_l \in \mathscr{M}_{MIN}$, the support of α is the quotient of the difference between the value of α and the minimum value of the measure, and the difference between the maximum and the minimum of the current measure: $Supp(\alpha) = \frac{\alpha - MIN(\mathscr{M}_l)}{MAX(\mathscr{M}_l) - MIN(\mathscr{M}_l)}$;
  4) $\mathscr{M}_l \in \mathscr{M}_{AVG}$
     - If $(\alpha \leq AVG(\mathscr{M}_l))$, if α belongs to the measure aggregating using the AVG function and the value of α is lower than the average value of the current measure, the support of α is the quotient of the value of α and the average of values belonging to the current measure; $Supp(\alpha) = \frac{\alpha}{AVG(\mathscr{M}_l)}$;
     - else the support of α is the quotient of the difference between the value of α and the minimum value, and the difference between the maximum and minimum of the current measure; $Supp(\alpha) = \frac{\alpha - MIN(\mathscr{M}_l)}{MAX(\mathscr{M}_l) - MIN(\mathscr{M}_l)}$.

In our running example referring to the table III, we consider the following items:

- (α_1=77500) $\in$ *Q* and $Q \in \mathscr{M}_{MAX}$, α_1 has a support

equal to $Supp(\alpha_1) = \frac{\alpha_1}{MAX(\mathscr{M}_l)} = \frac{\alpha_1}{MAX(Q)} = \frac{77500}{80000} = 0.968$

- (α_2=700) $\in$ ET and $ET \in \mathscr{M}_{MIN}$, α_2 has an a support equal to $Supp(\alpha_2) = \frac{\alpha_2 - MIN(\mathscr{M}_l)}{MAX(\mathscr{M}_l) - MIN(\mathscr{M}_l)} = \frac{\alpha_2 - MIN(ET)}{MAX(ET) - MIN(ET)} = \frac{700-600}{700-600} = 1$;
- (α_3=1,382%) $\in$ ER and $ER \in \mathscr{M}_{AVG}$ with AVG(ER)=$\frac{(1,947*4)+(1,382*2)}{6} = 1,758$.
  α_3=1,382% $< AVG(ER)$ =1,758 so α_3 has a support equal to $Supp(\alpha_3) = \frac{\alpha_3}{AVG(ER)} = \frac{1,382}{1,758} = 0.786$.

Definition 5: ***The Support of Measure-Based Inter-dimensional Cyclic Itemset***: Let's consider a measure-based inter-dimensional cyclic itemset $I = \{\alpha_1, ..., \alpha_p ..., \alpha_n\}$ composed p dimensions of $\mathscr{D}_A$ and/or $n-p$ measures of $\mathscr{M}_A$. The support of I, denoted $Supp(I)$ is decomposed into the support of measure-based uni-dimensional items belonging to p dimensions and computed using the following formula $\frac{COUNT(\alpha_1 \bigcup \alpha_p)}{COUNT(\mathscr{D}_A = ALL)}$ and the ones belonging to $n-p$ measures and obtained using the following formula $\prod_{i=p+1}^{n} SUPP(\alpha_i)$. $Supp(I) = \frac{COUNT(\alpha_1 \bigcup \alpha_p)}{COUNT(\mathscr{D}_A = ALL)} * \prod_{i=p+1}^{n} SUPP(\alpha_i)$;

Example 2: The measure-based inter-dimensional cyclic itemset I=[L = *Jordan*, R = *Licensed Range*,Q = 77500] has a support related to the sales of 77500 units of the *Licensed Range* sold in *Jordan* $Supp(I) = \frac{COUNT(L=Jordan, R=Licensed\ Range)}{COUNT(\mathscr{D}_A = ALL)} * \prod_{i=2}^{3} SUPP(Q = 77500) = \frac{COUNT(L=Jordan, R=Licensed\ Range)}{COUNT(\mathscr{D}_A = ALL)} * SUPP(Q = 77500) = \frac{4}{6} * \frac{Q=77500}{MAX(Q)} = \frac{4}{6} * \frac{77500}{80000} = 0.666 * 0.968 = 0.645$.

D. Support and Confidence of Measure-Based Inter-dimensional Cyclic Rule

Definition 6: ***Support of Measure-Based Inter-dimensional Cyclic Rule***: The rule support $\mathscr{R} : X \Rightarrow Y$, denoted $Supp(\mathscr{R})$, is equal to the ratio of the support of X and Y to the total number of tuples in the subcube.

$$Supp(\mathscr{R}) = \frac{SUPP(X \cup Y)}{SUPP(ALL, ALL)};$$

The support of de $\mathscr{R}$, $Supp(\mathscr{R}) \in [0, 1]$.

Definition 7: ***Confidence of Measure-Based Inter-dimensional Cyclic Rule***: The rule confidence $\mathscr{R} : X \Rightarrow Y$, denoted $conf(\mathscr{R})$, is equal to the ratio of the number of tuples that contain X and Y to the number of tuples that contain X in the subcube.

$$conf(\mathscr{R}) = \frac{Supp(\mathscr{R})}{Supp(X)};$$

The confidence of $\mathscr{R}$, $conf(\mathscr{R}) \in [\ 0,\ 1\ ]$.

In our running example, the rule $\mathscr{R}$: R = Licensed Range $\Rightarrow$ $Q = 77500$ has : $Supp(\mathscr{R}) = Supp(R = LicensedRange \cup Q = 77500) = \frac{COUNT(R=Licensed\ Range)}{COUNT(ALL)} * SUPP(Q = 77500) = \frac{4}{6} * \frac{77500}{MAX(Q)} = \frac{4}{6} * \frac{77500}{80000} = 0.666 * 0.968 = 0.645$. $Conf(\mathscr{R}) = \frac{Supp(R)}{Supp(Y)} = \frac{Supp(R)}{Supp(R=Licensed\ Range)} = \frac{0.645}{0.666} = 0.429$.

IV. Mining Measure-Based Inter-dimensional Cyclic Association Rules

Starting from a data cube, we propose the following two phases to generate measure-based inter-dimensional cyclic association rules:

- ***Pre-processing phase*** contains two steps which are : *(i)* the derivation of the endogen sub-cube based on the user-specification of the endogen context using an SQL query; *(ii)* the discretization of continuous valued measures within the equal frequency discretization technique;
- ***Processing phase*** consists on the mining of measure-based inter-dimensional CAR from data cube.

In what follows, these phases are detailed.

A. Pre-processing phase

Date	Location	Range	Sold Quantity	Exchange Rate	External Turnover
D	*L*	*R*	*Q*	*ER*	*ET*
Q1 of 2007	Jordan	Licensed Range	75000-80000	1,946%- 1,948%	450-750
Q2 of 2007	Jordan	Licensed Range	75000-80000	1,946%- 1,948%	450-750
Q3 of 2007	Jordan	Licensed Range	75000-80000	1,946%- 1,948%	450-750
Q4 of 2007	Jordan	Licensed Range	75000-80000	1,946%- 1,948%	800-1000
Q1 of 2008	Yemen	Model Hospital	71100-75000	1,38%- 1,382%	450-750
Q2 of 2008	Yemen	Model Hospital	75000-80000	1,38%- 1,382%	800-1000

TABLE II

TABLE $\mathscr{T}$ WITH A DISCRETIZATION OF MEASURES USING **EQUAL FREQUENCY** TECHNIQUE

Based on the user-specification of the endogen context, an SQL query can be launched to derive the endogen sub-cube in the respect of the selected analysis dimensions and measures and restrictions on reference dimensions and measures.

Time	Location	Range	Sold Quantity	Exchange Rate	External Turnover
T	*L*	*R*	*Q*	*ER*	*ET*
Q1 of 2007	Jordan	Licensed Range	77500	1,947%	600
Q2 of 2007	Jordan	Licensed Range	77500	1,947%	600
Q3 of 2007	Jordan	Licensed Range	77500	1,947%	600
Q4 of 2007	Jordan	Licensed Range	77500	1,947%	700
Q1 of 2008	Yemen	Model Hospital	73050	1,381%	600
Q2 of 2008	Yemen	Model Hospital	77500	1,381%	700

TABLE III

TABLE $\mathscr{T}$ OBTAINED AFTER THE PREPROCESSING PHASE

To generate the measure-based inter-dimensional cyclic association rules from the endogen sub-cube, a discretization of the context measures is compulsory. We choose in our context the equal frequency technique for many reasons, namely, the intervals are created so that, roughly, the frequency of each interval is constant and each interval contains roughly the same number of contiguous data samples. In our case, the expert specify the depth to forty otherwise the forty nearest values will compose an independent interval. For example, 77520, 75000, 77520, 80000 and thirty six other values belonging to the range of 75000 and 80000 in the `sold quantity` measure compose the interval [75000-80000].

According to the same method, the obtained sub-cube after the discretization step is depicted by the table II. Once we obtain the discretized intervals, we use the median value for each interval related to continuous valued measure so that we can involve the summarizability of measures on the mining process. In fact, the median value can be computed using the following formula $V_{median} = \frac{V_{max} + Vmin}{2}$. For example, in our case (table II), the considered item for the `sold quantity` measure is $\alpha = \frac{V_{max} + Vmin}{2} = \frac{75000+80000}{2} = 77500$. The output table is shown by the table III which is the input of our method.

B. Mining phase: Extraction of Measure-based Inter-dimensional Cyclic Association Rules Using Appropriate Aggregate Functions

After deriving the endogen sub-cube and discretization of valued measures and dimensions and the transformation of valued analysis measures using the median of each interval, the MEasure-based inter-Dimensional Cyclic Association Rules using Aggregate functions algorithm, called MEDCAR takes as input the decortication of the analysis measures according to the appropriate aggregate functions to sets, the set of the minimum threshold of support $Minsupp$ and the minimum threshold of confidence $MinConf$ and the length of cycle. It outputs the list of measure-based inter-dimensional CAR. The used notations are depicted by table IV and its pseudo-code is illustrated by the algorithm in the following. In fact, MEDCAR, an iterative process, operates in three successive steps. First, we proceed by an increasing level wise search for measure-based inter-dimensional cyclic large i-itemsets, where the level (i) designs the number of items in the set.

Notation	Description
SC	Endogen Sub-cube
α	Potential measure-based uni-dimensional cyclic item
I	Measure-Based Inter-dimensional cyclic itemset
$\mathscr{C}_i$ (resp.$\mathscr{F}_i$)	Set of candidates (resp. frequent) measure-based inter-dimensional cyclic i-itemsets.
$\mathscr{M}insupp$	Minimum support threshold
$\mathscr{M}inconf$	Minimum confidence threshold
$\mathscr{D}_t$	Date t
l	Length of cycle
$Supp(\mathscr{C})$	Support of the measure-based inter-dimensional cyclic itemset $\mathscr{C}$
$\mathscr{M}_{MAX}$	set of measures that can be aggregated using MAX
$\mathscr{M}_{AVG}$, $\mathscr{M}_{SUM}$)	function(resp. $\mathscr{M}_{MIN}$ (resp. MIN, AVG, SUM)

TABLE IV

LIST OF USED NOTATIONS IN THE MEDCAR ALGORITHM.

We denote by $C(i)$ the measure-based inter-dimensional cyclic candidate i-itemsets potentially frequent, and $F(i)$ the measure-based inter-dimensional cyclic frequents i-itemsets. For each level (i), if the set $C(i)$ is not empty, the first step of our algorithm derives the measure-based inter-dimensional frequent cyclic patterns $F(i)$ from $C(i)$ with respect of two conditions: (C1) A measure-based inter-dimensional cyclic itemset $A \in C(i)$ must be a conjunction of members from analysis dimensions or/and analysis measures; (C2) and a measure-based inter-dimensional cyclic itemset must have a support above the $Minsupp$ to be included in $F(i)$. For an efficient extraction of measure-based inter-dimensional cyclic frequent itemsets, we exploit the anti-monotonicity property of the support in the multi-dimensional context. Indeed, any subset of a frequent cyclic set is frequent cyclic and any cyclic itemset failing to be frequent, all its supersets will not be frequent so they will be pruned. The second step uses the measure-based inter-dimensional cyclic large i-itemsets $F(i)$ to derive a new set $C(i+1)$ of $(i+1)$-candidates. A *(i + 1)*-candidate is formed by the union of two measure-based inter-dimensional cyclic i-itemsets A and B from $F(i)$ according to three conditions: (C1) A and B must have (i - 1) common measure-based uni-dimensional cyclic items; (C2) all measure-based inter-dimensional cyclic sub-itemsets A U B must be instances of $\mathscr{D}_A$ and/ or $\mathscr{M}_A$; (C3) and all nonempty measure-based inter-dimensional cyclic sub-itemsets of A U B must be measure-based inter-dimensional cyclic frequent itemsets.

Algorithm 1: MEDCAR: MEasure-based inter-Dimensional Cyclic Association Rules using Aggregate functions

Data: SC, $\mathscr{M}insupp$,$\mathscr{M}inconf$, l, $\mathscr{M}_{MAX}$,$\mathscr{M}_{MIN}$,$\mathscr{M}_{SUM}$, $\mathscr{M}_{AVG}$
Result: $\mathscr{R}$: Measure-based Inter-dimensional cyclic rules in SC using Aggregate functions

begin
  $\mathscr{F}_1$= *Find 1-frequent cyclic itemsets* (SC, l,$\mathscr{D}_t$, $\mathscr{M}$inSupp) ;
  for *(k=2; $\mathscr{F}_k \neq \emptyset$; k++)* **do**
    $\mathscr{C}_k$ = *CandidatGeneration*($\mathscr{F}_{k-1}$);
    if *$\mathscr{C}_k$ is a inter-dimensional cyclic itemset* **then**
      foreach *transaction $\mathscr{T} \in SC$ at date $\mathscr{D}_t$* **do**
        $\mathscr{C}_t$=subset($\mathscr{C}_k$, $\mathscr{T}$)
        foreach *candidat $\mathscr{C} \in \mathscr{C}_t$* **do**
          $\mathscr{C}$.count = *SupportComputing*(SC, l,$\mathscr{D}_t$, $\mathscr{C}$);
          $\mathscr{F}_k$ = { $\mathscr{C} \in \mathscr{C}_k$, $\mathscr{C}$.count > $\mathscr{M}insupp$ }
  Return $\mathscr{F}_k = \cup_k \mathscr{F}_k$;
  // *RuleGeneration*
  for *(i=2; i<k; i++)* **do**
    Generate All nonempty subset of $\mathscr{F}_i$;
    foreach *nonempty subset s of $\mathscr{F}_i$* **do**
    $r = s \rightarrow (\mathscr{F}_i\text{-}s)$;
    if *(confidence(r) > $\mathscr{M}inconf$)* **then**
      $\mathscr{R} = \mathscr{R} \cup r$;
  Return $\mathscr{R}$;
end
Procedure *Find 1-frequent cyclic itemsets* (SC, l,$\mathscr{D}_t$, $\mathscr{M}$inSupp)
Result: $\mathscr{F}_1$
begin
  while *(!End of tuples in SC)* **do**
    foreach *transaction $\mathscr{T} \in SC$* **do**
    foreach *item $\alpha \in \mathscr{T}$* **do**
    foreach *transaction $\mathscr{T}' \in SC$ at date $\mathscr{D}_{t+l}$* **do**
    // $\alpha \in M_k$
    if *(α exists in $\mathscr{T}'$) and ($M_k \in \mathscr{M}_A$)* **then**
      if $M_k \in \mathscr{M}_{SUM}$ **then**
        $Supp(\alpha)=\frac{\alpha}{SUM(M_k)}$; // with m the current measure
      if $M_k \in \mathscr{M}_{MAX}$ **then**
        $Supp(\alpha)=\frac{\alpha}{MAX(M_k)}$;
      if $M_k \in \mathscr{M}_{MIN}$ **then**
        $Supp(\alpha)=\frac{\alpha-MIN(M_k)}{MAX(M_k)-MIN(M_k)}$;
      if $M_k \in \mathscr{M}_{AVG}$ **then**
        if $\alpha >= AVG(M_k)$ **then**
          $Supp(\alpha)=\frac{\alpha}{AVG(M_k)}$;
        $Supp(\alpha) = \frac{\alpha-MIN(M_k)}{MAX(M_k)-MIN(M_k)}$;
    if *(α exists in $\mathscr{T}'$) and ($\alpha \in \mathscr{D}_A$)* **then**
      $Supp(\alpha)$=COUNT(α);
    if *($Supp(\alpha) > \mathscr{M}inSupp$)* **then**
      $\mathscr{F}_1 = \mathscr{F}_1 \cup \alpha$;
  Return $\mathscr{F}_1$;
end
Procedure *SupportComputing* (SC, l,$\mathscr{D}_t$, $\mathscr{C}$)
Result: $Supp(\mathscr{C})$
begin
  NoMoreCyclic: Boolean;
  NoMoreCyclic = false;
  while *((!End of tuples in SC)* ***and*** *(!NoMoreCyclic))* **do**
    $\mathscr{C}_k$ = *CandidatGeneration*($\mathscr{C}_{k-1}$);
    foreach *transaction $\mathscr{T} \in SC$ at date $\mathscr{D}_{t+l}$* **do**
    if *$\mathscr{C}$ exists in $\mathscr{T}$* **then**
      if *Itemset I of $\mathscr{C}$ contains $\mathscr{M}_A$* **then**
        $Supp(\mathscr{C})$=*COUNT*($\mathscr{C}$ - *I*) $\prod_{i=1}^{n} Supp(i)$.
      $Supp(\mathscr{C})$=COUNT($\mathscr{C}$);
    NoMoreCyclic = true;
  Return $Supp(\mathscr{C})$;
end

Finally, the third stage consists on scanning $F(i)$ level by level. From every $A \in F(i)$, we extract the measure-based

inter-dimensional CAR with respect to condition, *i.e.*, having a confidence above the minimum confidence threshold *MinConf*.

V. Experimental results

Experiments were conducted on a Pentium PC with 1.73 GHz and 1 GB of main memory. In the following, we report experiments performed on a real sales data warehouse [1], which contains three dimensions (e.g., `Time` dimension, `Range` dimension, `Location` dimension) and one sales fact table. The data warehouse is built using relational OLAP (ROLAP). Figure 1.(a) shows the behavior of our approach when the frequency interval changes. In fact, the increase of the frequency intervals, decreases the performance of our method because if we rise the frequency, the number of measure-based inter-dimensional cyclic candidates itemsets will highly grow leading to an increase of the measure-based inter-dimensional cyclic frequent itemsets. Figure 1.(b) presents a test of our algorithm on varying the length of cycle. For small support values, the running time considerably increases by decreasing the length of cycle. However, for large supports, the length of cycle has a less effect on the performance of our algorithm. In the figure 1.(c), we clearly notice that the efficiency of the algorithm closely depends on the number of dimensions involved during the mining process. Similarly, the running time obviously increases according to the number of involved measures as shown by figure 1.(d). When the number of dimensions (respectively measures) is high, the number of measure-based inter-dimensional cyclic frequent itemsets mined among several candidates is high. On the contrary, when this number of dimensions (respectively measures) is low, the potential measure-based inter-dimensional cyclic frequent itemsets is also low).

Interestingly enough, these experiments highlight acceptable runtime processing. The efficiency of our algorithm is due to: (i) the use of constraint based mining through the choice of dimensions and measures analysis which highly reduce the search space of CAR and therefore, considerably decreases the runtime of the mining process; (ii) the use of the Apriori property which is definitely suited to sparse data cubes and considerably reduces the complexity of measure-based inter-dimensional cyclic large itemsets search. The usefulness of our proposal is motivated by coupling dimensions and measures.

VI. Conclusion

In this paper, we proposed a new method to extract measure-based cyclic association rules taking into consideration the summarizability of measures during the evaluation of the generated measure-based inter-dimensional patterns from the OLAP context. Thus, a new definition of measure-based inter-dimensional cyclic patterns is provided. A redefinition of their evaluation metrics inspired from the summarizability characteristic of involved measures is presented using the relevant aggregation functions characterizing such a measure. Accordingly, a new method called MEDCAR to extract such patterns is introduced.

Other avenues for future work mainly address the following issues: (i) use of other discretization technique to guarantee a lossless information for measure, (ii) discovery of unexpected cyclic patterns in OLAP context, (iii) mining of measure-based inter-dimensional cyclic patterns under time constraint such as the integration of excluded intervals.

[1]The data warehouse is related to pharmaceutical listed company. It is built using the available information at http ://www.bvmt.com.tn/companies/ ?view=listed.

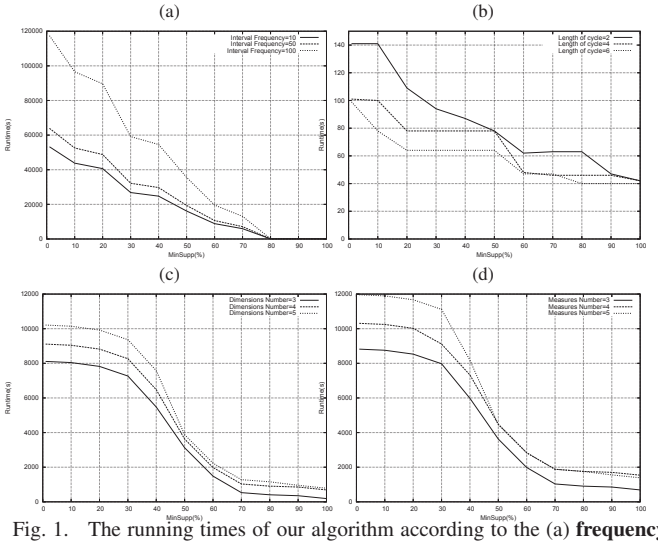


Fig. 1. The running times of our algorithm according to the (a) **frequency interval**, (b) **length of cycle**, (c) **number of dimensions**, (d) **number of measures**.

References

[1] E.Ben Ahmed and M.S.Gouider, "Towards a new mechanism of extracting cyclic association rules based on partition aspect", *IEEE International Conference on RCIS*, pp 69–78,2010.

[2] E.Ben Ahmed, A.Nabli and F.Gargouri, "Usage Des Mesures Pour La Génération Des Règles d'Associations Cycliques", *Conférence francophone sur les entrepôts de données et l'analyse en ligne*,2011.

[3] E.Ben Ahmed and F.Gargouri, "Règles d'association cycliques dans un contexte multidimensionnel", *Atelier des Systmes Décisionnels (ASD'10)*, Tunisia, 2010.

[4] R.Ben Messaoud, O.Boussaid, S.L Rabasda and R.Missaoui, "Enhanced mining of association rules from data cubes", *Proceedings of the 9 th ACM International Workshop on Data Warehousing and OLAP (DOLAP 2006)*, pp 11–18, 2006.

[5] D.Chiang, C.Wang, S.Chen and C.Chen, "The Cyclic Model Analysis on Sequential Patterns", *IEEE Trans. on Knowl. and Data Eng.*, pp 1617–1628, 2009.

[6] J.Han, W.Gong and Y.Yin "Efficient Mining of Partial Periodic Patterns in Time Series Database", *ICDE*, pp 106–115,1999.

[7] M.Kamber, J.Han and J.Y.Chiang "Metarule-guided mining of multi-dimensional association rules using data cubes", *Proceedings of the KDD'97* pp 207–210,1997.

[8] H.J.Lenz and A.Shoshani, "Summarizability in OLAP and Statistical Data Bases", pp 132–143, 1997.

[9] B.Ozden, S.Ramaswamy and A.Silberschatz , "Cyclic Association Rules", *Proceedings of the Fourteenth International Conference on Data Engineering* pp 412–421, 1998.

[10] S.Palaniappan and T.K.Hong, "Discretization of Continuous Valued Dimensions in OLAP Data Cubes", *International Journal of Computer Science and Network Security*, pp 116–126, 2008.

[11] M.Plantevit, A.Laurent and M.Teisseire, "Motifs séquentiels multidimensionnels et mesure:Différentes techniques pour calculer le support", *Ingénierie des Systèmes d'Information*, pp9–32, 2008.

[12] M.Plantevit, A.Laurent, D.Laurent, M.Teisseire and Y.Choong "Mining multidimensional and multilevel sequential patterns", *ACM Transactions on Knowledge Discovery from Data*, pp 155–174,2010.

[13] N.D.Thuan, "Mining Cylic Association Rules in Temporal Database ", *The Journal Science and technology developement, Vietnam National University* , pp 12–19, 2004.

[14] N.D.Thuan, "Mining Time Pattern Association Rules in Temporal Database", *SCSS*, pp 7-11, 2008.

Measuring Similarity in Large-scale Folksonomies

Giovanni Quattrone[1], Emilio Ferrara[2], Pasquale De Meo[3], Licia Capra[1]
[1]*Dept. of Computer Science, University College London, UK*
[2]*Dept. of Mathematics, University of Messina, IT*
[3]*Dept. of Physics, Informatics Section, University of Messina, IT*
E-mail: {g.quattrone,l.capra}@cs.ucl.ac.uk; {eferrara,pdemeo}@unime.it

Abstract

Social (or folksonomic) tagging has become a very popular way to describe content within Web 2.0 websites. Unlike taxonomies, which overimpose a hierarchical categorisation of content, folksonomies enable end-users to freely create and choose the categories (in this case, tags) that best describe some content. However, as tags are informally defined, continually changing, and ungoverned, social tagging has often been criticised for lowering, rather than increasing, the efficiency of searching, due to the number of synonyms, homonyms, polysemy, as well as the heterogeneity of users and the noise they introduce. To address this issue, a variety of approaches have been proposed that recommend users what tags to use, both when labelling and when looking for resources. As we illustrate in this paper, real world folksonomies are characterized by power law distributions of tags, over which commonly used similarity metrics, including the Jaccard coefficient and the cosine similarity, fail to compute. We thus propose a novel metric, specifically developed to capture similarity in large-scale folksonomies, that is based on a mutual reinforcement principle: that is, two tags are deemed similar if they have been associated to similar resources, and vice-versa two resources are deemed similar if they have been labelled by similar tags. We offer an efficient realisation of this similarity metric, and assess its quality experimentally, by comparing it against cosine similarity, on three large-scale datasets, namely Bibsonomy, MovieLens and CiteULike.

1. Introduction

The rise of Web 2.0 has transformed users from passive consumers to active producers of content. This has exponentially increased the amount of information that is available to users, from videos on sites like YouTube and MySpace, to pictures on Flickr, music on Last.fm, blogs on Blogger, and so on. This content is no longer categorised according to pre-defined taxonomies (or ontologies). Rather, a new trend called *social* (or *folksonomic*) *tagging* has emerged, and quickly become the most popular way to describe content within Web 2.0 websites. Unlike taxonomies, which overimpose a hierarchical categorisation of content, folksonomies empower end users by enabling them to freely create and choose the tags that best describe a piece of information (a picture, a blog entry, a video clip, etc.). However, this freedom comes at a cost: since tags are informally defined, continually changing, and ungoverned, finding content of interest has become a main challenge, because of the number of synonyms, homonyms, polysemy, as well as the inevitable heterogeneity of users and the noise they introduce.

In order to assist users finding content of their own interest within this information abundance, new techniques, inspired by traditional recommender systems, have been developed: for example, whenever a user searches from some content using query tags $\{t_1, \ldots, t_m\}$, new tags $\{t_{m+1}, \ldots, t_{m+n}\}$ are being added to the query, based on their *similarity* to their original query tags. This is done to increase the chances of finding content of relevance in these extremely sparse settings. Various metrics have been used to compute the similarity among folksonomy entities, including, for instance, cosine similarity, Jaccard coefficient, and Pearson Correlation. Performance results demonstrate an increase in accuracy and coverage of searches when using these techniques; however, evaluation has been conducted on manipulated datasets so to obtain a much denser one. We argue that such manipulations alter the nature of real folksonomies, and indeed eliminate the problem, rather than solving it.

Unmodified real-world folksonomies are characterized by two key properties: the *power law* distribution of tags, and the *non-independence* of data. Empirical studies [4, 5] illustrate that tag usage in folksonomies follows a power law distribution; this means that, if we were select any two tags, the probability that the resources jointly labelled by them is non-zero is extremely low. As a result, comput-

ing tag similarity on un-modified folksonomies, using traditional metrics like cosine similarity, would almost always yield close-to-zero values, thus failing to support users in retrieving resources relevant to their queries. Furthermore, metrics like cosine assume that tags are semantically independent of each other; once again, this assumption does not hold in real folksonomies, where tags may be synonyms to each other.

In this paper, we propose a novel similarity metric that can be used to accurately quantify tag similarity in large-scale real-world folksonomies (Section 3). This similarity metric is computed following an iterative algorithm, grounded on a *mutual reinforcement* principle: that is, two tags are similar if they label similar resources, and vice-versa, two resources are similar if they have been labelled by similar tags. We describe an efficient realisation of this similarity metric (Section 4), and empirically quantify its quick convergence on three large-scale datasets, namely BibSonomy[1], MovieLens[2], and CiteULike[3]. We measure Precision and Recall of our metric, and compare it to cosine similarity on these unprocessed datasets (Section 5). Our findings demonstrate that, when considering our unmanipulated datasets, the performance of our novel similarity metric provides higher Precision and Recall w.r.t. the cosine similarity. Section 6 covers related works on similarity measures, mainly applied to folksonomies. Finally, in Section 7 we draw our conclusions.

2. Background

In this section, we formally introduce some concepts that will be extensively used in the following, when presenting our approach. The first concept we consider is that of a folksonomy [12]:

Definition 2.1 *Let $US = \{u_1, \ldots, u_{n_u}\}$ be a set of users, let $RS = \{r_1, \ldots, r_{n_r}\}$ be a set of resource URIs and let $TS = \{t_1, \ldots, t_{n_t}\}$ be a set of tags. A* folksonomy *F is a tuple $F = \langle US, RS, TS, AS \rangle$, where $AS \subseteq US \times RS \times TS$ is a ternary relationship called* tag assignment set.

In this definition we do not make any assumption about the nature of resources; they could be URLs (like in Delicious), photos (as in Flickr), music files (as in Last.fm), documents (as in CiteULike), and so on.

According to Definition 2.1, a folksonomy F is a "three-dimensional" data structure whose "dimensions" are represented by users, tags and resources. In particular, an element $a \in AS$ is a triple $\langle u, r, t \rangle$, indicating that user u labelled resource r with tag t. To simplify modeling and management of folksonomies, their inherent tripartite graph structure is often mapped into three *matrices*, whereby each matrix models one relationship at a time (i.e., between tags and resources, tags and users, and resources and users) [19]. In this paper, we adopt the same matrix-based representation. Specifically, being n_r, n_t and n_u the number of resources, tags and users respectively, we represent a folksonomy as the following three matrices:

- $\mathbf{TR}$ (Tag-Resource): a $n_t \times n_r$ matrix such that $\mathbf{TR}_{ij}$ is the number of times the tag i labelled resource j;
- $\mathbf{TU}$ (Tag-User): a $n_t \times n_u$ matrix such that $\mathbf{TU}_{ij}$ is the number of times the tag i has been used by user j;
- $\mathbf{RU}$ (Resource-User): a $n_r \times n_u$ matrix such that $\mathbf{RU}_{ij}$ is the number of times resource i has been labelled by the user j.

Tag similarity within a folksonomy can then be computed by looking at the resources these tags have been attached to. In particular, each tag t_i can be mapped onto a vector $\mathbf{t}_r(i)$ corresponding to the i-th row of $\mathbf{TR}$. Given an arbitrary pair of tags t_i and t_j, their similarity $s(t_i, t_j)$ can be computed as the *cosine similarity* (CS) of the vectors $\mathbf{t}_r(i)$ and $\mathbf{t}_r(j)$:

$$s(t_i, t_j) = \frac{\langle \mathbf{t}_r(i), \mathbf{t}_r(j) \rangle}{\sqrt{\langle \mathbf{t}_r(i), \mathbf{t}_r(i) \rangle} \sqrt{\langle \mathbf{t}_r(j), \mathbf{t}_r(j) \rangle}} \quad (1)$$

being $\langle \cdot, \cdot \rangle$ the usual *inner product* in $\mathbb{R}^{n_r}$.

Cosine similarity has been successfully applied in the context of Information Retrieval [16]. Within a folksonomy, Equation 1 states that the similarity score of a pair of tags is high if they *jointly co-occur* in labelling the same subset of resources. However, two key properties of folksonomies, that are, *(i)* the power law distribution of tags and *(ii)* their non-independence, cause Equation 1 to yield very poor results in this domain, as we shall discuss next.

Power Law in Tag Distribution. Let us consider a real-world folksonomy like BibSonomy. BibSonomy [11, 13] is a social bookmarking service in which users are allowed to tag both URLs and scientific papers. A power law distribution of tags on scientific references emerges. In particular, resources were described by no more than 5 different tags (roughly 81%), and usually less than 3 (roughly 58%). A small portion of frequently adopted tags used to bookmark scientific references, and a long tail of tags (roughly 81%) being used less than 5 times.

Following the above observations, matrix $\mathbf{TR}$ is rather sparse; thus, if we were to select any pair of tags t_i and t_j, most of the components of the corresponding vectors $\mathbf{t}_r(i)$ and $\mathbf{t}_r(j)$ would be 0 and, therefore their inner product would be close to 0. The cosine similarity between *any* t_i and t_j would therefore be almost 0, regardless of the initial

[1] http://www.bibsonomy.org/
[2] http://www.movielens.org/
[3] http://www.citeulike.org/

choice of t_i and t_j. Such counter-intuitive result is an effect of the inadequacy of cosine similarity to capture properties of tags in large-scale real folksonomies.

Non-Independence of Tags. Cosine similarity implicitly assumes that the components of the vectors appearing in Equation 1 are *independent* of each other. Such an assumption does not often hold true. For instance, consider a folksonomy consisting of two resources r_1 and r_2, representing two different scientific papers, both discussing about folksonomies. Suppose that the paper associated with r_2 is an extension of the paper associated with r_1. Finally, assume to bookmark the resource r_1 with the tag t_1 = "folksonomy" and to bookmark the resource r_2 with the tag t_2 = "social tagging". In this case, the similarity between t_1 and t_2 computed according to Equation 1 would be 0, even if t_1 and t_2 should result similar each other. The mutual similarity between t_1 and t_2 can be assessed only if we consider the non-independence of the resources they label.

3. Approach Description

In this section, we present a new definition of tag (and resource) similarity, that is particularly suited to quantify similarity of elements (be them tags of resources) in datasets characterized by power law distribution and non-independence of data. Our definition of similarity relies on the *mutual reinforcement principle*:

> *Two tags are similar if they label similar resources, and conversely, two resources are similar if they are labelled by similar tags.*

In the following, we shall derive a mathematical formula to compute tag and resource similarity on the basis of the principle stated above. After this, we shall illustrate why our formula is able to effectively address the power law and non-independence challenges.

We designed an *iterative algorithm* to compute the similarity score. In the base case, given a pair of tags $\langle t_a, t_b \rangle$ and a pair of resources $\langle r_a, r_b \rangle$, we define the *tag similarity* $st^0(t_a, t_b)$ and the *resources similarity* $sr^0(r_a, r_b)$ as follows:

$$st^0(t_a, t_b) = \delta_{ab} \qquad sr^0(r_a, r_b) = \delta_{ab} \tag{2}$$

being δ_{ab} the *Kronecker symbol*[4]. Equation 2 reflects the fact that, in the initial step, each tag (resp., resource) is similar only to itself and it is dissimilar to all other tags (resp., resources).

At the k-th step, let $st^{k-1}(t_a, t_b)$ (resp., $sr^{k-1}(r_a, r_b)$) be the tag (resp., resource) similarity between the tags t_a and t_b (resp., resources r_a and r_b). We apply the following rules to update $st^{k-1}(t_a, t_b)$ (resp., $sr^{k-1}(r_a, r_b)$):

$$st^k(t_a, t_b) = \frac{ST^k(t_a, t_b)}{\sqrt{ST^k(t_a, t_a)} * \sqrt{ST^k(t_b, t_b)}} \tag{3}$$

$$sr^k(r_a, r_b) = \frac{SR^k(r_a, r_b)}{\sqrt{SR^k(r_a, r_a)} * \sqrt{SR^k(r_b, r_b)}} \tag{4}$$

where:

$$ST^k(t_a, t_b) = \sum_{i,j=1}^{n_r} \mathbf{TR}_{ai} * \Psi_{ij} * sr^{k-1}(r_i, r_j) * \mathbf{TR}_{bj} \tag{5}$$

$$SR^k(r_a, r_b) = \sum_{i,j=1}^{n_t} \mathbf{TR}_{ia} * \Psi_{ij} * st^{k-1}(t_i, t_j) * \mathbf{TR}_{jb} \tag{6}$$

Here Ψ_{ij} is equal to 1 if $i = j$ and it is equal to ψ if $i \neq j$, where ψ (called *propagation factor*) is a value belonging to the interval $[0, 1] \in \mathbb{R}$.

Equations 3–4 rely on the following intuitions. Given a pair of tags $\langle t_a, t_b \rangle$, at the k iteration, we consider *all pair of resources* $\langle r_i, r_j \rangle$ in our folksonomy and we take their similarity $sr^{k-1}(r_i, r_j)$ into account to compute $st^k(t_a, t_b)$. In particular, we compute a *weighted sum* of all the similarity values $sr^{k-1}(r_i, r_j)$, where the weights reflect the *strength of the association* between the tag t_a and the resource r_i, and the tag t_b and the resource r_j. As a consequence, the higher the similarity between r_i and r_j, the higher the contribution of the association between the tag t_a and the resource r_i, and the tag t_b and the resource r_j. Finally, the term Ψ_{ij} is instrumental to give higher relevance to tags that labelled the very *same* resources, w.r.t. the fact that they labelled two *similar* (but different) resources.

Note that, in the special case in which $\psi = 0$, our method does not depend on k and Equations 3–4 reduce to the cosine similarity formulation. In fact, in this particular case, all the contributions $sr^{k-1}(r_i, r_j)$ and $st^{k-1}(r_i, r_j)$ are disregarded when $i \neq j$, and are taken into consideration only when $i = j$. Since all contributions $sr^{k-1}(r_i, r_i)$ and $st^{k-1}(r_i, r_i)$ are equal to 1 by definition, it follows that Equations 3–4 reduce to the cosine similarity formulation.

Equations 3–4 are able to effectively address the power law and non-independence of data challenges we outlined above. In fact:

- In the computation of tag (resp., resource) similarity, we leverage on the similarity of all pairs of resource (resp., tag) similarities. As a consequence, unlike cosine similarity, we do not restrict ourselves to consider only the resources jointly labelled by two tags (resp., the tags jointly labelling two resources), which can be few, but we *iteratively propagate* similarity scores by considering all the pairs of similar resources jointly labelled by the two tags (resp., all the pairs of similar

[4]We recall that the Kronecker symbol δ_{ab} is equal to 1 if a and b coincide and 0 otherwise.

tags jointly labelling two resources). In this way we are able to face the power law occurring in tag usage.

- In our definition of similarity, if two tags label similar, *even if not coincident*, resources their similarity score will be greater than 0, whereas the cosine similarity would return 0. As a consequence, our similarity method takes into account forms of correlation among pairs of resources and/or tags rather than assuming their independence.

4. Realization

From a computational standpoint, Equations 3–4 could entail a large overhead for two reasons:

- From a theoretical standpoint, our approach may need an infinite number of iterations. As a consequence, we need a stopping criterion allowing us to safely terminate the execution of Equations 3–4 after a finite (and low) number of iterations.
- Equation 3 (resp., Equation 4) requires the computation of n_r^2 resource-resource (resp,. n_t^2 tag-tag) similarities, at each k-th step. This could make our similarity measure inapplicable in practical cases, because each iteration requires exactly $n_r^2 \times n_t^2$ computations.

Fortunately, there are two important results making our similarity measure applicable and entailing the same complexity level as cosine similarity. The first result can be stated by the theorem showed and proved in the Appendix[5] which affirms that the sequences $st^k(t_a, t_b)$ and $sr^k(r_a, r_b)$ defined as in Equations 3–4 converge.

This theorem ensures that, after a certain number of iterations, Equations 3–4 converge to stable values. During experimentation conducted on three real folksonomies (see Section 5.1), we empirically found that convergence was achieved after as little as five iterations, thus suggesting that our similarity measure is applicable in practical cases.

Furthermore, Equations 3–4 can be defined, without any loss of generality, as a simple matrix product (such as in cosine similarity). Specifically, let $\mathbf{st}^k$ and $\mathbf{sr}^k$ be the tag-tag and resource-resource similarity matrices respectively, with $\mathbf{st}^0 = \mathbf{I}_t$ and $\mathbf{sr}^0 = \mathbf{I}_r$; here $\mathbf{st}^0 = \mathbf{I}_t$ (resp., $\mathbf{sr}^0 = \mathbf{I}_r$) is the $n_t \times n_t$ (resp., $n_r \times n_r$) *identity matrix*. If we indicate with the symbol "$\circ$" the *Hadamard matrix product*[6] [7], at the k-th step, the $\mathbf{st}^k$ and $\mathbf{sr}^k$ matrices can be computed as:

$$\mathbf{st}^k = \mathbf{ST}^k \circ \mathbf{DT}^k \tag{7}$$

$$\mathbf{sr}^k = \mathbf{SR}^k \circ \mathbf{DR}^k \tag{8}$$

[5] See `http://tinyurl.com/proof-seke2011`.

[6] Given two matrices $\mathbf{A}$ and $\mathbf{B}$ of the same dimensions, the *Hadamard product* $\mathbf{A} \circ \mathbf{B}$ is a matrix of the same dimensions of $\mathbf{A}$ and $\mathbf{B}$ and it is defined as follows: $(\mathbf{A} \circ \mathbf{B})_{ij} = \mathbf{A}_{ij} \cdot \mathbf{B}_{ij}$

where:

$$\mathbf{ST}^k = \mathbf{TR} \times \left(\mathbf{\Psi_r} \circ \mathbf{sr}^{k-1}\right) \times \mathbf{TR}^t \tag{9}$$

$$\mathbf{SR}^k = \mathbf{TR}^t \times \left(\mathbf{\Psi_t} \circ \mathbf{st}^{k-1}\right) \times \mathbf{TR} \tag{10}$$

$$\mathbf{DT}^k_{ab} = \frac{1}{\sqrt{\mathbf{ST}^k_{aa}}\sqrt{\mathbf{ST}^k_{bb}}} \tag{11}$$

$$\mathbf{DR}^k_{ab} = \frac{1}{\sqrt{\mathbf{SR}^k_{aa}}\sqrt{\mathbf{SR}^k_{bb}}} \tag{12}$$

In the above equations, we have indicated with $\mathbf{\Psi}_r$ (resp., $\mathbf{\Psi}_t$) a square matrix $n_r \times n_r$ (resp., $n_t \times n_t$) where all the elements are set to ψ, with the exception of the diagonal where the elements are set to 1; the symbol $\mathbf{TR}^t$ represents the transpose of matrix $\mathbf{TR}$. We have thus reduced the computational complexity of each iterative step from $n_r^2 \times n_t^2$ to a simple matrix product; this reduction, coupled with the empirical observation that 5 iterative steps are sufficient to find convergence, makes our similarity metrics suitable in practical contexts. The last question that needs answering is how effective (in terms of Precision and Recall) our similarity metric is w.r.t. traditional ones like cosine. We answer this question next.

5. Experiments

In order to evaluate the performance of our similarity measure, we built a prototype in Java and MySQL and we conducted experiments using three well known social tagging websites: Bibsonomy, CiteULike, and MovieLens. The experiments we carried out aimed to answer the following question:

> *If we consider any two tags t_i and t_j belonging to a folksonomy, is our similarity measure capable of accurately assessing the extent to which they are related (similar) each other? And can it do so even when such tags have been drawn from the long tail of low popularity tags?*

5.1. The Dataset

To answer the above question, we conducted experiments on the following three datasets.

Bibsonomy. Bibsonomy is a social bookmarking website promoting the sharing of both scientific reference and general URL. We downloaded a snapshot of the website in June 2009, containing bookmarks made between January 1999 and June 2009.

CiteULike. CiteULike is a social bookmarking website that aims to promote and develop the sharing of scientific references amongst researchers. CiteULike enables scientists to organize their libraries with freely chosen tags which produce a folksonomy of academic interests. CiteULike

Dataset	*Users*	*Resources*	*Tags*	*Bookmarks*
Bibsonomy	4,696	578,587	147,076	648,924
CiteULike	57,053	1,928,302	401,620	2,281,609
MovieLens	4,009	7,601	15,240	55,484

Table 1. Features of our datasets

runs a daily process which produces a snapshot summary of what articles have been posted by whom and with what tags up to that day. We downloaded one such archive in November 2009, containing bookmarks made between November 2004 to November 2009.

MovieLens. MovieLens is a rate-based recommendation website that suggests to users movies they might like. We downloaded such dataset in January 2009, containing bookmarks made from December 2005 to January 2009.

Table 1 summarizes the features of the involved datasets.

5.2. Simulation Setup

Our experimental investigation aimed to quantify, in each of the above datasets, the extent to which our similarity measure was capable of identifying related tags, especially when tags were drawn from the long tail. To investigate this, for each dataset of Table 1 has been used as follows. We split it into two different sets, called *test set* and *train set*. Each train set was composed of 90% random bookmarks taken from the involved dataset; we used these bookmarks for training purposes. Test sets contained the remaining 10% of bookmarks which were used for testing. Each bookmark in a test set has then been used as a query; specifically, if the number of tags in such bookmark was large enough, then these were split into two different sets – if possible of the same size – called $tSet_Q$ (*query tag set*) and $tSet_E$ (*expected tag set*). In our experiments, a bookmark was considered large enough if it had at least 3 tags associated. Tags composing $tSet_Q$ were used to query the train set; in particular, we selected from the train set the k tags most similar to tags belonging to $tSet_Q$, according to two metrics: the one we proposed in Section 3, and cosine similarity, which we used as benchmark. We denote this set as $tSet_R$ (*result tag set*). The value of k was chosen equal to the size of the expected set in such a way that $tSet_R$ and $tSet_E$ had the same size. Finally, we compared $tSet_R$ with $tSet_E$: the higher the overlap between $tSet_R$ and $tSet_E$, the more effective the similarity measure in identifying related tags. This follows the intuition that, if a user associated a set of tags to a certain resource, such tags are related to each other (that is, $tSet_E$ contains tags related to those contained in $tSet_Q$).

To quantitatively evaluate our similarity measure, we computed two metrics commonly used in Information Retrieval, namely Precision and Recall [1]:

Propagation	*Bibsonomy*	*CiteULike*	*MovieLens*
$\psi = 0$	0.100638896	0.057922233	0.075126961
$\psi = 0.15$	0.128318833	0.063290603	0.112358995
$\psi = 0.3$	0.139761842	0.070652236	0.115026291
$\psi = 0.6$	0.140748308	0.079320913	0.115534133

Table 2. Precision values in our datasets

Propagation	*Bibsonomy*	*CiteULike*	*MovieLens*
$\psi = 0$	0.100625714	0.057864697	0.075143054
$\psi = 0.15$	0.128373044	0.063273342	0.110927464
$\psi = 0.3$	0.139939546	0.070634975	0.119373901
$\psi = 0.6$	0.140429576	0.079303652	0.119949092

Table 3. Recall values in our datasets

$$Precision = \frac{|tSet_R \cap tSet_E|}{|tSet_R|} \quad (13)$$

$$Recall = \frac{|tSet_R \cap tSet_E|}{|tSet_E|} \quad (14)$$

We computed Precision and Recall values for each test bookmark; we repeated this process 10 times over different train and test splits of the datasets. The results we present next are averages of such runs.

5.3. Results

Tables 2 and 3 shows values of Precision and Recall we obtained by applying our similarity measure on the datasets of Table 1, for different values of ψ (see Equations 3–4). The benchmark is our similarity measure with $\psi = 0$, that is, the case in which our similarity measure reduces into cosine similarity.

From the analysis of Tables 2 and 3 we can draw the following main observation: in large scale folksonomies, classical approaches – such as cosine similarity ($\psi = 0$) – have difficulties finding similarity relationships among the tags belonging to the long tail, as their Precision and Recall is lower than those achieved with our iterative approach for any value of ψ. The considered datasets are characterized by a very long and prominent tail of low popularity tags; in these real cases, out iterative measure of similarity produces Precision/Recall that is approximately 40% better than cosine similarity for BibSonomy and CiteULike, and approximately 50% better for MovieLens.

6. Related Work

In the last few years, folksonomies have been the subject of extensive research. An interesting survey on the characteristics of folksonomies can be found in [4]. One of the first investigations into the characteristics of folksonomies

has been presented by Mathes [18]: in that work, the author discusses advantages (e.g., simplicity of use) and disadvantages (e.g., ambiguity, synonyms) of folksonomies, and investigates the community aspects behind folksonomies, on two scenarios, Flickr[7] and Delicious[8].

Despite their easy-of-use, the lack of structure that characterises folksonomies makes it difficult to browse and find relevant content. To tackle this issue, the research community has been actively researching techniques to support information retrieval. Approaches in this area have followed one of two streams: they have either tried to empirically derive an ontology from the underlying folksonomy, or they have tried to apply graph-exploration techniques on the folksonomy itself.

Lambiotte [14] and Mika [19], for example, were the first to extend the classic bipartite model of tag-resource towards a tripartite model, which takes into account both users (as actors), tags (as concepts) and resources (as instances); they showed that, by applying this model to Delicious, a lightweight ontology could be extracted from the underlying folksonomy. Similarly, [9] used similarity metrics to reconstruct a concept hierarchy.

Hotho et al. [12, 20] followed a different approach instead: they presented a formal model, which converts a folksonomy into an undirected weighted graph, and coupled it with a new search algorithm, namely "FolkRank", based on the well-known seminal "PageRank" [2]. They applied this algorithm to Delicious, and showed how it can be used as a tag recommender system. Other extensions of recommender systems to folksonomy structures have been explored [21, 10]; some of these have been assessed against one of the datasets we adopted in this study, namely BibSonomy [11, 13].

All the above approaches rely on a similarity measure to quantify tag relatedness. Measures which have been often used in the literature include the Jaccard coefficient [8], the cosine similarity [6], and a number of improvements over it [15, 22]. Liu et al. [15] dwelt further into the problem of computing similarities in folksonomies; in particular, they questioned the common assumption that text categorization can be mapped onto orthogonal spaces, due to problems of synonyms and ambiguities (as already figured out by [18]). They then devised an improved similarity metric ("SNOS", Similarity equations in the Non-Orthogonal Space) which is optimized for comparing objects mapped onto non-orthogonal spaces, considering a principle of "mutual reinforcement" from which we drew inspiration in this work. They proved the convergence of this technique and experimentally investigated the performance of SNOS on synthetic datasets, such as the formerly called MSN search engine (now, Bing[9]). Their novel metric was shown to outperform the classic cosine similarity, if applied to the context of finding similar queries. Some of their findings are here extended to the domain of folksonomies.

Similarity measures have often been evaluated on different datasets, making it difficult to assess their relative advantages and disadvantage in different domains. Furthermore, they have often been applied to manipulated datasets, making the comparison even more difficult. Indeed, in order to critically compare them, an evaluation framework has recently been proposed [17], with the aim of providing support to systematically compare several tag similarity measures, using data from Delicious [3]. This work contributes to the assessment of the suitability of similarity measures to scenarios characterized by power-law distribution of tags and non-independence of data, showing how traditional measures like cosine do not work, and proposing an alternative, iterative measure that provides good accuracy instead.

7. Conclusions

In this paper, we have shown that real world folksonomies are characterized by power law distributions of tags and non-independence of data. Under these conditions, traditional similarity measures like cosine similarity fail to capture tags relatedness. To remedy this, we have proposed a novel metric, specifically developed to capture similarity in large-scale folksonomies, that is based on the mutual reinforcement principle: that is, two tags are deemed similar if they have been associated to similar resources, and vice-versa two resources are deemed similar if they have been labelled by similar tags. We have described an efficient realisation of this similarity metric, and assessed its quality experimentally, by comparing it against cosine similarity, on three large-scale datasets, namely Bibsonomy, MovieLens and CiteULike.

Acknowledgement. The research leading to these results has received funding from the European Community's Marie Curie Fellowship Programme (FP7-PEOPLE-2009-IEF) under the Grant Agreement n. 38675. The authors are solely responsible for it and it does not represent the opinion of the Community. The Community is not responsible for any use that might be made of information contained therein.

References

[1] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison Wesley Longman, 1999.

[7] http://www.flickr.com/

[8] http://www.delicious.com/

[9] http://www.bing.com/

[2] S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. *Computer networks and ISDN systems*, 30(1-7):107–117, 1998.

[3] C. Cattuto, D. Benz, A. Hotho, and G. Stumme. Semantic Grounding of Tag Relatedness in Social Bookmarking Systems. In *Proc. of the 7th International Conference on The Semantic Web*, pages 615–631. Springer-Verlag, 2008.

[4] C. Cattuto, C. Schmitz, A. Baldassarri, V.D.P. Servedio, V. Loreto, A. Hotho, M. Grahl, and G. Stumme. Network properties of folksonomies. *AI Communications*, 20(4):245–262, 2007.

[5] P. De Meo, G. Quattrone, and D. Ursino. Exploitation of semantic relationships and hierarchical data structures to support a user in his annotation and browsing activities in folksonomies. *Information Systems*, 34(6):511–535, 2009.

[6] J. Diederich and T. Iofciu. Finding communities of practice from user profiles based on folksonomies. In *Proc. of the 1st International Workshop on Building Technology Enhanced Learning solutions for Communities of Practice*, pages 288–297, 2006.

[7] G.H. Golub and C.F. Van Loan. *Matrix Computations, third edition*. Johns Hopkins University Press, 1996.

[8] Y. Hassan-Montero and V. Herrero-Solana. Improving tag-clouds as visual information retrieval interfaces. In *International Conference on Multidisciplinary Information Sciences and Technologies*, pages 25–28, 2006.

[9] P. Heymann and H. Garcia-Molina. Collaborative creation of communal hierarchical taxonomies in social tagging systems. Technical Report 2006-10, Stanford InfoLab, April 2006.

[10] P. Heymann, D. Ramage, and H. Garcia-Molina. Social tag prediction. In *Proc. of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 531–538. ACM, 2008.

[11] A. Hotho, R. Jäschke, C. Schmitz, and G. Stumme. BibSonomy: A social bookmark and publication sharing system. In *Proc. of the First Conceptual Structures Tool Interoperability Workshop*, pages 87–102, 2006.

[12] A. Hotho, R. Jäschke, C. Schmitz, and G. Stumme. Information retrieval in folksonomies: Search and ranking. *The Semantic Web: Research and Applications*, pages 411–426, 2006.

[13] R. Jäschke, A. Hotho, C. Schmitz, and G. Stumme. Analysis of the publication sharing behaviour in BibSonomy. *Conceptual Structures: Knowledge Architectures for Smart Applications*, pages 283–295, 2007.

[14] R. Lambiotte and M. Ausloos. Collaborative Tagging as a Tripartite Network. *Lecture notes in computer science*, pages 1114–1117, 2006.

[15] N. Liu, B. Zhang, J. Yan, Q. Yang, S. Yan, Z. Chen, F. Bai, and W.Y. Ma. Learning similarity measures in non-orthogonal space. In *Proc. of the thirteenth ACM international conference on Information and knowledge management*, pages 334–341. ACM, 2004.

[16] C.D. Manning, P. Raghavan, and H. Schutze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[17] B. Markines, C. Cattuto, F. Menczer, D. Benz, A. Hotho, and G. Stumme. Evaluating similarity measures for emergent semantics of social tagging. In *Proc. of the 18th international conference on World Wide Web*, pages 641–650. ACM, 2009.

[18] A. Mathes. Folksonomies-cooperative classification and communication through shared metadata. *Computer Mediated Communication*, 2004.

[19] P. Mika. Ontologies are us: A unified model of social networks and semantics. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5(1):5–15, 2007.

[20] C. Schmitz, A. Hotho, R. Jäschke, and G. Stumme. Mining association rules in folksonomies. *Data Science and Classification*, pages 261–270, 2006.

[21] Z. Xu, Y. Fu, J. Mao, and D. Su. Towards the semantic web: Collaborative tag suggestions. In *Proc. of Collaborative Web Tagging Workshop at 15th International World Wide Web Conference*, 2006.

[22] B. Zhang, H.J. Zeng, W.Y. Ma, Z. Chen, N. Liu, and J. Yan. Method and system for determining similarity of items based on similarity objects and their features, May 12 2009. US Patent 7,533,094.

Exploiting semantic aspects to evolve a text-based search on a legacy document management system

Johann Grabner, Andreas Mauczka, Mario Bernhart and Thomas Grechenig
Research Group for Industrial Software
Vienna University of Technology, 1040 Vienna, Austria
{johann.grabner, andreas.mauczka, mario.bernhart, thomas.grechenig}@inso.tuwien.ac.at

Abstract

Semantic technologies provide the means to create a more efficient human-computer interaction. In this paper we showcase how to use semantic technologies in the design of a Document Management System (DMS) to re-engineer a legacy DMS using a text-based search. Due to the historical growth of the database, a text-based search without using domain knowledge delivers bloated result-sets that prevent an efficient use of the DMS. The requirement of an intelligent search can be suitably addressed by using semantic technologies, thus we propose such a new design of a DMS. We employ concepts used in ontologies to provide search results based on hierarchies that are based on concepts of natural language. We construct a metamodel for the application domain to deliver fitting suggestions along with the search results. Finally, we realise a prototype as a proof-of-concept based on the introduced design to highlight the evident advantages of a semantic approach.

1. Introduction

In recent times social interaction and collaboration entered the spotlight of websites, flaunting the title Web 2.0 [10]. After the wide acceptance and success of applications in Web 2.0, the semantic web as an evolutionary follow up of the classical web, which extends the classical web by using the context of information to process information, is rapidly gaining importance[9]. Folksonomies of Web 2.0 are the product of the collaboration of large amounts of individuals with unique perspectives. To be able to gain a foothold in daily life, semantic web products will have to extend Web 2.0 products like Folksonomies by semantic aspects.

A text-based search retrieves information containing the exact or terms similar to the search string. If you search just for a certain text, a text-based search is capable of delivering satisfying results. A text-based search is very likely to fail to find the context behind a search string, however. In the present study whe introduce a search capable of getting both the information and the context of a given text.

Legacy systems tend to evolve into behemoths that might get the required task done, but rather slowly. This is especially tedious when the legacy system is a stand-alone system of an organisation - and every user is forced to work in this system. The legacy system analysed in this work is a document management system, called internally Office Information System, used for filing and searching of documents. This document management system (DMS) has, due to natural growth of the document base, outlived the usefulness of its own text-based search as search results tend to deliver unmanageable amounts of data.

Therefore the capabilities of text-based search are seemingly met and a new approach was suggested, which is presented in this paper. We propose a concept for a semantic search for this DMS and we implement a prototype for the users of the legacy system. We will employ tagging-mechanics and constructs of natural language to support search, file and even navigation through the database. We realise a customised triplestore by using open source technologies. Furthermore we use a Folksonomy with semantic extensions to provide qualitative suggestions for categorisation, navigation and search. Finally, we give a short introduction of the metamodel used to model semantic relations between hypernyms and instances.

2. The Legacy Document Management System

The office (Organisational Unit E010 - "Organisation and Coordination") of the Vienna University of Technology is responsible for all incoming documents by mail, the distribution of these documents to the according organisational units and finally for the archiving of these documents for a period of ten years. The existing workflows are supported by a legacy software system named "OIS" (Office Informa-

tion System). The existing system is in its final software lifecycle stage, since it is not capable of meeting today?s requirements and problems in the processing of the existing amount of data arise. The system is in use since approximately 20 years and more than 200.000 files have been entered into the system, averaging at 10.000 files per year.

Examples of problems with the legacy DMS are:

- Lack of support when filing documents - currently the order of the system is maintained by the experience and established conventions of the users.
- No filing across topics - the legacy system can only attach documents to one topic. Content covering multiple topics can not be realised at the moment.
- Traceability of content - the description of content of a document is currently subject to undocumented conventions. These conventions determine which attributes are used to describe the content (e.g. contract-name, paragraph..).
- Limited number of search results - the maximum number of results is currently limited. Due to the nature of the DMS, not all relevant search results are listed by the system.
- Step-wise refining of search criteria - based on a result set, it is currently not possible to step-wise refine the search criteria to get better results.

The existing problems are faced by proposing a design and a prototype realisation as a proof of concept of an innovative and semantic document management system (DMS). By using a semantic approach we hope to improve the search beyond the capabilities of a classical, text-based search engine. To improve acceptance of the new DMS, current system users were heavily involved during design and implementation of the prototype. We integrate domain knowledge of the OIS users to find possible innovations in the area of DMS as the user group is considered as very experienced DMS users.

Finally a prototype of the DMS is implemented that integrates the semantics of the domain into the current workflows. By taking these semantic relationships into consideration, this prototype is capable of assisting during search and filing of documents by giving content and context related suggestions. To be able to give these suggestions, the system needs to derive the knowledge from the domain and from the current usage of the DMS by the users.

3. Design and Implementation of a Semantic Solution

The scenario described earlier is augmented by a number of factors that have to be taken into consideration for the final solution. These are current user-problems, additional requirements beyond the scope of the legacy DMS gathered in stakeholder-interviews and project management input. The boundaries of the new semantic solution are given by current workflows and processes. The search-function is considered a primary factor for the quality of the DMS in the given scenario as it is of central importance for the workflow in the enterprise. Another important factor are standardised descriptions and how to handle them in a user-friendly manner. The integration of semantic aspects into a folksonomy is required for a working tagging system in the context of this work. As it is one of the major problems of the current system and the system continously grows every year, suggestions for navigation through the data sets, but also for tagging have become a key requirement that needs to be addressed.

In the following chapters we show how we addressed these concerns in our concept and we give a short insight into the technical realisation in form of a prototype.

3.1. Improving the Search Beyond the Scope of a String Comparison

A text-based search is capable of returning exact or similar matches This scope however is too narrow for the requirements of the stakeholders because it does not provide accurate search results in the existing database. A semantic factor is missing [3]. Due to this deficiency, semantic relationships between terms won't be recognised and therefore excluded from the search results. To address these issues the system is able to ask the desired meaning of a term, if the user just entered a homonym. Furthermore the system is capable of mapping alternate naming to the same term. By supporting homonyms, synonyms and abbreviations in the DMS a new quality of search results for the user can be achieved.

3.2. Standardised Descriptions

Natural language with its numerous possibilities of expression and different points of view of similar terms represent obstacles in the standardisation of textual descriptions. This lack of standardisation prevents an automated processing of descriptions. To avoid these pitfalls, we employ the use of tags. This concept is being used in the Web 2.0 [10], [11] with big promise. However, using tags does not guarantee standardised description texts. But the simplification of language lowers the risk of complex descriptions and improves the automated processing of these descriptions.

3.3. Folksonomy with Semantic Extensions

A certain degree of freedom for creating tags in the daily usage of the system is a required trade off to increase user-

acceptance of the application. Similar to [9] and the success of Web 2.0, we believe a collective intelligence of the users is more productive than restrictions on the system. This holds especially true considering the collective need to have a usable tool. The focus of the concept therefore lies in the support of the workflows of the users and to create a more efficient work-experience. By using semantics in the processing of the data, the system is capable of supporting the user by proposing meaningful suggestions that are similar in content (and not just textually). Furthermore suggestions build a fundamental design element for navigating the existing datasets. To provide qualitative suggestions for categorisation, the language concept of hypernyms [4] is used in addition to homonyms and synonyms. By using the natural boundaries of the application domain, it was possible to build a metamodel based on the relationships between the generic terms (similar to the ontologies presented in [6]). We use only one metamodel in the application instance for the prototype.

Connections between hypernyms are modelled using only one kind of semantic relation. This relationship is a hierarchical relationship and shows which hypernym is the subordinate of the other (a meronym). Therefore, the focus is on the existence of a relation between two hypernyms and not on a detailed description of the relation itself. The user has to define the kind of relation of the suggested tags with the entered term himself, the system just shows connected tags. The system relies on the associations of the user to the shown combinations of different tags.

The simplified ontology of the user-domain reflected by the metamodel is the key element for semantic extensions. When performing a search with a combination of tags, the result set not only will contain matching tags, but also combinations of subordinate tags of the search tags (meronyms of the search tags). A tag used as a keyword in a search therefore represents a semantic tag space (all meronyms of the tag are included), containing the existing semantically relevant tags in the system.

3.4. Suggestions for navigation and tagging

Each result set of a user search comes with suggestions for further navigation[2]. These suggestions include recommendations derived from the domain knowledge to further constrain the result set of the follow up search and the tags created together with the tags of the search criteria Figure 1. To create the domain knowledge driven recommendations, meronyms of the entered tags are used. To refine the initial search, the user can add meronyms step by step to semantically tune the result set until it fits his expectations. The system also delivers suggestions for creating a tag or tagging documents (see [5]). During tag creation the user gets a suggested list of meronyms related to the created tag after choosing a hypernym. The metamodel is used to find the tags qualified for a relation Figure 2. For tagging documents, the system suggests combinations of tags sorted by frequency of appearance. Thus the user ins informed which sets of tags were used for prior tagging of documents. By showing possible matchings, the system guides the user to place his documents in the proper (best-fitting) filing area. The system supports the user by using the domain knowledge to provide suggestions in every aspect of the application.

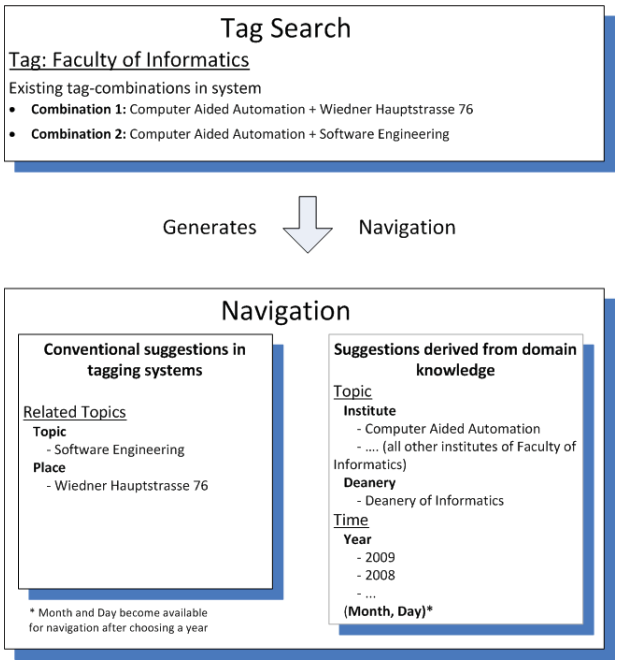


Figure 1. Example of suggestions for navigation

The structure of the hypernyms is built using three semantic axes (based on concepts presented in[12]). These axes are time, place and content. The suggested tags for navigation are grouped according to hypernyms on the semantic axes Figure 1. A strategy for visualisation of the search results is selected depending on the numbers of tags related to the search criterion. If there are less than six entries, the tags are listed consecutively. For more than six and less than 100 tags a combo-box is used. If more than hundred related tags exist, a text field with auto-completion is used together with a list of the top five tags used together with the search criterion.

3.5. Technical Implementation

The concept presented in the earlier sections is implemented in a prototype. The database technology was con-

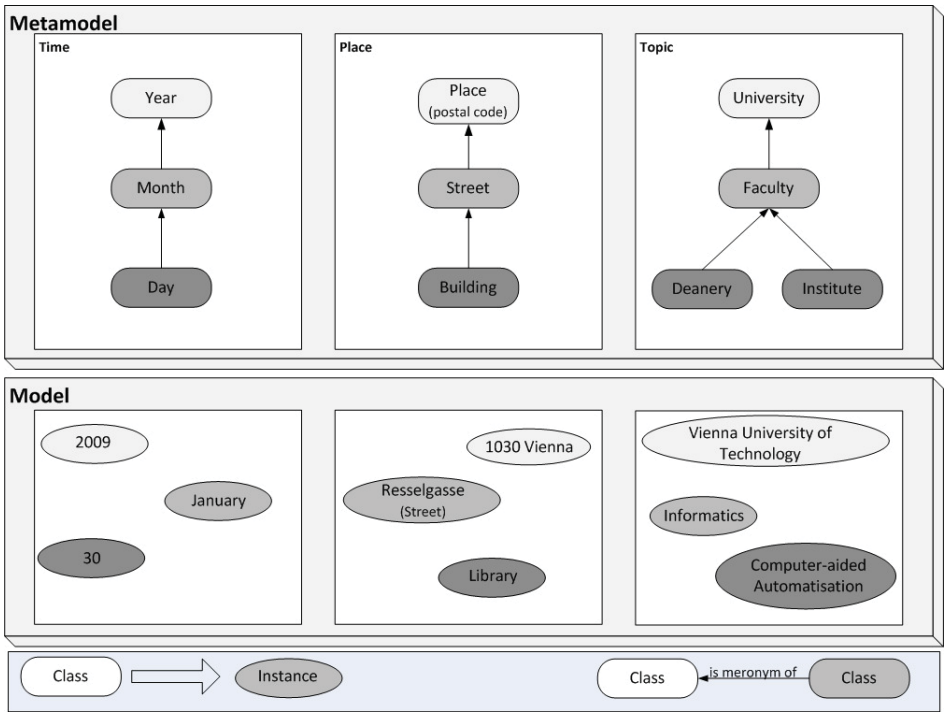


Figure 2. Overview of meronyms, classes and instances structure

strained due to a given software environment of the legacy application. Therefore a relational database system had to be used to create a customised triplestore. Due to the constraints earlier using a stand-alone triplestore or an existing triplestore based ona relational database management system (RDBMS) had to be discarded. The underlying RDBMS of the legacy application is MySQL[1], thus MySQL is the RDBMS on which the customised triplestore is build on. The triplestore is realised by using two tables in the database. The first table holds unique identifiers of the resources (realised as URIs) including the used literals. The second table holds ternary relationships between the resources of the first table and is representing therefore the triplestore. This option for realising the triplestore was not chosen deliberately but to conform to organisational constraints on the prototype.

As there were no such constraints on the choice of the web application framework, we chose "Ruby on Rails"[2]. Development started out swiftly due to numerous Code-Generators and Plugins available. RESTful Routes [3] are used to give access to the existing resources in the system. Every tag has a URL, which is used as unique identifier for the resources in the database. Auto-completion for entering tags is implemented using functionality provided by the "Yahoo! User Interface Library"[4].

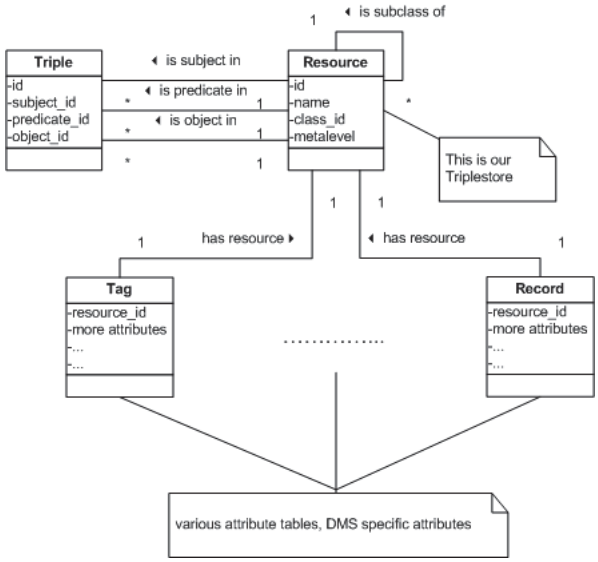


Figure 3. Classdiagram of the triplestore

We use the Metadata Terms of the Dublin Core Metadata Initiative[5] (DCMI) to describe the resources in the triplestore. Every resource of a type has at least a title.

[1] http://www.mysql.com/
[2] http://rubyonrails.org/
[3] http://guides.rubyonrails.org/routing.html#restful-routes
[4] http://developer.yahoo.com/yui/
[5] http://dublincore.org/

The type of the resource is specified by its given hypernym. Synonyms and denominators or abbreviations are optional for the descriptions. The predicates "dcterms:title", "dcterms:identifier" and "dcterms:alternative" are included from the standard to put tags and literals into the relations according to the DCMI Standard[5]. We use "dcterms:hasPart" to express meronym-like relationships between two tags. For reasons of performance additional attributes of resources are stored in conventional database tables. These attribute-tables hold the application-specific, DMS relevant data about the corresponding resources in the triplestore. To improve data queries, statistics about tags and documents are recorded in caching-tables. These tables are constantly updated. Furthermore, to improve performance consecutive queries on the same table are congregated into a single query.

4. Related Work

Folksonomies and the extensions of Folksonomy by using semantic aspects is currently a hot topic due to high activity in this area in the semantic research. Marchetti et al. identify in [8] weaknesses of existing, collaborative tagging-systems. Based on services of Wordnet [6] and Wikipedia [7] a semantic, collaborative tagging-system is implemented. Entries in Wikipedia and Wordnet are used to allow the user to learn more on a concept behind a tag. To connect a resource and a concept the user is able to choose from different relations. The main difference between the system in [8] and our system is that the system in [8] has external dependencies (e.g. Wordnet, Wikipedia, ..). The specification of a semantic statement is the argument for the semantic search, while our system accepts tags as search input. The main focus in [8] is semantic tagging - no semantic navigation for result sets is implemented.

Hope et al. [4] present a semantic tagging-system for blogs. They use hypernyms for their semantic. Compared to our work, Hope et al. chose a simplified approach as they do not rely on meronyms nor do they have a navigation. Similar to [8] they created a prototype using Wordnet to create an ontology of tags.

An interesting approach to generate semantic tag-hierarchies for navigation in an existing Folksonomy is presented in Laniado et al. [7] . In [11], Specia et al. build a semantic tag-hierarchie by using co-occuring tags in folksonomies. Both Specia et al. and Laniado et al. deliver the foundations for approaches to extend Folksonomies with semantic aspects. Our work relied on their research during migration of the data from the legacy system.

[5] http://dublincore.org/

[6] http://wordnet.princeton.edu/

[7] http://www.wikipedia.org/

5. Conclusion

The proposed design of a semantic search based on a semantic tagging-system is not only restricted to the use in the DMS area. Any resources can actually be described using the tagging system. By supporting aspects of natural language, the way in which users and machines interact with each other can be improved. Applications that incorporate domain knowledge and aspects of natural language are capable of providing a level of service superior to conventional text-based searches. Concrete suggestions to refine the search result set can be directly derived from domain knowledge. The system is capable of asking the user to correctly specify ambiguous entries and actively support the user with semantic suggestions. We only used hypernyms and meronyms out of the number of possible semantic relations to provide the users with tangible, in natural language common relations. Simplicity is a key factor for acceptance of a system, especially in the domain of document management as users working with a DMS tend to be no computer experts. By passing into the semantical web, applications are required to show the user semantic aspects in the processing of information in a simplified fashion [1].

The prototype of the DMS has an improved search functionality over the legacy system and was widely accepted by the current DMS users. Descriptions and categorisations into filing areas were migrated from the legacy system as tags into the prototype to allow a smooth integration for the users.

5.1. Future Work

There are several possibilities to extend the presented method for realising a semantic tagging-system. The realisation of distributed queries across several application instances in different domains might be a possible extension. Using ontologies to transfer domain knowledge seems an obvious, possible next step. The modelled domain knowledge may not only be used for semantic suggestions but also for answering questions a user might pose. New co-workers might be integrated easier by finding proper visualisation techniques of the domain knowledge. A problem that has not been addressed yet is the tracking of changes in domain knowledge. Filing structures in the DMS, terms and semantics all change over time. There is a strong need for mechanisms to reconstruct changes in the domain knowledge.

References

[1] H. Alani, Y. Kalfoglou, K. Hara, and N. Shadbolt. Towards a killer app for the semantic web. *4th International Semantic Web Conference, ISWC 2005, Gal-*

way, Ireland, November 6-10, 2005. Proceedings, Jan 2005.

[2] J. Gemmell, A. Shepitsen, B. Mobasher, and R. Burke. Personalizing navigation in folksonomies using hierarchical tag clustering. *Proceedings of the 10th International Conference on Data Warehousing and Knowledge Discovery*, Jan 2008.

[3] R. Guha, R. McCool, and E. Miller. Semantic search. *Proceedings of the 12th international conference on World Wide Web*, Jan 2003.

[4] G. Hope, T. Wang, and S. Barkataki. Convergence of web 2.0 and semantic web: A semantic tagging and searching system for creating and searching blogs. *Semantic Computing, 2007. ICSC 2007. International Conference on*, pages 201 – 208, Sep 2007.

[5] R. Jaschke, L. Marinho, A. Hotho, and L. Schmidt-Thieme. Tag recommendations in folksonomies. *11th European Conference on Principles and Practice of Knowledge Discovery in Databases, Warsaw, Poland, September 17-21, 2007. Proceedings*, Jan 2007.

[6] H. Kim, S. Scerri, J. Breslin, and S. Decker. The state of the art in tag ontologies: a semantic model for tagging and folksonomies. *Proceedings of the 2008 International Conference on Dublin Core and Metadata Applications*, Jan 2008.

[7] D. Laniado, D. Eynard, and M. Colombetti. A semantic tool to support navigation in a folksonomy. *HT '07: Proceedings of the eighteenth conference on Hypertext and hypermedia*, Sep 2007.

[8] A. Marchetti, M. Tesconi, and F. Ronzano. Semkey: A semantic collaborative tagging system. *WWW07 Workshop, Tagging and Metadata for Social Information Organization*, 2007.

[9] E. Motta and M. Sabou. Next generation semantic web applications. *First Asian Semantic Web Conference, Beijing, China, September 3-7, 2006. Proceedings*, Jan 2006.

[10] S. Murugesan. Understanding web 2.0. *IT Professional*, 9(4):34 – 41, Jul 2007.

[11] L. Specia and E. Motta. Integrating folksonomies with the semantic web. *4th European Semantic Web Conference, ESWC 2007, Innsbruck, Austria, June 3-7, 2007. Proceedings*, Jan 2007.

[12] M. Tvarozek and M. Bieliková. Personalized faceted navigation in the semantic web. *7th International Conference, ICWE 2007 Como, Italy, July 16-20, 2007 Proceedings*, Jan 2007.

Extracting Ontology Hierarchies From Text

Jone Correia, Rosario Girardi, Carla Faria
Federal University of Maranhão
São Luis, Brazil
jonecorreia@ufma.br, rgirardi@deinf.ufma.br, carlafaria@ifma.edu.br

***Abstract* — Ontologies are an approach for knowledge representation capable of expressing a set of entities and their relationships, constraints, axioms and vocabulary of a given domain. Manual construction of ontologies by domain experts and knowledge engineers is an expensive and time consuming task so, automatic and/or semi-automatic approaches are needed. Ontology Learning looks for automatically or semi-automatically identifying ontology elements like classes, taxonomic and non-taxonomic relationships, properties and axioms from textual resources. This article proposes a process for the automatic extraction of ontology taxonomic relationships from English texts using natural language processing techniques. Some experiments using a legal corpus were conducted in order to evaluate it. Initial results are promising.**

Keywords - Ontology, Ontology Learning, Natural Language Processing, Taxonomic Relationships, Ontology Hierarchies

I. INTRODUCTION

Ontologies are used by modern knowledge-based systems allowing the representation and sharing of knowledge about an application domain [9]. They provide a formal means of knowledge representation capable of expressing a set of entities, their relationships, constraints and rules (conditional statements) of a given domain [10][15].

Ontology Learning looks for identifying ontology elements like classes, taxonomic and non-taxonomic relationships, properties and axioms from textual resources. Manual construction of ontologies by domain experts and knowledge engineers is an expensive and time consuming task so, automatic and/or semi-automatic approaches are needed.

This paper proposes a process for automatic learning of ontologies from text based on Natural Language Processing (NLP) [1][4] techniques. Particularly, the process looks for identifying taxonomic relationships. An experiment conducted to evaluate this the process using a legal corpus is also described.

The article is organized as follows. Section II introduces the ontology definition used in this work. Section III presents an overview of the proposed process. Section IV describes an experiment conducted to evaluate it. Section V summarizes related work and, finally, section VI concludes the article discussing results and future work.

This work is supported by the Brazilian Government agencies: CNPq, CAPES and FAPEMA

II. AN ONTOLOGY DEFINITION

Ontologies are formal specifications of concepts in a domain of interest. Their classes, relationships, constraints and axioms define a common vocabulary to share knowledge [10].

Formally, an ontology can be defined as the tuple:

$$O = (C, H, I, R, P, A) \quad (1)$$

where,

$C = C_C \cup C_I$ is the set of entities of the ontology. They are designated by one or more terms in natural language. The set C_C consists of classes, i.e., concepts that represent entities that describe a set of objects (for example, "Mother" $\in C_C$) while the set C_I is constituted by instances, (for example "Anne Smith" $\in C_I$).

$H = \{kind\_of\ (c_1,c_2) \mid c_1 \in C_C,\ c_2 \in C_C\}$ is the set of taxonomic relationships between concepts, which define a concept hierarchy and are denoted by "$kind\_of(c_1,c_2)$", meaning that c_1 is a subclass of c_2, for instance, "kind_of(Mother, Person)".

$I = \{is\_a\ (c_1,c_2) \mid c_1 \in C_I \wedge c_2 \in C_C\} \cup \{prop_K\ (c_i,value) \mid c_i \in C_I\} \cup \{rel_K\ (c_1, c_2, ...cn) \mid \forall I, c_i \in C_I\}$ is the set of relationships between ontology elements and its instances. For example, "is_a ("Anne Smith", Mother)", "date_of_birth("Anne Smith", 02/12/1980)" and "mother_of("Anne Smith", " Katie Smith")" are relationships between classes, relationships, properties with its instances.

$R = \{rel_k\ (c_1,c_2,..., c_n) \mid \forall i, c_i \in C_C\}$ is the set of ontology relationships that are neither "kind_of" nor "is_a". For example, "mother_of(Mother, Daugther)".

$P = \{prop_K\ (c_i,datatype) \mid c_i \in C_C\}$ is the set of properties of ontology entities and its basic datatype. For instance, "date_of_birth (Mother, mm/dd/yyyy)".

$A = \{condition_x \Rightarrow conclusion_y\ (c_1,c_2,..., c_n) \mid \forall j, c_j \in C_C\}$ is a set of axioms, rules that allow checking the consistency of an ontology and infer new knowledge through some inference mechanism. The term $condition_x$ is given by $condition_x = \{(cond_1,cond_2,\ldots,cond_n) \mid \forall z, cond_z \in H \cup I \cup R\}$. For example, " ∀ Mother, Daughter1, Daughter2, mother_of(Mother, Daughter1), mother_of(Mother, Daughter2) ⇒ sister_of (Daughter1, Daughter2)" is a rule that

indicates that if two daughters have the same mother then, the daughters are sisters.

III. A Process For Acquiring Taxonomic Relationships

Figure 2 illustrates the proposed process for the automatic acquisition of ontology taxonomic relations, that is, the H set of the ontology definition in section II. It consists of four steps: "Tagging", "Extraction of Candidate Classes", "Identification of Hyponyms and Synonyms" and "Identification and Representation of Taxonomic Relationships".

The input of the process is a corpus consisting of a set of textual documents in a particular application domain. The step "Tagging" is intended to identify the tokens, sentences, grammatical classes and lemmas with the application of NLP techniques. Then, the "Extraction of Candidate Classes" phase is responsible for separating the tokens which are likely to be classes of the ontology hierarchy. The step "Identification of Hyponyms and Synonyms" identifies the synonyms and hyponyms of the candidate classes obtained in the previous step. The last stage of the process is the "Identification and Representation of Taxonomic Relations" phase which aims at identifying the final taxonomic relations using heuristic patterns and their representation in an ontology specification language. In the next sub-sections the transformations made at each stage of the process are presented in detail.

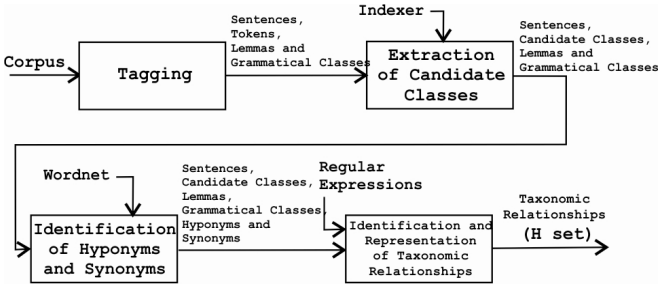


Figure 2. A process for the acquisition of taxonomic relations of an ontology

A. *Tagging*

The step "Tagging" has a corpus as input and aims at transforming them into a model that can be processed computationally. This step consists of the following activities: "Tokenization", "Division in Sentences", "Lemmatization" and "Lexical Analysis" (Figure 3).

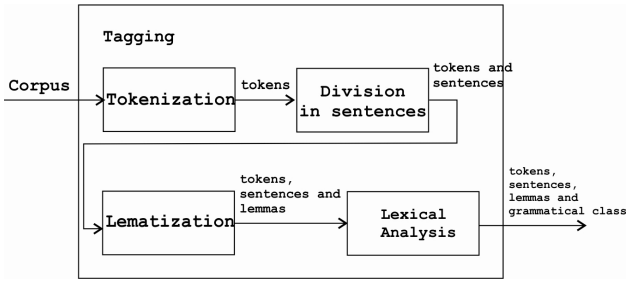


Figure 3. Activities of the "Tagging" phase

The "Tokenization" activity identifies the terms (tokens) in the document. The activity "Division in Sentences" organizes the identified tokens by grouping them into sentences. The activity "Lemmatization" performs the reduction of each token to its basic form . This form generalizes the inflected forms of a token making possible the grouping of tokens. The activity "Lexical Analysis" aims at identifying the grammatical classes for each token selected in the tokenization activity.

The result of this step is the corpus tagged with tokens, sentences, lemmas and grammatical categories. For example, Figure 4 shows the Tagger phase applied to the sentence "Mens have mothers. Mothers are great people!". For example, Figure 4 shows the processing results of applying the "Tagging" phase to the text fragment "*Mens have mothers. Mothers are great people!*".

The first frame shows the terms that were marked as tokens: "Men", "Have", "Mothers", "Mothers", "Are", "Great" and "People". Dots and spaces are also tokens, however, for illustrative purposes, only the terms were shown. The second frame shows the result of dividing the text into sentences: "Men Have mothers" and "Mothers are great people". Next "Lemmatization" is performed on each term. For example, the term "mothers" has "mother" as lemma. The last frame corresponds to Lexical Analysis and shows each token and its corresponding part of speech. For example, the term "mothers" is tagged NNS [14] indicating that this is a plural noun.

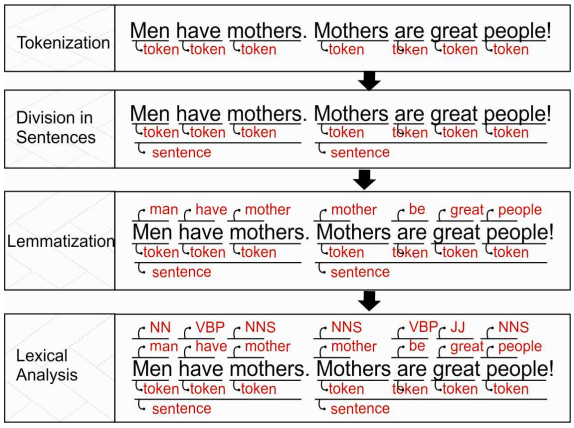


Figure 4. Example of a "tagged" text

B. *Extraction of Candidate Classes*

This phase aims at selecting candidate classes among concrete and abstract nouns.

The hypothesis assumed in this work is that only nouns may be considered classes. Thus, conceptually speaking, we can find two kinds of nouns: concrete nouns and abstract nouns. Concrete nouns are used to represent people names (anthroponyms) or place names (toponyms), which essentially characterizes class instances. Therefore, in the "Extraction of Candidate Classes" anthroponyms and toponyms are ignored and only concrete and abstract nouns are selected.

The products of this step are candidate classes, sentences, lemmas and grammatical classes.

C. *Identification of Hyponyms and Synonyms*

The "Identification of Hyponyms and Synonyms" looks for the identification of synonyms and hyponyms in WordNet [21], a lexical database that contains natural language terms, their definitions and their semantic relationships as synonymy, hyponymy and hypernym. The classes selected in the previous step are located in Wordnet and whenever a synonym or hyponym is found this occurrence in the text is stored. That is, each class is associated with an array of synonyms and hyponyms from the text. The product of this step is the same product from the previous stage plus the relations of synonymy and hyponymy, of each select term, found in Wordnet.

D. *Identification and Representation of Taxonomic Relationships*

The "Identification and Representation of Taxonomic Relationship" aims at discovering taxonomic relationships through the application of heuristic patterns and at representing them in an ontology specification language. Each sentence is analyzed in order to match the patterns. To test the existence of hyponymy relations we apply regular expressions on sentencing patterns with heuristic examples as the Hearst patterns [11]. Regular expressions are a formal method to specify a text pattern [12]. For example, applying the regular expression "([A-Za-z]+,.)+[A-Za-z] + and other [A-Za-z]+" to "*My daughter loves dolls, masks and other toys*", the matched pattern should be "*My daughter loves dolls, masks and other toys*". The tokens "Dolls" and "Masks" are related to "Toys" in a relationship of hyponymy, thus being "Toy" a superior class of "Doll" and "Mask" in the hierarchy of an ontology. This matching represents the pattern (iv) in the table of Hearst heuristics patterns of Figure 5, where NP0 represents a token hierarchically superior to the other NP.

(i) NP0 such as NP1 {, NP2 ... , (and | or) NPi}
(ii) such NP0 as {NP ,}* {(and | or)} NP
(iii) NP {, NP}* {,} or other NP0
(iv) NP {, NP}* {,} and other NP0
(v) NP0 {,} including { NP ,}* {or | and} NP
(vi) NP0 {,} especially { NP ,}* {or | and} NP

Figure 5. Heuristic Patterns of Hearst (Hearst, 1992)

The products of this phase are the taxonomic relationships, the set H of the definition of ontology in Section II, represented in an ontology specification language.

IV. EVALUATION

A case study in the area of Family Law allows for a preliminary assessment of the effectiveness of the proposed process. For that, a prototype tool called T-NLPDumper has been developed for automating the process using Java [13].

A corpus from the Family Law Doctrine [7] has been used in the case study. One hundred sentences randomly selected from the corpus has been manually analyzed by a domain expert, who did the manual identification of taxonomic relationships. These results (Figure 6) were manually compared with the results found on the automatic extraction of taxonomic relationships with the T-NLPDumper tool .

The results were also compared with an adaptation of the precision measure from the Information Retrieval area [5], considering the number of correctly extracted taxonomic relations.

Precision is the ratio between the number of taxonomic relationships extracted correctly (NREC) and number of taxonomic relationships extracted (NRE).

$$P = NREC / NRE \quad (2)$$

The precision value obtained in this experience was 81,57%.

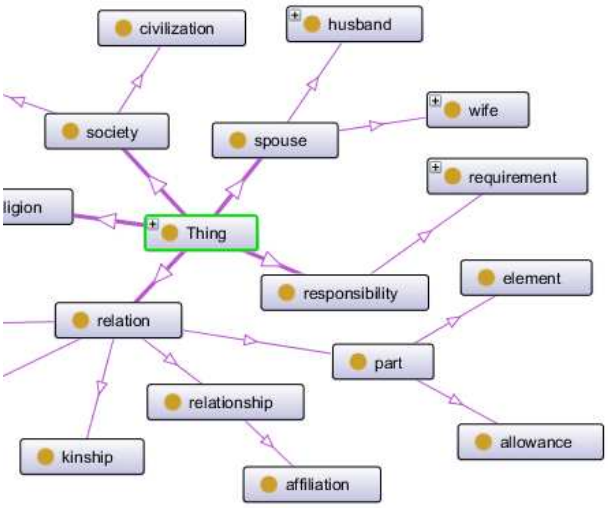


Figure 6. Part of the ontology hierarchy obtained as a result of the experience.

V. RELATED WORK

Table 1 shows the main approaches for automatic and semi-automatic identification of taxonomic relationships.

The main techniques used (second column of Table 1) are based on lexical syntactic patterns [11][19], statistics with Markov Logic Networks (MLR) [6] and Machine Learning techniques (ML) [3][2][16]. More recent works have in common the application of techniques of Natural Language Processing (NLP) in association with other techniques.

The approaches described in [6][16][19] and the one proposed in this article use WordNet as a lexical base for the extraction of hyponyms and GATE [8] to perform the Natural Language Processing tasks. Weka [20] was used as a framework for the clustering tasks in [16], while the tools TreeTagger [18] and Lopar [17] are used in [3] to find grammatical categories and as a parsing tool, respectively.

The effectiveness is shown in terms of the precision obtained for each approach in their experiments. However, these numbers cannot be considered for comparative purposes, because the evaluation of each approach was performed with different corpora and ontologies.

TABLE I. COMPARATIVE TABLE OF THE MAIN TECHNIQUES FOR LEARNING TAXONOMIC RELATIONSHIPS

Approach / Year	Main Technologies	Main Tools	Effectiveness (Precision) / Domain	Level of Automation
Cimiano et al., 2004 [3]	CFA NLP	TreeTagger Lopar	29,33% / Tourism 33,11% / Finance	Automatic
Drumond, 2009[6]	NLP MLN Dictionary PREHE	GATE Wordnet	17% / Tourism	Semi-Automatic
Caraballo, 1999 [2]	ML-Clustering	-	39% / Wall Street Journal	Automatic
Sarmento, 2009 [16]	ML - Clustering NLP Dictionary	WEKA GATE Wordnet	n/a / Tourism	Semi-Automatic
Hearst, 1992 [11]	Lexical Syntactic Patterns	-	63%	Automatic
Ting Wang et al. 2006 [19]	Lexical Syntactic Patterns SVM NLP Dictionary	Wordnet GATE	73,87% /-	Automatic
Correia et al, 2011	NLP Heuristic Patterns Dictionary	GATE Wordnet	81,57% / Family Law	Automatic

The approaches in [2], [3], [11] and the one proposed in this work provide solutions for a fully automatic acquisition of taxonomic relations, while the approaches in [16] and [6] propose just semi-automatic solutions.

Main advantages of the technique proposed here are its domain independency and high precision value (more than 80%).

VI. CONCLUSION

The automatic process for ontology learning proposed in this paper is based on natural language processing and consists of four steps: "Tagging", "Extraction of Candidate Tokens", "Identification of Hyponyms and Synonyms" and "Identification and Representation of Taxonomic Relations".

The process has been evaluated through a case study conducted in the domain of Family Law, showing good results, and demonstrating that the use of techniques of natural language processing represents a promising approach for learning taxonomic relationships of ontologies.

Currently, new heuristic patterns are being developed in order to improve the identification of taxonomic relationships. The development of a tool for implementing all the proposed process is also under construction.

Finally, to improve the results of the proposed tool it is necessary to compare them with concept extractions done manually by domain experts in other domains.

REFERENCES

[1] Allen J.: Natural Language Understanding. Redwood City, CA: The Benjamin/Cummings Publishing Company, Inc (1995).

[2] Caraballo, S.A. Automatic construction of a hypernym-labeled noun hierarchy from text. Proceedings of the 37th annual meeting of the Association for Computational Linguistics on Computational Linguistics, Association for Computational Linguistics Morristown, NJ, USA, p. 120–126, 1999.

[3] Cimiano, P.; Hotho, A.; Staab, S. Learning Concept Hierarchies from Text Corpora using Formal Concept Analysis. 2004.

[4] Dale R., Moisl H., Somers H. L.: Handbook of natural language processing. CRC (2000).

[5] Dellschaft K., Staab S. (2006) "On how to perform a gold standard based evaluation of ontology learning", In: Proceedings of the 5th International Semantic Web Conference, p. 228 – 241, Athens. Springer.

[6] Drumond, L. Automatic Acquisition of Concepts Hierarchies Using Statistical Relational Learning. Master's Dissertation, Federal University of Maranhão, 2009. (In Portuguese)

[7] Family Law. Available at <http://en.wikipedia.org/wiki/Family_law/> Accessed 13/June/2010.

[8] GATE. General Architecture for Text Engineering. Available at <http://gate.ac.uk/> Accessed 14/June/2010.

[9] Girardi, R. Guiding Ontology Learning and Population by Knowledge System Goals. International Conference on Knowledge Engineering and Ontology Development. Valencia. 2010

[10] Guarino N., Masolo C., Vetere C. (1999) "Ontoseek: Content-based Access to the web", IEEE Intelligent Systems, v. 14(3), p. 70-80.

[11] Hearst, M. Automatic acquisition of hyponyms from large text corpora. In: International Conference on Computational Linguistics, 14., 1992, Nantes. Proceedings... . Morristown: Association For Computational Linguistics, 1992. p. 539 - 545.

[12] Jargas, Marinho. Regular Expressions. Publisher Novatec. 2ed. 2008. (In Portuguese).

[13] Java. Oracle. Available at: <http://www.oracle.com/technetwork/java/index.htm>. Accessed: 15/September/2010.

[14] Marcus, M.; Santorini, B.; Marcinkiewicz, M. Building a Large Annotated Corpus of English: The Penn Treebank. Computational Linguistics: Special Issue on Using Large Corpora, [S. l.], v. 19, n. 2, p. 313-330, 1993.

[15] Nierenburg S., Raskin V.: Ontological Semantics, MIT Press (2004).

[16] Sarmento, G. Application of Hierarchical Cluster Techniques for Extracting Taxonomic Relations of Ontology. CGCC-UFMA Final Degree work (2009). (In Portuguese)

[17] Schmid, H. Lopar: Design and implementation. In Arbeitspapiere des Sonderforschungs bereiches 340, No. 149. 2000.

[18] Schmid, H. Probabilistic part-of-speech tagging using decision trees. In Proceedings of the International Conference on New Methods in Language Processing. 1994.

[19] Ting Wang et al. Automatic Extraction of Hierarchical Relations from Text. 2006

[20] Weka project. Available at: <http://www.cs.waikato.ac.nz/ml/weka/>. Accessed: 13/June/2010..

[21] Wordnet. WordNet: A lexical database for English. Cognitive Science Laboratory, Princeton University, Available at: <http://wordnet.princeton.edu/>. Accessed: 16/September/2010.

From Glossaries to Ontologies: Disaster Management Domain

Katarina Grolinger, Kevin P. Brown, Miriam A.M. Capretz
Department of Electrical and Computer Engineering, Faculty of Engineering
The University of Western Ontario
London, ON, Canada N6A 5B9
{kgroling, kbrow43, mcapretz}@uwo.ca

Abstract - **Our society's reliance on a variety of critical infrastructures (CI) presents significant challenges for disaster preparedness, response and recovery. Experts from different domains including police, paramedics, firefighters and various other CI teams are involved in the fast paced response to a disaster, increasing the risk of miscommunication. To ensure clear communication, as well as to facilitate CI software interoperability, a common disaster ontology is needed.**
We propose using the knowledge stored in domain glossaries, vocabularies and dictionaries for the creation of a lightweight disaster management domain ontology. Glossaries, vocabularies and dictionaries are semi structured representations of domain knowledge, where significant human effort has been invested in choosing relevant terms, determining their definitions, acronyms, synonyms and sometimes even relations. We use that knowledge built into semi formatted documents for ontology learning. In particular, we look at five glossaries/vocabularies from the disaster management domain and analyze their content similarity and structure. A lightweight disaster ontology is created exploiting the structure of the semi-structured source documents.

Keywords- Ontology, Glossary, Disaster Management, Ontology Learning

I. 1. INTRODUCTION

Today we rely heavily on a variety of critical infrastructures (CI) such as electrical systems, water supplies, telecommunications, transportation, emergency services and others. Each of these systems is highly complex with their daily operation, maintenance and repairs requiring specialized domain knowledge. On the other hand, there exists significant interdependency between these systems: water distribution systems rely on electricity to power water pumps and emergency response teams rely on a variety of telecommunication methods. The significance of the cascading effects caused by interdependencies of CI has been recognized and significant efforts have been made in attempts to understand and manage them better [1] [2].

The variety of infrastructure systems involved, together with their interdependencies, demand an interdisciplinary approach to disaster management. Involvement of experts from different domains, and the need for exchange of information across domains (between people, as well as between machines), represent significant challenges in achieving successful communication. A word in common to two or more domains may have different meanings; or, conversely, different terms may represent the same concept. In fast paced emergency response situations, this may cause misunderstandings and possibly result in severe consequences.

The importance of having a common understanding within the disaster management field has been recognized; consequently, different glossaries, vocabularies and dictionaries have been created by multiple agencies involved in the disaster prevention, response and recovery processes [3] [4] [5] [6] [7]. Depending on the main focus of the agency, these glossaries, vocabularies and dictionaries vary with regards to the terms they include as well as term definitions. Furthermore, because they are mainly created to be used by people, they are in text form (PDF or text files) that cannot be easily read and understood by computers. As such, these glossaries, vocabularies and dictionaries cannot be used in the information systems that are becoming an integral part of any disaster response process. The way of making this knowledge available for use by computers is through ontologies.

An ontology is a formal, explicit specification of a shared conceptualization [8] that provides a common understanding of information. Additionally, ontologies provide a way of representing human knowledge, making it readable and understandable for machines. This, in turn, represents the basis for achieving semantic interoperability.

In this paper we look at five glossaries and vocabularies each from a different Canadian or American disaster management agency, analyze their differences and propose the creation of an ontology from these glossaries, vocabularies and dictionaries. In the remainder of the paper, the term glossary will be used to represent glossaries, vocabularies and dictionaries.

This paper is organized as follows: Section II reviews related work including ontologies in disaster management and ontology learning. Emergency management glossaries are presented in Section III together with comparative analyses. Section IV describes ontology creation from glossaries and the conclusions and future work are presented in Section V.

II. Related Work

The main directions of research efforts related to CI interdependencies are: research dealing with analyzing past incidents that involved CI interdependencies [1] [2], studying infrastructure interdependencies through the use of simulators [9] [10] [11] and ontologies in disaster management [12] [13]. Particularly relevant to our work is the work on ontologies.

Peng et al. [12] propose the Emergency Case Ontology Model (ECOM) as a way of organizing the emergency case knowledge by taking into consideration relations among emergency cases. The proposed model handles heterogeneity among different earthquake disasters, but it is earthquake specific and does not consider other emergencies.

Castorini et al. [13] propose the Knowledge Base System (KBS) founded on ontologies with the main goal of modeling CI and their interdependencies. The proposed framework consists of: MKIONT (Meta Knowledge Infrastructure ONTology) which defines a template for conceptualization; IONT (Infrastructure ONTology) which represents knowledge of a specific CI domain (e.g. water distribution or telecommunication); FONT (Federation ONTology) which describes interactions between infrastructures and the Gateway which provides a connection between the KBS and simulators.

Another field relevant to this work is ontology learning, which is the process of building an ontology from scratch by enriching, or adapting an existing ontology in a semi-automatic fashion using distributed and heterogeneous knowledge and information sources. The majority of research efforts in ontology learning focuses on learning from unstructured sources [14] [15] [16], primarily text documents, as a significant portion of today's knowledge is stored in such form. Gómez-Pérez and Manzano-Macho compared different ontology leaning methods from unstructured text [17], and presented advantages and disadvantages of each method. Other possible sources of information for ontology learning are structured sources such as databases and semi-structured documents including XML schemas, web pages, glossaries, dictionaries and vocabularies. Since semi-structured sources have various structure elements, the structure can be exploited to extract concepts and/or relations

Zhao and Li [18] propose ontology learning from the hierarchy structure of organization websites. The approach is motivated by the observation that the organization web site is organized in a hierarchical sitemap that reflects a shared view of the organization structure. Consequently, they use the sitemap hierarchy to create the lightweight organizational ontology.

Karoui et al. [19] combine exploiting the structure of the HTML documents with natural language processing techniques for ontology learning. They propose the Contextual Concept Discovery (CCD) algorithm based on K-means clustering and guided by a structural context. In HTML documents they observe physical links, such as heading-paragraph links that represent the structure of documents and logical links that represent links between tags, such as keyword tags. When terms appear in the same context, within the same block tag or within linked elements, it indicates their co-relation. This structural context drives the incremental use of K-means algorithm in identification of ontology concepts.

Davulcu et al. [20] use taxonomy-directed web sites to bootstrap the ontology population task of extracting instances of concepts and their classification into ontology concepts. OntoMiner detects HTML markup and turns it into hierarchical structures that are in turn used for ontology population. Shinzato and Torisawa [21] use itemization and listing in HTML documents to extract hyponym relations.

Navigli and Velardi [22] enrich the CIDOC CRM cultural heritage core ontology using the Art and Architecture Thesaurus (AAT). Descriptions of the meanings of the terms from AAT are processed using NLP (Natural Language Processing) techniques, annotated with CIDOC properties and formalized. The core ontology is enriched from formalized term definitions.

García-Silva et al. [23] propose a methodology for creating ontologies by reusing and re-engineering non-ontological resources. Their proposed patterns for re-engineering define a process to transform non-ontological resources into ontologies. While [23] is concerned with the transformation process, we explore to which extent the structural elements of glossaries can be used in ontology creation.

III. Emergency Management Glossaries and Vocabularies

Glossaries and vocabularies play a significant role in emergency management due to the importance of clear communication during disaster response. Misunderstandings could lead to severe consequences, even the loss of life. Therefore government and private agencies involved with disaster management often create and publish dictionaries of relevant emergency terms. Depending on the focus of each agency, the included terms and their definitions may be significantly different.

We have analyzed five glossaries/vocabularies readily available from the web. Two are from Canadian sources: the Emergency and Crisis Communication Vocabulary from Government Services Canada [3] and the EMO (Emergency Management Ontario) glossary [4] from the Ontario provincial government. The remaining three sources are American: NIMS (National Incident Management System) glossary [5], ICDRM (The Institute for Crisis, Disaster and Risk Management) glossary [6] and ICS (Incident Command System) glossary [7]. The five listed glossaries were chosen since they are relatively generic nonspecific management glossaries dealing with generic disasters and are not disaster (i.e. flood or earthquake) or responder type (i.e. firefighters or CI teams) specific. The ICS glossary is somewhat specific since it deals with command and control in particular, but its main goal is coordination among the different actors in emergency situations which is not highly dependent on disaster or responder type. The two Canadian sources contain English and French terms and definitions, but for the purpose of this analysis, due to the need to compare to American glossaries

which are only available in English, we have considered only the English terms of the glossaries. The same approach can be applied on the French part of those glossaries, or even glossaries in different languages.

A. Glossary Content Comparison

All five observed glossaries describe the same domain and therefore it is expected that the terms included in them are similar and that a high number of terms is defined in most glossaries. Some discrepancy between Canadian and American dictionaries may exist due to the slight differences of American and Canadian English. As a first step, we looked into the five glossaries to see if, and to which degree, our expectations of content similarity were correct.

From the five observed glossaries, four are relatively close in the size (Table I) and include between 115 and 167 term definitions, while ICDRM is significantly larger, including 572 term definitions. As opposed to the other four glossaries that are intended to be used in practice, ICDRM is established for the purposes of emergency management education and practice. ICDRM was created by the Institute for Crisis, Disaster, and Risk Management, at The George Washington University.

Table II depicts the overlap between the observed glossaries. Even though all five glossaries are relatively generic emergency glossaries, there is very little overlap between them. Only seven terms are defined in all five glossaries: *emergency, hazard, mitigation, preparedness, recovery, response* and *threat*. And only five terms appear in four glossaries: *communications, incident, incident management team, prevention* and *public information officer*. Some terms that could be considered significant disaster management terms, such as *risk assessment, disaster, crisis* and *alert*, are defined in only two glossaries. This demonstrates that a single glossary does not fully cover the domain and cannot be used as a standalone source for the creation of a disaster ontology. The high number of terms appearing in only one glossary is in part caused by the fact that the ICDRM glossary contains significantly more terms than the remaining four.

To further investigate the commonalities among glossaries, we observed the number of overlapping terms between pairs of glossaries. Table III shows the number of overlapping terms in pairs of glossaries and the percentage of relative overlap. The relative overlap is calculated as:

```
Relative overlap = (2 x number of overlapping
terms) / total number of terms in the pair of
glossaries x 100%.
```

The overlap between the Canadian (Emergency and Crisis Communication and EMO) and the American glossaries is relatively low – 10% or less. However, overlap between the two Canadian glossaries is still only 14%. A high number of term overlap occurs between the different American glossaries, with the highest between NIMS and ICS at 52%. This high term overlap can be explained by the fact that both ICS and NIMS are created by Federal Emergency Management Agency (FEMA).

TABLE I. GLOSSARY SIZE

	Emer. and Crisis Com	EMO	NIMS	ICDRM	ICS
# of terms	115	129	167	572	153

TABLE II. TERM OVERLAP BETWEEN GLOSSARIES

Number of glossaries term appears in	Number of terms
1	638
2	84
3	67
4	5
5	7

TABLE III. NUMBER OF OVERLAPPING TERM IN PAIRS OF GLOSSARIES AND RELATIVE OVERLAP.

	Emer. and Crisis Com	EMO	NIMS	ICDRM	ICS
# of terms	115	129	167	572	153
Emer. and Crisis Com		17 (14%)	13 (9%)	30 (9%)	10 (7%)
EMO			14 (9%)	34 (10%)	9 (6%)
NIMS				93 (25%)	84 (52%)
ICDRM					81 (22%)

There is a significant overlap of 25% between NIMS and ICDRM caused by the fact that ICDRM is developed with NIMS as its basis; moreover, a high number of ICDRM terms cite NIMS' definitions.

Among the five glossaries, we expected significant term overlap due to their shared domain and purpose, but the findings are on the contrary. Only high overlap is found between glossaries created within the same agency, ICS and NIMS glossaries. Overlaps between other pairs are generally low, and somewhat higher between pairs of glossaries from American sources.

Even for terms defined in all glossaries, definitions are often quite different across the glossaries; this is illustrated in Table IV with *Threat* as an example. In the five glossaries, there are five different definitions of term *Threat*. ICDRM gives two definitions of the term *Threat*, with one being the same as the definition in ICS.

In some situations, different terms have similar meanings. If we look at definition of the term *Hazard* in the ICDRM and NIMS glossary, 'Something that is potentially dangerous or harmful, often the root cause of an unwanted outcome,' it is very similar to the definition of the term *Threat* in the EMO glossary. The EMO glossary defines a *Threat* as, 'A person, thing or event regarded as a likely cause of harm or damage.' The two terms, *threat* and *hazard*, are defined as distinct terms in all five glossaries, but some of their definitions make it hard to distinguish between the meanings.

This analysis shows that even though the five glossaries deal with the same domain and have the same purpose, they are very different in the terms that they define as well as in term definitions.

TABLE IV. TERM *THREAT* IN DIFFERENT GLOSSARIES

Glossary	Term Definition
Emerg. and Crisis Comm.	The combination of the presence of a hazard and an exposure pathway.
EMO	A person, thing or event regarded as a likely cause of harm or damage.
NIMS	Natural or manmade occurrence, individual, entity, or action that has or indicates the potential to harm life, information, operations, the environment, and/or property.
ICDRM	An indication of possible violence, harm, or danger.
ICDRM	The possibility of a hazard occurrence; something that has the potential to cause harm.
ICS	An indication of possible violence, harm, or danger.

Therefore, the creation of a domain ontology needs to use a variety of domain glossaries to encompass vocabularies of different domain members and to achieve better domain coverage.

B. Glossary Structures used for Ontology Creation

Glossaries, as semi-structured documents, have structure and formatting that can be used to facilitate ontology creation. The typical structure of a glossary is a term or label followed by the term's definition. For example, '**Communications:** The process of transmission of information through verbal, written, or symbolic means.' In all five observed glossaries, the term label is distinguished from the rest of the text by bold font; for example, '**Hazard:** Something that is…'. In three sources, the term is separated from its definition by a colon, while the other two use new lines. Glossary terms are concepts significant for a domain of interest. Therefore, we extract glossary terms using source document formatting and create initial ontology concepts.

Acronyms are typically included in glossaries. In [3], acronyms are separated from the term by using a semicolon: '**business resumption planning; BRP**'. In the remainder of the observed glossaries, acronyms are in brackets following the term: '**Emergency Operations Center (EOC):**…**'**. This is used to extract properties of the concepts for the ontology. Some of the documents, such as EMO, NIMS and ICS also have a separate section for lists of the acronyms, where only the acronym and its meaning are listed. Often, this is a duplication of the acronym listed with the term definition.

Redirection is commonly used to lead from one term to another one: '**Action Plan:** See Incident Action Plan'. If the term does not have definition, but it only has redirection, redirection is used to lead to a synonym term where the description is specified. In the observed documents, redirection is performed through the use of the 'See' word preceding a redirecting term. Therefore, if the term does not have a description and it is followed by a redirecting 'See' word, the two terms are considered synonyms. For the identification of synonyms, EMO also uses the 'synonym' word in formatting such as '**full-scale exercise** (synonym: **field exercise**)'.

Some terms are described and also contain a redirecting 'See' word; for example: '***Competency***: A specific knowledge element… See "Proficiency"'. In this case, description of a redirecting term c*ompetency* and a redirected *proficiency* term are not the same, and therefore the terms are not synonyms. In this situation 'See' indicates a related or similar concept. In the presented example, 'See' indicates that term c*ompetency* is related to term *proficiency*. EMO uses 'See also' in lieu of 'See' from the presented example to indicate related terms: '**Incident Action Plan (IAP):** An oral or written plan ... See also "Action Plan."'. These two patterns are used for the creation of relations between ontology concepts. In the Emergency and Crisis Communication Vocabulary, relations among terms are given more significance than in the other observed documents. The abbreviation 'cf.' is used to identify a cross-reference to a related concept, each being separated by a semi-colon; for example, '**mitigation** … cf. emergency management; preparedness; recovery; response; resumption'. Of the 115 terms defined in this document, related term(s) are specified for 91. Commonly, several related terms are specified for a single term, bringing the total number of relations specified in this way to 242. Even though this pattern appears in only one of the observed glossaries, it is a significant resource for the creation of relations in an ontology.

Some glossaries distinguish among different meanings of a single term. The emergency and Crisis Communication Vocabulary uses number superscripts to indicate different meanings, while EMO uses numbers in brackets following the term: '**hazard (1)** A risk that is a threat. **hazard (2)** An event ...'. ICDRM uses bullets to specify different definitions, as in:

'Hazard:
- A potential or actual force, …
- Something that is …'

ICDRM has formatting features indicating taxonomic hierarchies. For example, the term *volunteer* is in the ICDRM, but its definition is not specified. The term *volunteer* is followed by bulleted list where each bullet specifies definitions of a specific kind of volunteer, such as *Accepted volunteer*, *Affiliated volunteer*, *Recruited volunteer* and others. This pattern defines a hyponym (is-a) relation where one concept is a subconcept of another concept; for example, *Affiliated volunteer* is a special kind of *Volunteer*. This is the only structural pattern in observed glossaries that we use for the extraction of hyponym relations.

IV. DISASTER ONTOLOGY

A glossary, in general, contains explanations of concepts relevant for certain field. Domain experts' knowledge was used in the process of the glossary creation and it is built into the glossary itself. Terms defined in the glossary are identified as relevant terms by the people and organizations that created the glossary. Other elements, such as synonyms, related terms, acronyms and subconcepts, are extractions of domain

knowledge as well. The structures of the glossaries represent a strong foundation for the creation of an initial domain ontology.

We use the formatting aspects of glossaries described in subsection III.B as a source of information for the ontology's creation. We use only the formatting elements to extract an ontology, without the use of any natural language processing techniques. Fig. 1 depicts the use of the formatting elements of glossaries in ontology learning. The left column shows the structural elements observed in one or more glossaries. It is followed by the example of each pattern from one of the observed glossaries. Those formatting patterns identify fragments of the glossaries that are translated into different ontology elements as shown in the ontology element column of the Fig. 1. Patterns used in the ontology creation process are only those found in the five disaster management dictionaries analyzed. The use of other glossaries, dictionaries or vocabularies may demand different or additional patterns depending on the structure of the source document.

Using a single glossary for ontology creation would limit the ontology to the view of the domain described by the glossary creator; this single view is likely to not be shared by other participants in the same domain. The coverage of the domain would also be limited. To alleviate this, we use the multiple glossaries described in Section III for the ontology creation.

Fig. 2 illustrates a fragment of the ontology created from the five observed glossaries using only their formatting elements without the use of any natural language processing techniques.

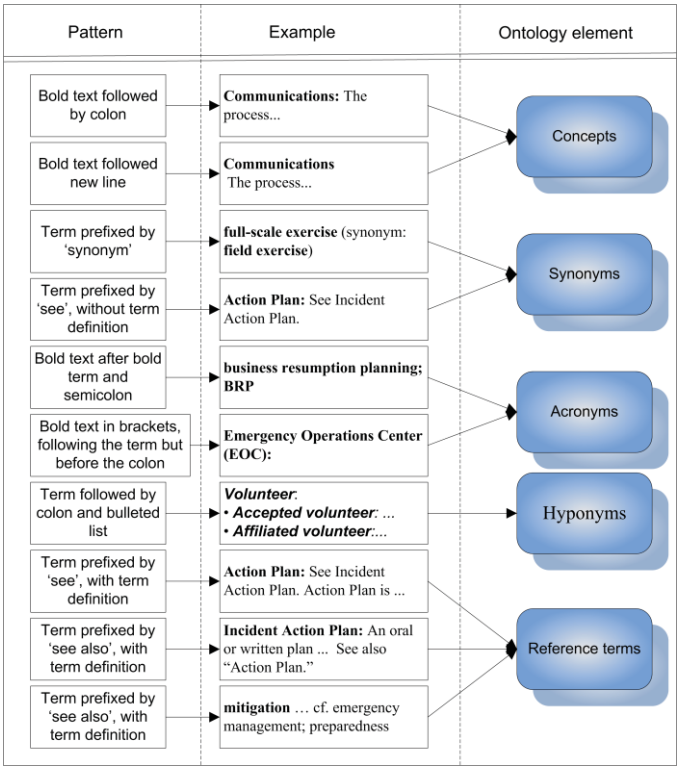


Figure 1. From formatting element to ontology component

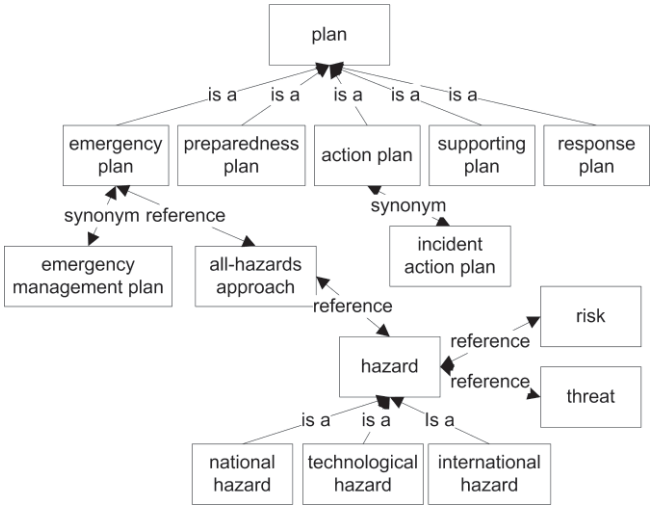


Figure 2. Fragment of ontology created from glossaries

The concepts *hazard* and *threat* appear in all five glossaries. The concept *Plan* is defined only in the ICDRM glossary, while the other glossaries contain definitions of more specific plans, such as *preparedness plan* and *response plan*. *Emergency management plan* is identified as a synonym of *emergency plan* and *action plan* is a synonym of *incident action plan*.

For use by machines and software systems, the created ontology can be represented in an ontology language of choice, such as OWL (Web Ontology Language), OIL (Ontology Interchange Language) or others. The choice of representation language does not change the ontology learning process, but it only changes how the ontology is represented for automatic processing.

V. Conclusions and Future Work

In disaster management, involvement of various teams with different views of the domain presents a significant challenge in achieving a successful communication process. A way of achieving a common understanding among teams is through ontologies. Even though this work focused on disaster management, similar situations exist in other fields where different views of the domain are expressed through a variety of glossaries.

We explore the use of glossaries as semi-formatted stores of domain knowledge for ontology creation. Domain experts' knowledge was used in the process of glossary creation and is built into the glossary in the form of content and formatting.

Not only do glossaries contain a list of terms that may be used for ontology creation, but they also contain additional structural patterns that we propose may also be used. In addition, we investigate the discrepancy between different views of the same domain expressed in domain glossaries. As a case study, we observe five glossaries from the disaster management domain; however, is it expected that glossaries from other domains will demonstrate similar properties. Even

though the five observed emergency management glossaries are relatively generic, their content overlap is very low, with only five terms appearing in all five. Even between two glossaries created by the same emergency management agency, term overlap is only 52%. This indicates that for ontology creation, it is preferable to use multiple domain glossaries from different sources. This will lead to better domain coverage and facilitate a true shared conceptualization.

Formatting of the documents is similar across observed glossaries. All five glossaries use similar formatting to distinguish between terms and their definitions, and similar methods to identify synonyms, acronyms and related terms. We use the formatting of documents to extract terms, synonyms, acronyms, hyponyms and referenced terms to create an initial ontology. The main advantage of this approach is the use of domain knowledge built into domain glossaries and the relative simplicity of the processing. The initial ontology created using this approach is lightweight, without considerable detail, but the quality of included terms and relations is high due to the high reliability of the source document. The created lightweight ontology can be enriched by applying further processing using statistical methods or natural language processing methods, such as the approach proposed by Navigli and Velardi [22].

The direction of the future work is towards fully utilizing the structure of semi-formatted documents for ontology learning. A rule engine that will enable specifying custom rules for the extraction of concepts and relations from generic semi-formatted documents needs to be created. This rule engine will enable the user to specify how formatting should be used in ontology learning. Because this approach creates lightweight ontologies, we want to integrate it with other ontology learning mechanisms from un-structured text. Also, a way of distinguishing between the significance of different source documents is needed; that is, a method that will account for source relevance and reliability.

ACKNOWLEDGMENT

Support for this work was provided by Canada's Advanced Research and Innovation Network (CANARIE) and Natural Sciences and Engineering Research Council (NSERC) of Canada.

REFERENCES

[1] S.M. Rinaldi, J.P. Peerenboom, T.K. Kelly. Identifying, Understanding, and Analyzing Critical Infrastructure Interdependencie, *Control Systems Magazine, IEEE.* Issue 6 vol.21, 2001, pp.11-25.

[2] E. Luiijf, E. Nieuwenhuijs, M. Klaver, M. van Eeten, E. Cruz, Empirical Findings on Critical Infrastructure Dependencies in Europe, Critical Information Infrastructure Security, Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2009, pp. 302-310.

[3] Emergency and Crisis Communication Vocabulary, Public Works and Government Services Canada, http://www.btb.gc.ca/publications/documents/crise-crisis.pdf.

[4] Emergency Management Ontario (EMO) English-French Glossary, Emergency Management Ontario, Ministry of Community Safety and Correctional Services, http://www.onterm.gov.on.ca/EMOlexicon.pdf.

[5] National Incident management System (NIMA) glossary, Federal Emergency Management Agency, United States Department of Homeland Security, http://www.fema.gov/emergency/nims/Glossary.shtm.

[6] ICDRM/GWU Emergency Management Glossary of Terms. The Institute for Crisis, Disaster, and Risk Management (ICDRM) at the George Washington University (GWU), Washington, D.C., http://www.gwu.edu.proxy2.lib.uwo.ca:2048/~icdrm/publications/PDF/EM_Glossary_ICDRM.pdf.

[7] Incident Command System (ICS) glossary, Federal Emergency Management Agency, United States Department of Homeland Security, http://training.fema.gov/EMIWeb/IS/ICSResource/assets/ICSGlossary.pdf.

[8] R. Studer , V.R. Benjamins, D. Fensel. Knowledge Engineering: Principles and Methods, *Data and Knowledge Engineering.* Issue 1-2 vol.25, 1998, pp.161-197.

[9] H.A. Rahman, M. Armstrong, D. Mao, J.R. Marti, I2Sim: A Matrix-Partition Based Framework for Critical Infrastructure Interdependencies Simulation, *Proceedings of the Electric Power Conference.* 2008, pp. 1-8.

[10] A. Usov, C. Beyel, E. Rome, U. Beyer, E. Castorini, P. Palazzari, et al., The DIESIS Approach to Semantically Interoperable Federated Critical Infrastructure Simulation, *Proceedings of the Second International Conference on Advances in System Simulation.* 2010, pp. 121-128.

[11] A. Tofani, E. Castorinia, P. Palazzaria, A. Usovb, C. Beyelb, E. Romeb, et al. Using Ontologies for the Federated Simulation of Critical Infrastructures, *Proceedings of the International Conference on Computational Science.* Issue 1 vol.1, 2010, pp.2301-2309.

[12] Y. Peng, W. Wenjun, D. Cunxiang, Application of Emergency Case Ontology Model in Earthquake, *Proceedings of the International Conference on Management and Service Science.* 2009, pp. 1-5.

[13] E. Castorini, P. Palazzari, A. Tofani, P. Servillo, Ontological Framework to Model Critical Infrastructures and their Interdependencies, *Complexity in Engineering.* 2010, pp. 91-93.

[14] J. Völker, P. Haase, P. Hitzler, Learning Expressive Ontologies, *Proceedings of the 2008 conference on Ontology Learning and Population: Bridging the Gap between Text and Knowledge.* vol. 167,Issue. 2/8/2010,2008, pp. 45-69.

[15] A. Maedche, S. Staab, Discovering Conceptual Relations from Text, *Proceedings of the 13th European Conference on Artificial Intelligence.* 2000, pp. 321-325.

[16] M.A. Hearst, Automatic Acquisition of Hyponyms from Large Text Corpora, *Proceedings of the 14th conference on Computational linguistics.* vol. 2,1992, pp. 539-545.

[17] A. Gómez-Pérez, D. Manzano-Macho. An Overview of Methods and Tools for Ontology Learning from Texts, *The Knowledge Engineering Review.* Issue 3 vol.19, 2004, pp.187–212.

[18] Y. Zhao, J. Li, Domain Ontology Learning from Websites, *Ninth Annual International Symposium on Applications and the Internet.* 2009, pp. 129-132.

[19] L. Karoui, M. Aufaure, N. Bennacer, Contextual Concept Discovery Algorithm, *Proceedings of the Twentieth International Florida Artificial Intelligence Research Society Conference.* 2007, pp. 460-465.

[20] H. Davulcu, S. Vadrevu, S. Nagarajan, I.V. Ramakrishnan, OntoMiner: Bootstrapping and Populating Ontologies from Domain-Specific Web Sites, *Intelligent Systems, IEEE.* vol. 18,Issue. 5,2003, pp. 24-33.

[21] K. Shinzato, K. Torisawa, Acquiring Hyponymy Relations from Web Documents, *Proceedings of North American Chapter of the Association for Computational Linguistics - Human Language Technologies NAACL-HLT.* 2004, pp. 73-80.

[22] R. Navigli, P. Velardi, From Glossaries to Ontologies, Extracting Semantic Structure from Textual Definitions, *Proceeding of the 2008 conference on Ontology Learning and Population: Bridging the Gap between Text and Knowledge.* vol. 167,2008, pp. 71-104.

[23] A. García-Silva, A. Gómez-Pérez, M.C. Suárez-Figueroa, B. Villazón-Terrazas, A Pattern Based Approach for Re-Engineering Non-Ontological Resources into Ontologies, *Lecture Notes in Computer Science.* vol. 5367 LNCS,2008, pp. 167-181.

Packaging Controlled Experiments Using an Evolutionary Approach Based on Ontology

Lilian Passos Scatalon, Rogério Eduardo Garcia, Ronaldo Celso Messias Correia
Departamento de Matemática, Estatística e Computação
Faculdade de Ciências e Tecnologia – Universidade Estadual Paulista "Júlio de Mesquita Filho"
Rua Roberto Simonsen, 305 – CEP 19060-900 – Presidente Prudente - SP, Brazil
lilian.scatalon@gmail.com, rogerio@fct.unesp.br, ronaldo@fct.unesp.br

Abstract

A body of knowledge in Software Engineering requires experiments replications. The knowledge generated by a study is registered in the so-called lab package, which must be reviewed by an eventual research group with the intention to replicate it. However, researchers face difficulties reviewing the lab package, what leads to problems in share knowledge among research groups. Besides that, the lack of standardization is an obstacle to the integration of the knowledge from an isolated study in a common body of knowledge. In this sense, ontologies can be applied, since they can be seen as a standard that promotes the shared understanding of the experiment information structure. In this paper, we present a workflow to generate lab packages based on $_{EXPER}Ontology$*, an ontology of controlled experiments domain. In addition, by means of lab packages instantiation, it is possible to evolve the ontology, in order to deal with new concepts that may appear in different lab packages. The iterative ontology evolution aims at achieve a standard that is able to accommodate different lab packages and, hence, facilitate to review and understand their content.*

Keywords: Controlled Experiment, Experimental Software Engineering, Ontology, Knowledge Representation.

1. Introduction

Controlled Experiment in Software Engineering (SE) attempts to assess methods, techniques and tools applied on software development activities [5, 4]. By using a similar model of practitioners building software, subjects apply methods, techniques and tools under controlled environment, producing data that allow evaluating, measuring and comparing their performance under pre-defined conditions. The collected data set leads to conclusions that are meaningful on controlled conditions, considering the population from which subjects are representative. However, results from a single experiment cannot establish definitive facts about a phenomenon due to variations introduced by different system domains, personal background and experience and cultural environments [6, 21, 23, 27, 20]. Gaining insight into such variations requires running multiple independent studies on a topic [26, 20] – for example, varying the subjects profiles, the adopted procedures or even the experimental design help on establish knowledge about a topic [22, 20]. So, by executing multiple experiments that address these variations, researchers build knowledge on SE discipline, as well as help the practitioner understand how to build software systems better [4].

In this sense, the *International Software Engineering Research Network* (ISERN) was formed with researchers that promote SE research in an experimental context. According to the ISERN Manifesto [17], in order to build basic models and components on SE discipline, it is important to consider characteristics from specific environments, since each one imposes variations in the effects of technologies.

Replications of a study that investigates a technology by different research groups can deal with those execution variations, allowing to draw conclusions that reach a broader context and, thus, consolidating knowledge in SE. Replicate a study depends on the effective review of its lab package, to understand the adopted procedures and guarantee process conformance with the previous experiment [6, 28]. The description of a study – including the procedures, the results and conclusions – is registered in the lab package [26]. The lab package carries the knowledge to be transferred among researchers in order to enable replications and to report the experimental findings, aimed to contribute to the advance of the discipline and the application in industry.

However, knowledge sharing problems among research groups arise, such as difficulties in reviewing lab packages [26] and integrating this knowledge in a common body [18],

mostly because the lack of standardization of lab packages. Dealing with such problems minimizes the risk of an experiment being isolated. Noticing how those problems have influenced on replications, Mendonça et al. [20] presented the *Framework for Improving the Replication of Experiments* (FIRE). The FIRE suggests standardizing packages and evolving knowledge repositories in order to share knowledge, which should be available in a common body and also understandable to researchers willing to execute a replication.

In this direction, Garcia et al. [14] elaborated the $_{EXPER}Ontology$: an ontology of controlled experiments domain, with the purpose of formally describe concepts that compose a lab package. Ontologies have already been applied in Software Engineering aiming at the standardization of the concepts of a domain and knowledge reuse and share [12, 13, 2, 7, 3].

The packaging of information about a study can incorporate the ontology as a standard in order to deal with the complications in integrating and transferring the knowledge from experiments, considering that knowledge representation and sharing is a recurring application to which ontologies are designed for [15, 29]. According to Amaral and Travassos [1], packaging should be done phase by phase, throughout the experimental process [31]. So, in this context, it is proposed a workflow that suggests this packaging based on the concepts defined in $_{EXPER}Ontology$. And, since its definitions are not necessarily static, the ontology can absorb new concepts, as lab packages containing different sets of information are instantiated.

The remainder is organized as follows. In Section 2 is given an overview of replications importance and is presented the problem in knowledge sharing among research groups. With the purpose to address such problems, in Section 3 is argued about the use of an ontology to represent the knowledge from a study. In Section 4 is presented the proposed workflow that applies and evolves the ontology as a standard to lab packages. Finally, in Section 5, the contributions and future work directions are summarized.

2. Replications, Knowledge Transfer and Integration

Several issues involved in running an experiment present sources of variations that limit the generalization to apply the conclusions. The participants may be from different cultural environments or imposed to a different set of conditions during the execution [22, 20]. Therefore, in order to generalize the conclusions, these variations should be explored and dealt with in replications of the study.

To effectively attend to the conditions variation, different research groups might execute replications in their own laboratory environment. Thereby, if obtained results confirm previous conclusions, it is demonstrated that they are applicable to a broad context. Conversely, if the results do not confirm the conclusions of the previous study, the influence of the variations can be identified and produce other relevant conclusions on the topic. Therefore, the knowledge about a technology can be consolidated across replications based in the study that investigates it. Besides, the selected variables to model the phenomenon or the adopted procedures in the realization may not be ideal. In this sense, similar replications, accomplished by the original experimenters or not, can build confidence in the procedure and the result [8].

Replicating a study, in a not completely independent way, depends on the effective review of its original execution, to understand the procedures and the experimental design of the previous study for process conformance sake [28], that is, to produce comparable results. The lab package is the tool that carries study information. Thus, replications can be encouraged with the availability of lab packages [26]. And, given the amount of effort required to conduct an experiment, it is reasonable to facilitate the reuse of experimental artifacts by providing a lab package [8]. From its review, researchers have access to how the previous study was conducted.

However, the review of lab packages by researchers presents difficulties [26], what can be considered an obstacle in knowledge transfer among research groups. Noticing knowledge transfer problems as barriers to the conduction of replications, Mendonça et al. [20] proposed the *Framework for Improving the Replication of Experiments* (FIRE), which is composed by the two cycles of activities, as illustrated in Figure 1. The internal cycle represents the execution of a study by a research group. The external cycle (inter-group) represents the activities whose purpose is to integrate the knowledge generated by running a study (internal cycle) in a common body of knowledge. In that sense, the lab package can be effectively reviewed by eventual replicators from a different research group aimed to set the experiment (replication) goals. One might observe in Figure 1 that *criate/evolve package* influences the external activity *share knowledge*. Likewise, in the other point that the cycles intercept, *understand lab packages* is crucial to *set experiment goals* of a new replication.

It can be considered, therefore, that the lab package represents the output of the internal cycle carrying out information about an isolated study, and also the input of the internal cycle of a possible replication, considering that researchers should review and understand the original experiment lab package. So, with regard to *share knowledge* and favor better understanding of lab packages (in the external cycle), the way which the lab package information must be represented is important, what is endorsed by the activity of *standardize packages*.

In the other hand, activities of the external cycle of FIRE

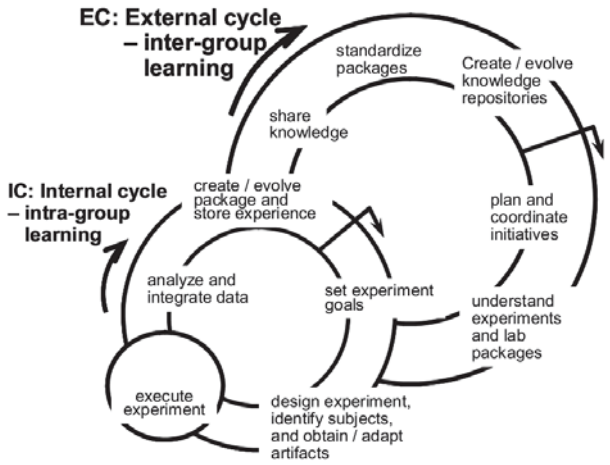


Figure 1. The FIRE [20]

also suggest to integrate the knowledge generated by an isolated study in a common body. This established body of knowledge may support decision making to practitioners in software development process [18] and the available lab packages should supply an experimental infrastructure to support future replications as well [26], whose results can sustain the progress of such body of knowledge. However, a major problem for this integration is the heterogeneity in the way that experiments are reported [18], which present different sets of information or level of details [9]. Accordingly to FIRE, to integrate the knowledge or to *evolve knowledge repositories*, it is necessary to *standardize packages* (see Figure 1). Indeed, Carver [9] highlights the importance of standardization to deal with this heterogeneity.

3. Ontologies as the Common Solution

Shull and his partners have pointed out that lab packages are important tools for supporting replications (easy-available designs and materials facilitate replications by reducing the amount of effort required from independent researchers), but well-designed lab packages are crucial for facilitating better and comparable replications [28]. Therefore, it must be promoted well-defined knowledge representation into the lab packages. Additionally, there is not a broadly adopted standard to report experiments in Software Engineering, even though guidelines have been proposed [19, 18]. So, those issues suggest the establishment of a structure for lab packages, that also support a better understanding to reviewers. In this sense, an ontology can be used to describe the model of information that compose a lab package, aiming at provide a standard and favor a better understanding.

Ontologies are means to formally represent the structure of a specific domain, by defining explicitly entities and their relationships that emerge from its observation [16]. The concepts interpretation is restricted by axioms, and thus the ontology-based information carries embedded meaning, allowing agents from different contexts to interpret it consistently.

Considering the role of ontologies in knowledge representation and sharing [15, 24], and problems related to knowledge integration and transfer in Experimental Software Engineering, Garcia et al. have proposed an ontology for the controlled experiments domain: the $_{EXPER}Ontology$ [14]. Basically, this ontology helps to elucidate concepts and relations concerning controlled experiments. Garcia et al. [14] presented the conceptualization in two levels of refinement and axioms to formalize it. Concepts like *controlled experiment*, *replication*, *experiment validity* and *lab package* are defined in the first level. The second level comprises the refinement of concepts on the first one. The main aspect is a more accurate definition of lab package, which defines and relates concepts from the experimentation process [31], that is, concepts whose instances are information of the lab package.

The main goal on using the $_{EXPER}Ontology$ is to incorporate the benefits that ontologies provide to the knowledge transfer by organizing lab packages: the shared understanding of information [24], what is essential to allow transferring effectively the knowledge embedded into a lab package. Besides, the ontology itself – with concept definitions – can be considered as standard of information organization that the lab package has to include.

4. The Proposed Evolutionary Approach Based on Ontology

Several experiments have been conducted and their results published, but each researcher publishes what considers most important in the experiment report [18]. There is not a broadly adopted standard to lab packages. In this context, the $_{EXPER}Ontology$ was proposed to deal with the necessity of a established standard, that also eases the understanding of lab package information. With the purpose to apply $_{EXPER}Ontology$, initially was proposed the instantiation of lab packages information based on $_{EXPER}Ontology$ concepts, according to the experimental process [31, 1]. To support this initial approach, the $_{EXPER}Ontology$ was implemented using OWL (Web Ontology Language) and, then, a tool was developed to instantiate the concepts [25]. However, during the tool validation, it was faced the lack of standardization problem, what leads to the difficulties in knowledge integration, as stated in Section 2.

Thus, not only the use of $_{EXPER}Ontology$ in packaging experiment data set should be considered, but also the current reporting practice, since it is usual lab packages con-

taining different sets of information. For example, details about treatments might be missing, becoming unable to apply ontology concepts straightly. So, it is necessary to deal with different sets of information to accommodate them according to $_{EXPER}Ontology$.

Despite problems found during instantiation using the tool [25], the ontology-based packaging proved itself to be appropriate, since the lab packages were generated in a format that makes explicit the meaning of the information represented, what enables machine processing – the ontology is expressed in a language with formal semantics [30]. However, considering the need for a standardization that complies with the inconsistency of published experiments [9], the $_{EXPER}Ontology$ should address the different manners in which an experiment execution is registered. This suggests that the initial ontology should evolve, incorporating new concepts.

In this direction, we propose a workflow, which includes the evolutionary approach to the lab packages instantiation using the $_{EXPER}Ontology$. As illustrated in Figure 2, the inputs of the workflow are the lab package to be instantiated and the $_{EXPER}Ontology$ – the concepts descripted and their relationships are used as parameters to instantiate the lab package. Each phase of experimental process has its own goal and concepts defined in $_{EXPER}Ontology$. Following experimental process, the concepts are used as parameters to a mining process aimed to match corresponding information, which is initially presented without a standard. Thereby, it is possible to instantiate information in the OWL lab package.

To describe the workflow activities, the concepts defined to Planning and Definition phases in the ontology for Lab Packages are presented throughout the experimentation process, highlighted in the following. At first, the **initial hypothesis** of a controlled experiment is established. It is composed by the **object of study**, in agreement with a **purpose**, under a **quality focus**, and in a specific **context**. The *Definition* phase is the basis for the *Planning* phase and the **initial hypothesis** generates the **hypotheses formalized**. These hypotheses have **null hypothesis** and the **alternative hypothesis**, as attributes. From the **hypothesis formalized**, the experimenter defines the experiment variables – **dependent** and **independent variables**. During the planning phase s/he also defines the **experiment objects**: **technologies** to be studied (**techniques**, **methods** or **tools**) and **artifacts** (**documents**, **tools** or **forms**) to be used. Each **subject** has his/her **profile** recorded to characterize his/her background. Capturing the subject background aims at identifying possible influence on results. For instance, previous knowledge about experiment objects or domain application might influence the results obtained. The subjects' profile must be considered to create the **experimental design**, which is built combining experiment objects, independent variables and subjects, in agreement with the hypothesis under investigation. In addition, the subjects' profile must be considered in analysis. Based on the experimental design, an **execution plan** must be elaborated in order to describe the entire controlled environment to conduct the experiment.

For instance, conducting the mining process focusing on Definition phase, the concepts Purpose and Context are essential to compose the Initial Hypothesis (parameters to mining process). And considering the experiments published by De Lucia et al. [11] as data set (lab package), after mining process we obtained "Compare ER and UML class diagrams in data modeling" and "Academic" as information corresponding to **Purpose** and **Context**, respectively.

Additionally, the mining process might fail: or the mining do not match meaningful values to concepts; or do not match at all. This situation might be consequence of missing data in the lab package, what indicate to experimenters that s/he should handle during the activities of FIRE internal cycle. Also, this situation might be consequence of missing concept, i.e., the $_{EXPER}Ontology$ have to be updated in order to evolve it. Considering the same experiment from the previous example [11], the researchers applied feedback questionnaires after tasks execution. The information of subject feedback analysis did not match properly to the concept Questionnaire, since Questionnaire concept is defined in the $_{EXPER}Ontology$ to represent subject profile, and its relation with feedback analysis was missing. This mechanism of discover and insert missing concepts automatically is facilitated by the ontology formal description, which enables machine processing.

The instantiation of several lab packages following the proposed workflow allows capturing missing concepts and, consequently, evolve the $_{EXPER}Ontology$. This can be accomplished similarly as the tool validation, by using information extracted from published experiments [10], but only after a comprehensive review of the literature. As the lab packages are instantiated through the workflow, the ontology approximates to the semantic standard that allows to accommodate variations on lab packages, and thereby, the $_{EXPER}Ontology$ evolution improves its application on packaging experiments as well.

5. Conclusions

In this paper we propose a workflow to apply and evolve the $_{EXPER}Ontology$ to package controlled experiments in Software Engineering. The use of an ontology as a standard to instantiate experiment information was already implemented [25] aiming at improve the understanding of lab packages, since problems concerning their review by researchers were pointed out [26].

The validation of such approach has shown that its main

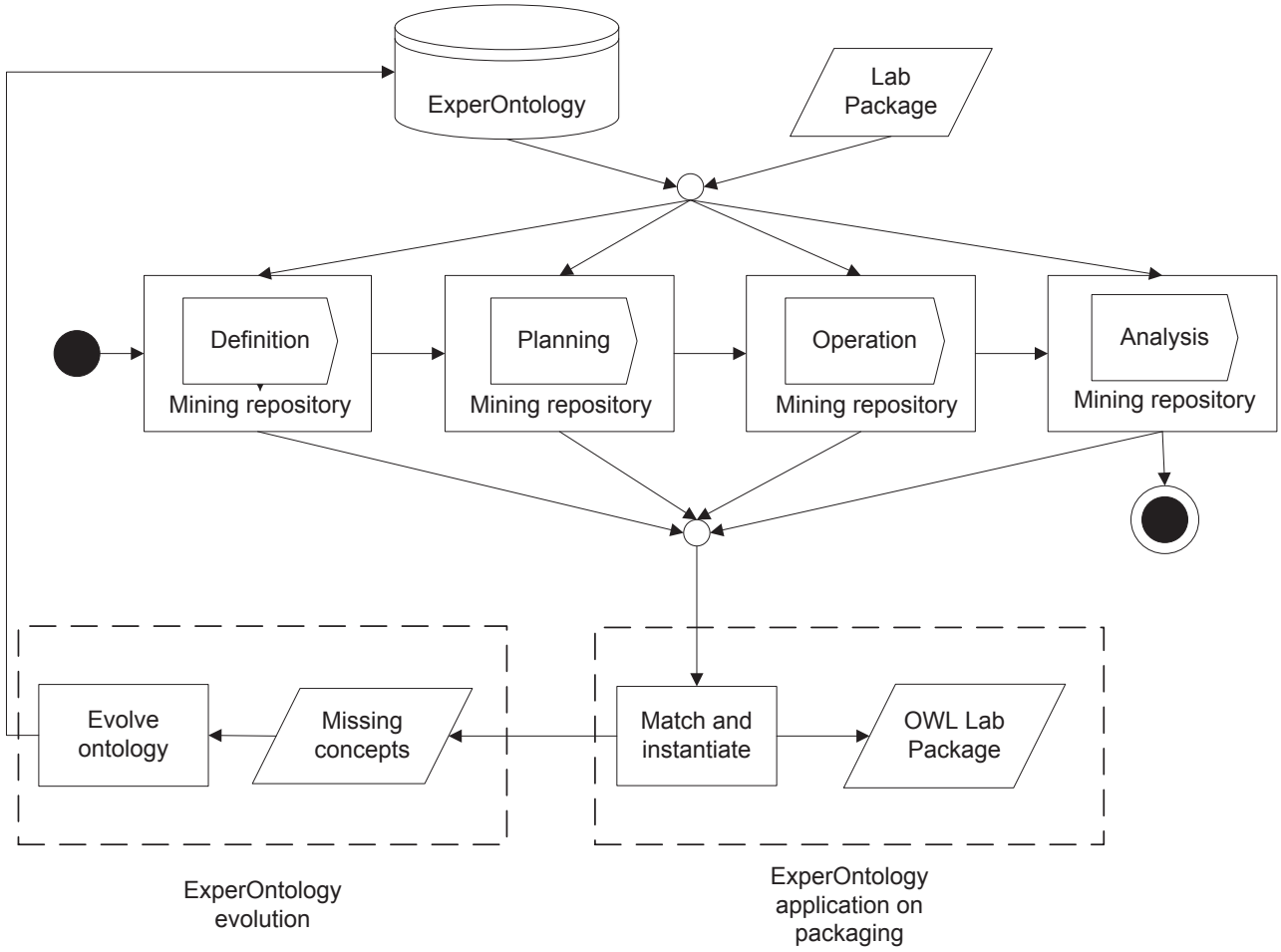


Figure 2. The evolutionary approach proposed for Packaging Controlled Experiments

contribution is the creation of lab packages according to a structured and organized way that makes explicit and meaningful the represented information. Also, ontologies enable machine processing, since are expressed in a language that presents formal semantics [30]. As pointed out by Jedlitschka et al. [18], experiments reports in practice contain different sets of information, what produced the requirement to evolve the ontology, in order to accommodate variations on the sets of information that lab packages can present. This heterogeneity entangles the integration of the knowledge generated by each isolated study in a common body [18]. In this sense, the $_{EXPER}Ontology$ can be seen as a unifying model, in which lab packages are instantiated, therefore enabling integration. Also, *create/evolve package* activity proposed on FIRE (see Figure 1) suggests that the initial ontology should evolve, incorporating new concepts, what is addressed by the proposed approach. Consequently, a semantic standard that deal with the heterogeneity issue can be accomplished.

As input to the workflow we consider the lab package as the data set describing an experiment. In this sense, any experiment description might be used: for example, an experiment description found in literature. In this case, the workflow should be applied in parallel with a literature review, in order to apply the information extraction from papers describing experiments [10]. Through this mechanism, it is also possible to assess the new concepts being inserted.

References

[1] E. A. G. Amaral and G. H. Travassos. A package model for software engineering experiments. In *Proceedings of ISESE 2003 - International Symposium on Empirical Software Engineering*, pages 21–22, 2003.

[2] E. F. Barbosa, E. Y. Nakagawa, and J. C. Maldonado. Towards the establishment of an ontology of software testing. In *SEKE (Software Engineering & Knowledge Engineering)*, pages 522–525, 2006.

[3] E. F. Barbosa, E. Y. Nakagawa, A. C. Riekstin, and J. C. Maldonado. Ontology-based development of testing related tools. In *SEKE (Software Engineering & Knowledge Engineering)*, pages 697–702, 2008.

[4] V. Basili. *Empirical Software Engineering Issues, LNCS 4336*, chapter The Role of Controlled Experiments in Software Engineering Research, pages 33–37. Springer Verlag, 2007.

[5] V. Basili and M. Zelkowitz. Empirical studies to build a science of computer science. *Communications of the ACM*, 50(11):33–782, 2007.

[6] V. R. Basili, F. Shull, and F. Lanubile. Building knowledge through families of experiments. *IEEE Transactions on Software Engineering*, 25(4):456–473, 1999.

[7] J. Biolchini, P. G. Mian, A. C. C. Natali, T. U. Conte, and G. H. Travassos. Scientific research ontology to support systematic review in software engineering. *Advanced Engineering Informatics*, 21(2):133–151, 2007.

[8] A. Brooks, M. Roper, M. Wood, J. Daly, and J. Miller. *Guide to Advanced Empirical Software Engineering*, chapter Replication's Role in Software Engineering, pages 365–379. Springer-Verlag London, 2008.

[9] J. C. Carver. Towards reporting guidelines for experimental replications: A proposal. In *International Workshop on Replication in Empirical Software Engineering Research (RESER)*, 2010.

[10] D. Cruzes, M. Mendonça, V. Basili, F. Shull, and M. Jino. Extracting information from experimental software engineering papers. In *Proceedings of the XXVI International Conference of the Chilean Society of Computer Science*, pages 105–114, Washington, DC, USA, 2007. IEEE Computer Society.

[11] A. De Lucia, C. Gravino, R. Oliveto, and G. Tortora. An experimental comparison of ER and UML class diagrams for data modelling. *Empirical Software Engineering*, 15(5):455–492, 2010.

[12] R. Falbo, F. B. Ruy, and R. D. Moro. Using ontologies to improve knowledge integration in software engineering environments. In *World Multiconference on Systemics, Cybernetics and Informatics*, 1999.

[13] R. Falbo, F. B. Ruy, and R. D. Moro. Using ontologies to add semantics to a software engineering environment. In *SEKE (Software Engineering & Knowledge Engineering)*, pages 151–156, 2005.

[14] R. E. Garcia, E. N. Hohn, E. F. Barbosa, and J. C. Maldonado. An ontology for controlled experiments on software engineering. In *SEKE (Software Engineering & Knowledge Engineering)*, pages 685–690. Knowledge Systems Institute Graduate School, 2008.

[15] T. R. Gruber. Toward principles for the design of ontologies used for knowledge sharing. *Int. Journal Human-Computer Studies*, 43:907–928, December 1995.

[16] N. Guarino, D. Oberle, and S. Staab. *Handbook on Ontologies*, chapter What is an ontology?, pages 01–17. Springer Verlag, 2009.

[17] ISERN. ISERN manifesto. Available at http://isern.iese.de.

[18] A. Jedlitschka, M. Ciolkowski, and D. Pfahl. *Guide to Advanced Empirical Software Engineering*, chapter Reporting Experiments in Software Engineering, pages 365–379. Springer-Verlag London, 2008.

[19] B. Kitchenham, S. L. Pfleeger, L. M. Pickard, P. W. Jones, D. C. Hoaglin, K. E. Emam, and J. Emam. Preliminary guidelines for empirical research in software engineering. *IEEE Transactions on Software Engineering*, 28:721–734, August 2002.

[20] M. G. Mendonça, J. C. Maldonado, M. C. F. de Oliveira, J. Carver, S. C. P. F. Fabbri, F. Shull, G. H. Travassos, E. N. Hohn, and V. R. Basili. A framework for software engineering experimental replications. In *ICECCS*, pages 203–212, 2008.

[21] J. Miller. Can results from software engineering experiments be safely combined? In *Proceedings of the 6th International Symposium on Software Metrics*, pages 152–, Washington, DC, USA, 1999. IEEE Computer Society.

[22] J. Miller. Applying meta-analytical procedures to software engineering experiments. *Journal of Systems and Software*, 54:29–39, September 2000.

[23] J. Miller. Replicating software engineering experiments: a poisoned chalice or the holy grail. *Information & Software Technology*, 47:233–244, March 2005.

[24] N. F. Noy and D. L. McGuinness. Ontology development 101: A guide to creating your first ontology. Technical report, Stanford University, 2001.

[25] L. P. Scatalon, R. E. Garcia, and R. C. M. Correia. Uma ferramenta para instanciação de ontologia de experimentos controlados em engenharia de software (in portuguese). In *6th Iberian Conference on Informations Systems and Technology (CISTI 2011)*, 2011. Accepted (to appear).

[26] F. Shull, V. R. Basili, J. Carver, J. C. Maldonado, G. H. Travassos, M. G. Mendonça, and S. C. P. F. Fabbri. Replicating software engineering experiments: Addressing the tacit knowledge problem. In *Proceedings of the 2002 International Symposium on Empirical Software Engineering*, pages 7–16, Washington, DC, USA, 2002. IEEE Computer Society.

[27] F. Shull, J. Carver, G. H. Travassos, J. C. Maldonado, R. Conradi, and V. Basili. *Lecture Notes on Empirical Software Engineering*, chapter Replicated Studies: Building a Body of Knowledge about Software Reading Techniques, pages 39–84. World Scientific Publications, 2003.

[28] F. Shull, M. G. Mendonça, V. R. Basili, J. Carver, J. C. Maldonado, S. C. P. F. Fabbri, G. H. Travassos, and M. C. Ferreira. Knowledge-sharing issues in experimental software engineering. *Empirical Software Engineering: An International Journal*, 9:111–137, March 2004.

[29] M. Uschold and M. Gruninger. Ontologies: Principles, methods and applications. *Knowledge Engineering Review*, 11:93–136, 1996.

[30] W3C. W3C Recommendation. OWL Web Ontology Language Semantics and Abstract Syntax. Available at http://www.w3.org/TR/owl-semantics, 2004.

[31] C. Wohlin, P. Runeson, M. Höst, M. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in Software Engineering: An introduction*. Kluwer Academic Publishers, Boston, USA, 1999.

Knowledge Engineering in the domain of Carbon Dioxide Capture Process System

Q. Zhou, A.J. Wiebe, C. W. Chan
Energy Informatics Laboratory, Faculty of Engineering and Applied Science
University of Regina
Regina, Saskatchewan, Canada, S4S 0A2
Christine.chan@uregina.ca

***Abstract*—This paper presents knowledge engineering work for building a knowledge-based system (KBS) for the amine-based CO_2 capture process system at International Test Center for CO_2 capture (ITC) located in Regina, Saskatchewan of Canada. The knowledge acquisition process was conducted for building knowledge base for the CO_2 capture process system. The knowledge base includes static and dynamic knowledge. The static knowledge includes the information on constructive components of the reaction instruments, fluids, and the control devices in the CO_2 capture system. The dynamic knowledge specified operation tasks of the CO_2 capture process system, which are expressed as the control strategies for dealing with 25 critical process parameters when a fault condition emerges. The knowledge obtained is represented in the knowledge table. In this paper, we present our work in knowledge engineering and how the knowledge table provided the foundation for developing an expert system for automatic monitoring, control, and fault diagnosis of the CO_2 capture process system at ITC and for implementing the ontology for the CO_2 capture process system.**

Keywords-knowledge engineering; ontology; knowledge-based expert system; CO_2 capture

I. INTRODUCTION

Combustion of fossil fuels in power generation and in various industrial processes such as cement manufacture and hydrogen production emits large amounts of CO_2. It is reported that CO_2 is a primary greenhouse gas and causes enhanced global warming. Due to increasing public concern about environmental pollution and climate change, the technology of amine-based carbon dioxide (CO_2) capture has been widely adopted for reducing industrial CO_2 emissions and mitigating global warming. The amine-based CO_2 capture process system is a complicated chemical process system which involves over a dozen reaction instruments and a number of control devices such as pumps and valves. The operators of the CO_2 capture process system are required to detect the abnormal condition in a timely manner and to fix the problem promptly so as to ensure reliable performance of the process system. Therefore, developing a systematic knowledge depository which collects both the domain and operation knowledge of the CO_2 capture process system is useful. A KBS can help the novice operators in monitoring and diagnosing the system. This paper discusses the process of knowledge acquisition (KA) and development of the knowledge base for the CO_2 capture process system. As a result of the KA process, a knowledge table was generated, which formed the basis for the conceptual model of the KBS as well as an implemented ontology of the domain.

The paper is organized as follows: Section 2 gives a brief description about the CO_2 capture process system at ITC. Section 3 describes the knowledge acquisition process and presents the conceptual structure of the developed knowledge base. Section 4 discusses the mapping of the elements of the knowledge base to the structure of DeltaV Simulate (a trademark of Emerson Corp., USA). Section 5 presents a case study of the expert system developed on DeltaV Simulate. Section 6 describes the development of the implemented ontology for the CO_2 capture process system based on the conceptual knowledge base. Section 7 gives a conclusion and includes some discussion about future work.

II. PROBLEM DOMAIN

In the amine-based CO_2 capture process, an amine solvent is used to absorb CO_2 from the flue gas, and CO_2 is subsequently extracted from the amine solvent, which can then be regenerated and reused. The amine-based CO_2 capture technology has been implemented at the International Test Center for CO_2 capture (ITC); the CO_2 capture process system is the problem domain for the KBS. More details of the CO_2 capture process system can be found in [6].

III. KNOWLEDGE ACQUISITION

The objective of the knowledge acquisition process is to obtain systematic knowledge about the CO_2 capture process system, and identify the knowledge needed for performing monitoring and diagnosis. Knowledge acquisition for this project lasted six months and involved discussions with the experts during interviews, observation, and field studies on site. During the interviews, the expert provided illustrations, answered questions, and verified the acquired knowledge. The result of the knowledge acquisition process was analyzed based on the Inferential Modeling Technique (IMT) [1], and the knowledge base of the CO_2 capture process system was developed. The knowledge base includes two parts: static knowledge and dynamic knowledge. The static knowledge defines the constructive components of the CO_2 capture plant

and specifies the reaction instruments, process parameters, and control devices. The dynamic knowledge was developed by representing the problem-solving skills of the experienced process operators, which included the diagnosis and remedial control actions for the 25 critical parameters when an abnormal condition is detected.

A. Static Knowledge

The static knowledge includes three classes: reaction instruments, process parameters, and control devices; they are discovered as follows.

1) Reaction Instruments

There are in total 16 reaction instruments in the CO_2 capture process. Each reaction instrument carries out a specific function and produces different output fluids. The sixteen reaction instruments are divided into three subclasses of objects according to their functions: (1) Pre-treatment section, which includes the steam boiler, micro turbine, inlet-gas scrubber; (2) Absorption-based CO_2 section, which includes the absorber, off-gas scrubber, lean amine storage tank, lean amine cooler, rich amine vessel, lean/rich amine exchanger, stripper, reboiler, and reclaimer; (3) Post-conditioning section for product purification, which includes the reflux condenser, reflux accumulator, CO_2 wash scrubber, and CO_2 dryer unit.

2) Process Parameters

The fluids circulating in the system include: flue gas containing CO_2, off gas free from CO_2, CO_2, steam, amine solvent, and water. They go through and undergo changes in different reaction instruments. The attributes of the fluids are defined as the process parameters which reflect performance of the plant, and there are in total over one hundred process parameters in the CO_2 capture plant. The values of the attributes are constantly changing in the process, even for the same type of fluid. The design decision was made to divide the attributes of each fluid into groups based on their locations, or which reaction instrument they have gone through. Therefore, the attributes of fluids have been defined as the attributes of the reaction instrument from which it flows. In this way, over one hundred process parameters in the plant can be grouped into 16 reaction instrument based groupings.

3) Control Devices

The control devices include the two classes of pumps and valves, which are used to regulate the flow of fluid. The pumps can be either opened or closed, and the valves can be opened, closed, or partially closed. Manipulating the pumps and valves can regulate the flow of fluids and change their attributes, or the process parameters. As the process parameters are divided into 16 classes according to the reaction instrument to which they belong, the control devices can be divided in the same way based on the process parameters that they regulate, i.e., 16 groups of control devices based on which reaction instrument they control.

As a result, the entire system can be viewed in terms of sixteen reaction components, their relevant attributes, and the relevant control devices. The relationships among these classes of the CO_2 capture process system are illustrated in Figure 1. The objects are denoted with the square boxes, and their respective attributes are denoted with the ellipse boxes. The object of reaction instrument processes some input fluids and produces the output fluids. The attributes of its output fluid are specified as this reaction instrument's attributes, which are represented by the dotted arrows. Also, the reaction instrument has its own attribute of fluid level, which is represented by the solid arrows. These attributes can be divided into the subclasses of temperature, pressure, flow rate, and level. The arrows pointing from the attributes of reaction instrument to the control devices indicate that these process parameters are directly controlled by the pumps and valves. The pumps and valves can be further classified by their own attributes. These attributes include: (1) type, which defines control mechanism of the valve/pump; (2) fluid, which specifies what fluid the valve/pump controls; (3) state, which specifies the status of a 2-state control device, it has the values of On and Off; (4) location, which indicates where the valve/pump is installed; and (5) name, which is a brief description of the valve/pump.

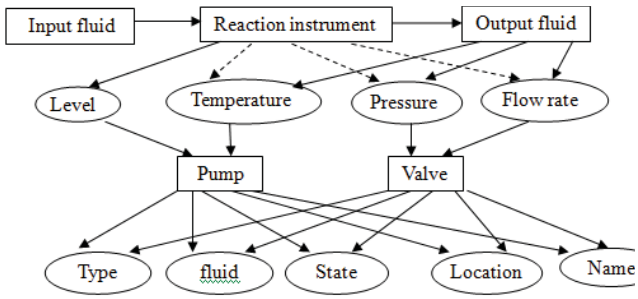


Figure 1. Relationship among the classes of the CO2 capture process system

Dynamic Knowledge

The dynamic knowledge represents the tasks of monitoring each reaction instrument, which includes the subtasks of monitoring its related attributes, or process parameters. During knowledge acquisition, twenty five critical process parameters, their desirable operating ranges, and the control actions when an abnormal condition emerges are specified.

The domain knowledge was analyzed based on the IMT, which emphasizes decomposition of the information into knowledge elements. The knowledge elements were then represented in the table format, which became the knowledge table. Table 1 provides a segment of the table that shows the knowledge element of inlet gas scrubber.

The table consists of 8 columns, the first four columns describe the static information, namely, the reaction instrument to which the attribute belongs, the attributes' number, tag, and name respectively. The fifth and sixth columns together describe the attribute's range of normal operation. For example, for the parameter of inlet-gas scrubber water level control (LC-410), the high limit of its desirable operating range is 65%, and the low limit is 60%. The "low low" limit indicates the critical condition which causes safety concerns. If the actual value of the attribute exceeds the limit value, the corresponding control action is executed. The seventh and eighth columns present the control action and diagnosis or explanation for that control

action, respectively. For inlet-gas scrubber water level control (LC-410), the valve ED-420 should be open when its value exceeds the high limit of 65% to increase water drainage of the tank; the valve EV-300 should be open if its value drops below the low limit of 60% to increase make-up water supply to the scrubber. However, if the remedial action is not taken in time and the attribute falls to the critical level of 30%, the pumps of P-420 and B-200 should both be shut off immediately to stop the wash water circulation and cut off the gas supply to the absorber.

TABLE I. SAMPLE SEGMENT OF KNOWLEDGE TABLE

Object Name	Attribute Number	Attribute Tag	Attribute Name	Operate Limit	Limit Value	Controlling Decision	Diagnosis & Explanation
C3. Inlet-Gas Scrubber	C3A1	DPT-200	Flue Gas Scrubber Differential Pressure				
	C3A2	LC-410	Inlet-Gas Scrubber Water Level Control	High	65.0	Open up EV-420	To increase wat drainage
				Low	60.0	Open up EV-300	To increase mal up water supply scrubber
				Low Low	30.0	Shut off P-420 & B-200	To stop the was water circulatin scrubber and st pumping flue g

IV. MAPPING KNOWLEDGE TABLE TO DELTAV STRUCTURE

The knowledge represented in the knowledge table is mapped into knowledge representation constructs in DeltaV Simulate. The DeltaV Simulate has a five-level hierarchical structure, which includes from top to bottom: (1) plant area, (2) module, (3) algorithm, (4) function block, and (5) parameter. The mapping from the knowledge table to the DeltaV system is presented as follows.

The plant areas are logical divisions of the process control system, which can be based on physical plant locations or main process functions. A plant area consists of modules, and each module is a logic control entity responsible for configuring the control strategies. It contains algorithms, alarms, and other characteristics that define the process control. Algorithms define the logic steps that describe how the module behaves and how the tasks are accomplished. In this intelligent system, the function block diagrams (FBD) were used to continuously execute control strategies. The basic component of a FBD is a function block, which contains the control algorithm and defines the behaviour of the module. Each function block contains parameters which are the user-defined data manipulated by the module's algorithm in its calculations and logic. The structure of DeltaV Simulate is shown in Figure 2, and the details of each hierarchical component are explained from the top to bottom level as follows.

Level 1: as the plant area is the logical division of the process based on the location or the function, each reaction instrument is defined as a plant area for two reasons. Firstly, every reaction instrument has its particular function. Secondly, the process parameters and the control devices are classified by which reaction instrument they belong to, i.e. the location. Therefore, defining 16 reaction instruments into 16 plant areas properly divides the entire plant into 16 functional and structural areas. It also can be observed from Figure 1 that the reaction instruments, the process parameters and the control devices are interrelated in a top-down hierarchy. Therefore, defining the reaction instruments as the top level logically supports the construction of the lower hierarchical components in DeltaV Simulate.

Level 2: modules are at the next level in the structural hierarchy of DeltaV Simulate, and a module is a logic control entity responsible for configuring the control strategies. These control strategies indicate the manipulation of the control devices, i.e., pumps and valves, to adjust the process parameters. The attribute of type of the control devices reflects that primarily two mechanisms of PID control and 2-state control are used to manipulate pumps and valves. Corresponding to these two types of control strategies, there are two types of modules: PID control modules and 2-state modules. The PID control modules configure the control strategies for the PID valves; the 2-state modules configure the control strategies for the control devices with two states, i.e., the pumps and solenoid valves.

Level 3: function block diagram (FBD) is a diagram that contains multiple interconnected function blocks. It generally contains input signal function block, control function block, and output signal function block. Different control function blocks are used and the choice depends on what control mechanism the module deploys. The control function block is the item that contains the most critical parameters.

Level 4: a function block defines the behaviour of an algorithm for a particular module. Two types of function blocks of 2-state function block and PID control function block are used for the control function block. Since the PID control valves are used to control the attributes of the reaction instruments, the attributes of reaction instruments and their relative PID control valves are combined into the PID control function blocks, which enable the present values of the attributes to approach their desired values by controlling the PID valves. The pumps and solenoid valves manipulate the attributes of the reaction instruments by switching between the ON/OFF states, i.e. between the current state and desired state. Therefore, the pumps and solenoid valves are represented by two-state control function blocks.

Level 5: parameters define the values of the attributes of the reaction instruments. As mentioned earlier, a PID function block controls a PID valve, so as to enable the present value of an attribute of the reaction instrument to approach its desired values. Therefore, the critical parameters of a PID function block include the present values of the attribute, the high/low values that specify the operating limits beyond which an alarm is activated, and the set-point value which represents the desired value. For the 2-state function blocks, the critical parameters are the current and desired states of the 2-state control devices.

The knowledge table developed based on knowledge about the CO_2 capture system clarifies the domain knowledge and supports encoding the knowledge into the hierarchical structure of DeltaV Simulate using a top-down approach. The knowledge table also supports a more efficient design of the expert system implemented on the DeltaV Simulate; the developed system involves a mapping of the knowledge table constructs to the DeltaV Simulate mechanisms. The reaction instrument of inlet gas scrubber and its related PID valve PCV-901 is used as an example to explain how each hierarchical

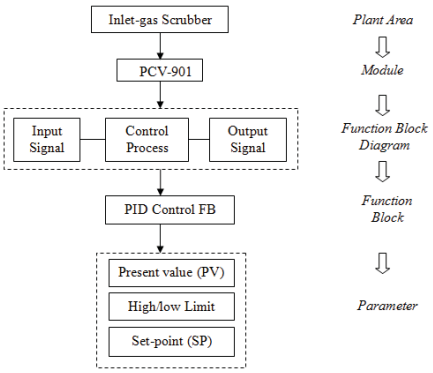


Figure 2. Hierarchical structure of sample component in DeltaV Simulate

level is developed on DeltaV Simulate based on information in the knowledge table. The components are shown in Figure 2 and the details are described below.

The reaction instrument of inlet gas scrubber is defined as a plant area. Its related PID valve PCV-901 together with the attribute it controls, i.e., the pressure of off gas (PT-901), are combined into a PID control module. The module is represented in a function block diagram, which consists of function blocks for input signals, output signals, and most importantly, for the control strategy. As shown in Figure 2, the function block for the control strategy conducts PID-control. The key parameters contained in this PID function block include: the present value of the off gas pressure, the set-point value of the off gas pressure, and the high and low limit values of the off gas pressure. If a value is beyond the range defined by the high and low limit values, the alarm function embedded in the PID function block is activated. A sample scenario is present in the next section.

V. Expert System on DeltaV Simulate

A sample interface of the expert system developed on DeltaV Simulate describes the scenario in which the inlet-gas scrubber wash water flow rate (FT-420) is in an abnormal condition. The interface is shown in Figure 3. The current value of FT-420 shown on the panel of FC-420 is 0kg/min, which is lower than its normal low limit of 5kg/min and indicates the water circulation between the inlet gas scrubber and water tank has stopped because the pump is shut. The alarm is activated and displayed on the interface, as a result, the panel for FC-420 turns to the blue color. The diagnosis and control suggestions are sent to a message board on the user interface: "If P-420 is on, open up FCV-420 to increase water returning from the scrubber to water tank; if P-420 is off, turn on P-420." The red color of pump P-420 indicates its closed status. Therefore, according to the control suggestion, P-420 should be opened so the water circulation will restart.

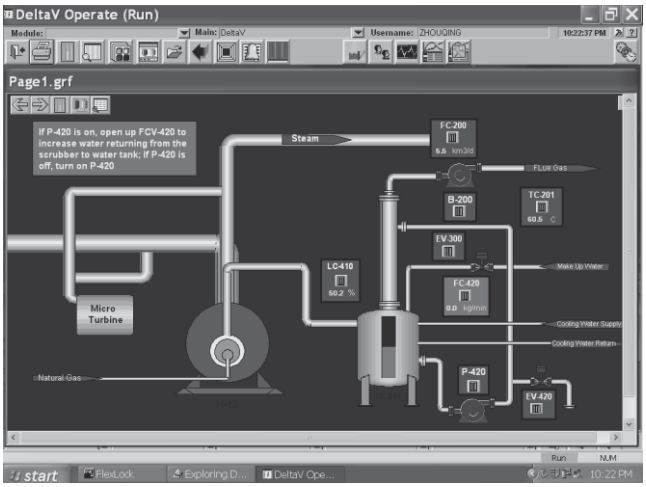


Figure 3. Interface of Expert System

VI. From Knowledge Tables to Implemented Ontologies

In addition to being used as the basis for development of the expert system, the knowledge tables were also used as the basis for building an implemented ontology of the domain. An implemented ontology can serve as the shareable and reusable knowledge base for supporting development of diverse KBS's. The implementation was conducted using both Protégé and Dyna [2]. Protégé is a freely available open source ontology editor and knowledge base framework implemented in Java. It supports constructing OWL ontologies as well as creating add-ons for other ontology or knowledge based applications. Dyna [2], is one such add-on. It is for the specification of dynamic knowledge. Being an add-on to Protégé allows Dyna to access the OWL ontology directly in the Dyna environment. This means that a dynamic action (or task) specified in Dyna can be linked to the OWL class in Protégé.

The static knowledge in the knowledge table was implemented using the OWLClasses of Protégé, and the dynamic knowledge was implemented with a Protégé add-on called Dyna. The conversion of knowledge elements specified in the knowledge table to the implemented ontology is explained with the knowledge element of CO_2 wash scrubber shown in the knowledge table segment in Figure 4 and discussed as follows.

TABLE II. Knowledge Table: Example of O15_CO2_Wash_Scrubber

Object Name	Attribute Number	Attribute Tag	Attribute Name	Operate Limit	Limit Value	Controlling Decision	Diagnosis & Explana
C15.	C15A1	DPT-700	CO_2 Wash Scrubber Differential Pressure				
CO_2 Wash Scrubber	C15A2	LC-710	CO_2 Wash Scrubber Level Control	High	95.0	Open EV-720	To increase water drain and lower the scrubber water level
				Low	5.0	Open EV-303	To increase make-up w supply to scrubber
				Low Low	0.0	Shut off P-720	To stop the wash water circulation in the scrubl

The class of CO_2 wash scrubber has the two attributes of the CO_2 wash scrubber differential pressure, and CO_2 wash scrubber level control. In the table shown in table 2, the class of CO_2 wash scrubber is referred to as "Object C15". Implicitly, there were two kinds of relationships among the classes. Firstly, an inheritance relationship exists among parent and child classes, e.g. the relationship between SC_3_Post-Conditioning_Unit_for_Product_Purification_Parts and O15_ CO_2_Wash_Scrubber is an inheritance relationship. The two classes are specified in Protégé and shown in Figure 4.

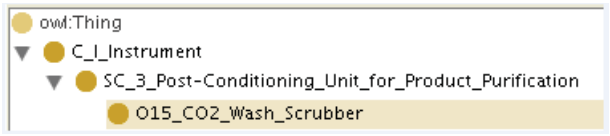


Figure 4. O15 Class Hierarchy Specified in Protégé

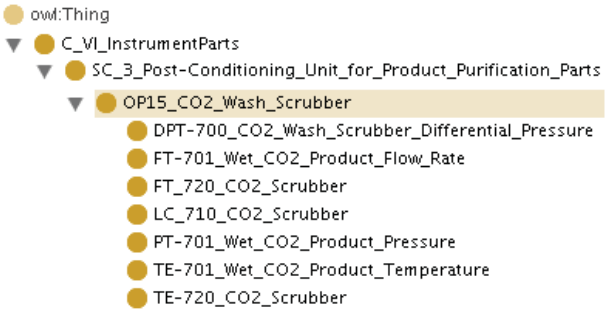


Figure 5. OP15 Class Specified in Protégé

The second kind of relationship is an aggregate relationship in which one class is a composite of several component classes, e.g. the relationship between O15_ CO_2_Wash_Scrubber and its component objects of FT_720_CO_2_scrubber and LC_710_ CO_2_scrubber, which are shown in Figure 5. Protégé converts the parent-child relationship in the first relationship into the OWL statements about the class of CO_2 wash scrubber, its parent class, and the other classes which are disjoint from it. These OWL statements are shown in Figure 6.

```
<owl:Class rdf:about="#O15_CO2_Wash_Scrubber">
  <rdfs:subClassOf rdf:resource="#SC_3_Post-
Conditioning_Unit_for_Product_Purification"/>
  <owl:disjointWith rdf:resource="#O16_CO2_Dryer_Unit"/>
  <owl:disjointWith rdf:resource="#O14_Reflux_Condenser"/>
  <owl:disjointWith rdf:resource="#O13_Reflux_Accumulator"/>
 </owl:Class>
```

Figure 6. OWL Statements on the Class of CO2 Wash Scrubber

The aggregate relationship among the classes shown in Figure 6 is converted into the OWL statements about the class of CO_2 product flow rate; it is a subclass of CO_2 wash scrubber, and the classes from which it is disjoint. The OWL statements are shown in Figure 7.

```
<owl:Class rdf:about="#FT-701_Wet_CO2_Product_Flow_Rate">
  <rdfs:subClassOf rdf:resource="#OP15_CO2_Wash_Scrubber"/>
  <owl:disjointWith rdf:resource="#DPT-
700_CO2_Wash_Scrubber_Differential_Pressure"/>
  <owl:disjointWith rdf:resource="#LC_710_CO2_Scrubber"/>
  <owl:disjointWith rdf:resource="#FT_720_CO2_Scrubber"/>
  <owl:disjointWith rdf:resource="#TE-720_CO2_Scrubber"/>
  <owl:disjointWith rdf:resource="#PT-701_Wet_CO2_Product_Pressure"/>
  <owl:disjointWith rdf:resource="#TE-701_Wet_CO2_Product_Temperature"/>
 </owl:Class>
```

Figure 7. OWL Statements on the FT-701 Wet CO2 Product Flow Rate and its disjoint classes

```
<Task><Name>C15A2_LC_710</Name>
<Documentation>CO2 wash scrubber level control</Documentation>
<SubTaskList/><DependencyList/>
<TaskArgList><TaskArg><VarType>float</VarType>
<VarName>co_level</VarName></TaskArg></TaskArgList>
<TaskReturn/>
<Behaviour>if(co_level &gt; 95.0){
        print "open EV_720"
        EV_720.state = "open"
}else if(co_level &lt;= 0.0){
        print "shut off P_720"
        P_720.power = "off"
}else if(co_level &lt; 5.0){
        print "open EV_303"
        EV_303.state = "open"
}</Behaviour><ObjectList>
<Object rdf:resource="http://www.owl-
ontologies.com/Ontology1276705464.owl#LC_710"/>
```

Figure 8. XML file dyna produced describing C15's behaviour

According to the IMT, the dynamic knowledge is decomposed into the knowledge elements of objectives and tasks, which are specified in the knowledge table. An example of this representation is given in section B *Dynamic Knowledge* Dyna performs two functions in converting the knowledge table into an implemented ontology. First, it converts the information into OWL statements, and secondly, it tests the specified behavior to ensure that the representation is correct. Based on the knowledge specified in the columns of Operate Limit, Limit value, Controlling Decision, and Diagnosis and Explanation of the knowledge table shown in Table 2, Dyna converts this segment of dynamic knowledge related to the attribute of level control of the class of CO_2 wash scrubber level control (C15A2_LC-710) into the OWL statements shown in Figure 8.

The testing function of Dyna requires the user to make up a test case for each behaviour that is specified in the knowledge table. The test case for the same knowledge element involves specifying three different values of 9.5, 5.0, and 0.0 for the attribute which would trigger the corresponding "print" statements. The test case is shown in Figure 9.

```
</ObjectList><PreCondition/><TestSuite><TestSetup/>
<TestCaseList><TestCase><TestCaseName>test</TestCaseName>
<TestCaseCode>C15A2_LC_710(96.0)
assert(EV_720.state == "open")
C15A2_LC_710(4.0)
assert(EV_303.state == "open")
C15A2_LC_710(0.0)
assert(P_720.power == "off")</TestCaseCode>
</TestCase></TestCaseList></TestSuite></Task>
```

Figure 9. Same XML file as above describing the testing of C15 behaviour

Decomposition of domain knowledge into its discrete elements facilitates knowledge sharing and reuse because an ontology that has a particular class, e.g., a pump class would enable it to link with other ontologies that also have that particular class of a pump. While the current version of the implemented ontology does not link to any other ontologies, it is possible that such a link can be established. For example, Ontocape [5] is a large ontology on the chemical process system domain that includes useful classes of knowledge on substance and measurement. It is an item on the future research agenda to see if some components of knowledge clarified and represented in Ontocape can be incorporated into the implemented ontology developed.

VII. Conclusion and Future Work

The objective of the work is to clarify and represent the domain knowledge of the CO_2 capture process system as the basis for building the ontology of this chemical process system. Knowledge engineering for the CO_2 capture process system based on the IMT resulted in documentation of the elicited domain expertise in a knowledge table. The domain knowledge was decomposed and clearly specified in the knowledge table. The explication was helpful because it clarified for the knowledge engineer the roles and relationships among the roles of the different knowledge elements. An expert system developed based on the represented knowledge can serve as a decision-support tool for the novice operator. When any fault occurs in the CO_2 capture process system, the time for diagnosis can be greatly shortened and the effective remedial control action can be deployed in a timely manner. This would help enhance efficiency the CO_2 capture process system. Also, the clarified knowledge about the constructive and reactive components facilitates understanding of the process system and construction of the expert system for automatic monitoring, control and diagnosis of the CO_2 capture process system. As well, it supports conversion of the knowledge table to an implemented ontology, which can be reused or extended in the future for building other knowledge based systems. Developing the ontology contributes to realization of the semantic web, which aims to structure data found on the internet by means of ontologies. Therefore, semantic web technology such as ontologies would enhance the ability of computers to search for information on the internet and support widespread use of agent technology for performing complex online tasks.

Acknowledgment

We are grateful for the generous support of Natural Science and Engineering Research Council (NSERC) and the Canada Research Chair Program.

References

[1] C.W. Chan, “From Knowledge Modeling to Ontology Construction,” International Journal of Software Engineering and Knowledge Engineering, vol. 14 (6), pp. 603–624, 2004.

[2] R. Harrison and C.W. Chan, “Tools for Industrial Knowledge Modeling and Management,” M.A.Sc. Thesis, U of Regina, Regina, pp. 1-175, 2007.

[3] T. Berners-Lee, J. Hendler, and O. Lassila, “The Semantic Web,” Scientific America, pp. 1-36, 2001.

[4] L.L. Chen and C.W. Chan, “Ontology Construction from Knowledge Acquisition,” Proceedings of Pacific Knowledge Acquisition Workshop, pp. 1-16, 2000.

[5] J. Morbach, A. Wiesner, W. Marquardt, “OntoCAPE—A (re) usable ontology for computer-aided process engineering,” Computers & Chemical Engineering, pp. 1546-1556, 2009.

[6] Q. Zhou, C.W. Chan, P. Tontiwachiwuthikul, R. Idem, D. Gelowitz, “A Statistical Analysis of the Carbon Dioxide Capture Process,” International Journal of Greenhouse Gas Control 3, pp. 535-544, 2009.

Maintainability Predictors for Relational Database-Driven Software Applications: Results from a Survey

Mehwish Riaz, Emilia Mendes, Ewan Tempero
Department of Computer Science, The University of Auckland
Auckland, New Zealand
mria007@aucklanduni.ac.nz, emilia@cs.auckland.ac.nz, e.tempero@cs.auckland.ac.nz

***Abstract*—Software maintainability is a very important quality attribute. Its prediction for relational database-driven software applications can help organizations improve the maintainability of these applications. The research presented herein adopts a survey-based approach where a survey was conducted with 40 software professionals aimed at identifying and ranking the important maintainability predictors for relational database-driven software applications. The survey results were analyzed using frequency analysis, and results suggest that maintainability prediction for relational database-driven applications is not the same as that of traditional software applications. The results also provide a baseline for creating maintainability prediction models for relational database-driven software applications.**

Software maintainability, relational database-driven software applications; survey; predictors; frequency analysis

I. INTRODUCTION

Software maintainability is the ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment [6]. It is inherently associated with the process of software maintenance which has long been known to have the major amount of software costs associated to it [12]. In order to manage these costs, it is important to understand, predict and improve software maintainability [12]. Software maintainability prediction involves proposing and validating predictors that have a bearing on software maintainability [1] and then employing these predictors to create software maintainability prediction models [13]. With the help of a maintainability prediction model, software organizations can better manage their maintenance resources and adopt a defensive design [11].

Database-driven applications have gained much popularity in modern software development [4] with relational databases as the most used and most successful type of database [15]. Relational databases are different from and more formalized than other types of databases and persistence mechanisms [2]. Database-driven applications consist of a database, a Database Management System (DBMS), and a set of applications that interact with the database through this management system [7]. Relational database-driven applications are therefore, those database-driven applications that have a relational database backend. The change in requirements cause these applications to undergo maintenance resulting in storing increased number of data sources and relationships; increased database complexity; and increased coupling between the database and application [9]. This suggests that the maintainability of these applications is also impacted by database specific factors in addition to application specific factors [13].

Given the importance of relational database-driven software applications in modern software development, it is important to investigate the factors that impact upon their maintainability in order to enable maintainability prediction. This paper is a step towards predicting maintainability of relational database-driven application by providing a validated list of predictors that have an impact on the maintainability of relational database-driven applications. This list of predictors was initially gathered with the help of twelve interviews conducted with software professionals [13]. The interviews' analyses resulted in the identification of 120 predictors. The focus of the work presented herein is to rank these predictors, with respect to the strength of their impact on the maintainability of relational database-driven applications, via another survey conducted with a larger sample of software professionals. The main contributions of this paper are therefore to:

- Further validate the predictors of maintainability identified by Riaz et al. [13] within the context of relational database-driven software applications.
- Rank the maintainability predictors for relational database-driven software applications in terms of their relative importance to predicting maintainability.
- Establish whether the predictors presented by Riaz et al. [13] positively or negatively impact the maintainability of relational database-driven applications.
- Present evidence on the type of applications and DBMS used the most in practice.

The remainder of the paper is organized as follows. Section II gives an account of the related work. Section III details the research methodology. Section IV presents the results. A discussion on the results and threats to validity are given in Section V, followed by conclusions in Section VI.

II. RELATED WORK

The research presented herein is informed by the results of: (1) a Systematic Review (SR) conducted on the topic of 'Software maintainability prediction and metrics' [12]; and (2) twelve interviews conducted with software professionals [13]

aimed at gathering evidence relating to the maintainability prediction of relational database-driven applications.

The results of the SR revealed very little evidence on maintainability prediction. The total number of studies selected in the SR was 15. These studies were further analyzed to assess if the datasets used in these studies completely or partially comprised relational database-driven software applications. Only three of these studies [3] [5] [8] had used relational database-driven software applications for the purpose of their research evaluation. However, only one of these studies [5] presented a maintainability prediction model but did not provide its prediction accuracy; the other two studies only presented measures [3] and factors [8] that impacted upon software maintainability, without any associated maintainability prediction model. Further analysis of these studies revealed that although they used relational database-driven applications, none of their proposed predictors or factors related specifically to a back-end database or to the interaction between a back-end database and the front-end application.

To further investigate maintainability prediction in the context of relational database-driven applications, interviews with software professionals were conducted to gather evidence from practice [13]. The results from the interviews revealed that a formal prediction model, or approach to predicting the maintainability of relational database-driven applications is not used in practice. The practitioners rely on expert judgement when assessing maintainability of relational database-driven applications. The analysis of these interviews also resulted in a list of factors that impact upon the maintainability of relational database-driven applications and may be used as their maintainability predictors.

III. RESEARCH METHODOLOGY

A. Survey-based Questionnaire

In order to determine which maintainability predictors are the most relevant and stand out the most from the list of predictors compiled with the help of SR and interviews' results, a survey was conducted with software practitioners. The list used in the survey comprised a total of 120 predictors [14], and the survey was informed by the results from the SR and the conducted interviews. It aimed at ranking these predictors in terms of their relative impact on the maintainability of relational database-driven software applications.

The survey questionnaire had 3 parts: The first part related to the respondents' characteristics, such as job function at the company, highest qualification, and total experience in years; and companies' characteristics, such as the year it was established, whether it is involved in developing relational database-driven applications, and the DBMS used most often.

The second part related to maintainability predictors. There were a total of 120 predictors, which belonged to one of the 10 categories below:

1. Database design
2. Relational database-driven applications
3. Design aspects of relational database-driven applications
4. Size, scope, and complexity of relational database-driven applications
5. Quality, quality assurance, and testing of relational database-driven applications
6. Change and maintenance of relational database-driven applications
7. Development and deployment environment of relational database-driven applications
8. Web applications with relational database back-end
9. Organizational culture and policies
10. Team communication and project management.

The possible responses against each of the factors, in relation to their impact on the maintainability of relational database-driven applications, were recorded on a 7-point bi-polar scale. The responses could be one of 'high decrease', 'medium decrease', 'low decrease', 'no impact', 'low increase', 'medium increase', and 'high increase'. The scale was designed such that both the direction of the factors' relationship (positive or negative i.e., whether a given factors increase maintainability or decreases maintainability, respectively) and the strength of the relationship with maintainability could be determined.

The third part of the questionnaire recorded opinions on which aspect of a relational database-driven application, from interface, data model, or algorithm objects/computation intensive modules, was hardest to maintain; asked of any additional predictors of maintainability; and recorded information on how a judgment is made on the maintainability of relational database-driven applications in a company (for which the option supplied was the amount of time to carry out a maintenance task). Refer to Riaz [14] for the complete survey questionnaire.

The survey was tested with a pilot run of four practitioners. Based on the feedback from the pilot, the survey was improved where the predictors from the SR, especially those that corresponded to various software metrics, were described using a simpler language in order to be clearly understandable by the practitioners.

B. Survey process

In order to conduct the survey, approval from the University of Auckland Human Participants Ethics Committee was obtained (Ref. 2009/527). The process prescribed by UAHPEC was to contact the competent authorities within the companies which would then extend the invitation to their employees. The invitations were sent to more than 50 companies in New Zealand and more than 25 companies in Pakistan through email. The email addresses were obtained through companies' listings on the Internet. In addition, a mailing list of the University of Auckland's Center of Software Innovation and authors' own contacts were also used to make contact with the companies. The email invitation explained the purpose and nature of the survey, provided the Web link to the survey, and asked the companies' competent authorities to request participation by those employees within their companies that had experience with developing and/or maintaining relational database-driven software applications.

The survey and the relevant UAHPEC documents were made available online and were filled out via a Web interface [14]. Some respondents also downloaded the questionnaire and returned an e-copy. The survey was conducted between June 2010 and September 2010.

C. Data Source

The data source used herein comprised responses of a total of 40 software professionals - 13 from 8 different software companies in New Zealand and 27 from 15 different software companies in Pakistan, where one respondent was self-employed. The roles of the participants varied from 'Software Developer' to 'Head of Development'. The participants had at least 2 years of experience with software development and/or maintenance, and had an average experience of 7.8 years with a minimum of 2 and maximum of 28 years of experience.

The two countries, Pakistan and New Zealand, were chosen for the following reasons: i) the interviews of which this research is informed were also conducted in New Zealand and Pakistan; ii) the results of the interviews established that the context of this research was not culture-sensitive as the focus was on software applications in general and not specific aspects related to people or their way of carrying out their work; and iii) the first author is originally from Pakistan and research is being carried out at the University of Auckland, New Zealand to which all the three authors are affiliated.

In terms of the participating companies, only 3 companies, from which only one respondent each participated, were created 1, 2 and 5 years ago. All the other companies were at least 10 years old. In addition, all companies except two were engaged in development or maintenance of applications where 90% or more of their applications had relational database backend. The other two also were involved in developing or maintaining relational database-driven applications; however, most of their work did not involve this type of development. In relation to the use of DBMS, all the participating companies used either Oracle or Microsoft SQL Server.

D. Data Analysis

Prior to analyzing the data, it was consolidated from two different sources, a database residing on Microsoft SQL Server used for the online survey and e-copies of the survey forms filled by some respondents for which the data was manually entered. Frequency analysis was the technique used for data analysis. It was well suited to the purpose as the intention was to see how many respondents thought of a predictor as having an impact 1 to 7 (high decrease to high increase) on the maintainability of relational database-driven software application.

During the frequency analysis, initially only the predictors that were reported to have an impact of 'high decrease' or 'high increase' on the maintainability of relational database-driven applications were considered. However, the frequency values did not stand out when only the extreme ends of the bi-polar scale were considered. For instance, for most of the factors the proportion of the respondents (out of 40) that chose one of the extreme ends of the scale (1 and 2 OR 6 and 7) were half or less than half i.e., between 15 and 20 (out of 40 respondents) and the proportion of the respondents that chose other options 'medium decrease' to 'medium increase' was equal or higher to those that selected one of the values at the extreme ends of the scale. Therefore, for final analysis, the sum of the frequencies for 'high decrease' and 'medium decrease' (1 and 2 on the bi-polar scale), and 'medium increase' and 'high increase' (6 and 7 on the bi-polar scale) were considered. The sum of the frequencies for the two ends of the bi-polar scale presented a better choice and represented better insights into the data.

IV. Results

A. Maintainability Predictors

The results of the survey data analysis corresponding to the second part of the survey are presented in Figure 1, and show only those maintainability predictors for relational database-driven applications that had frequency values of 21 or higher i.e., those predictors that were reported by more than 50% of the respondents as having a high impact on the maintainability of the mentioned applications. Note that, the predictors presented in the table are all those predictors that had an impact of 'medium increase' or 'high increase' on the maintainability of relational database-driven applications. None of the predictors that had an impact of 'medium decrease' or 'high decrease' on maintainability were selected as a result of the frequency analysis. This shows that all the predictors listed in Figure 1 have a positive relationship with the maintainability of relational database-driven software applications i.e., they all result in an increase in the maintainability of relational database-driven applications whenever they also increase.

The number of selected predictors for each category (as defined in the survey questionnaire and mentioned in Section III-A) against the total in that category are given in Table I. Note that predictors belonging to categories 'Size, Scope, and Complexity of Relational Database-Driven Software Applications' and 'Web Applications with Relational Database Backend' were not selected during frequency analysis and are therefore not presented in the results discussed further.

It is also worth noting that the best predictor relates to the relational database schema. In addition, other predictors, such as good database design, optimal use of DBMS features, are also among the top predictors of maintainability for relational database-driven software applications. Note that these predictors were not reported in the studies selected in the SR yet these rank higher than the predictors that were identified both by SR and interviews.

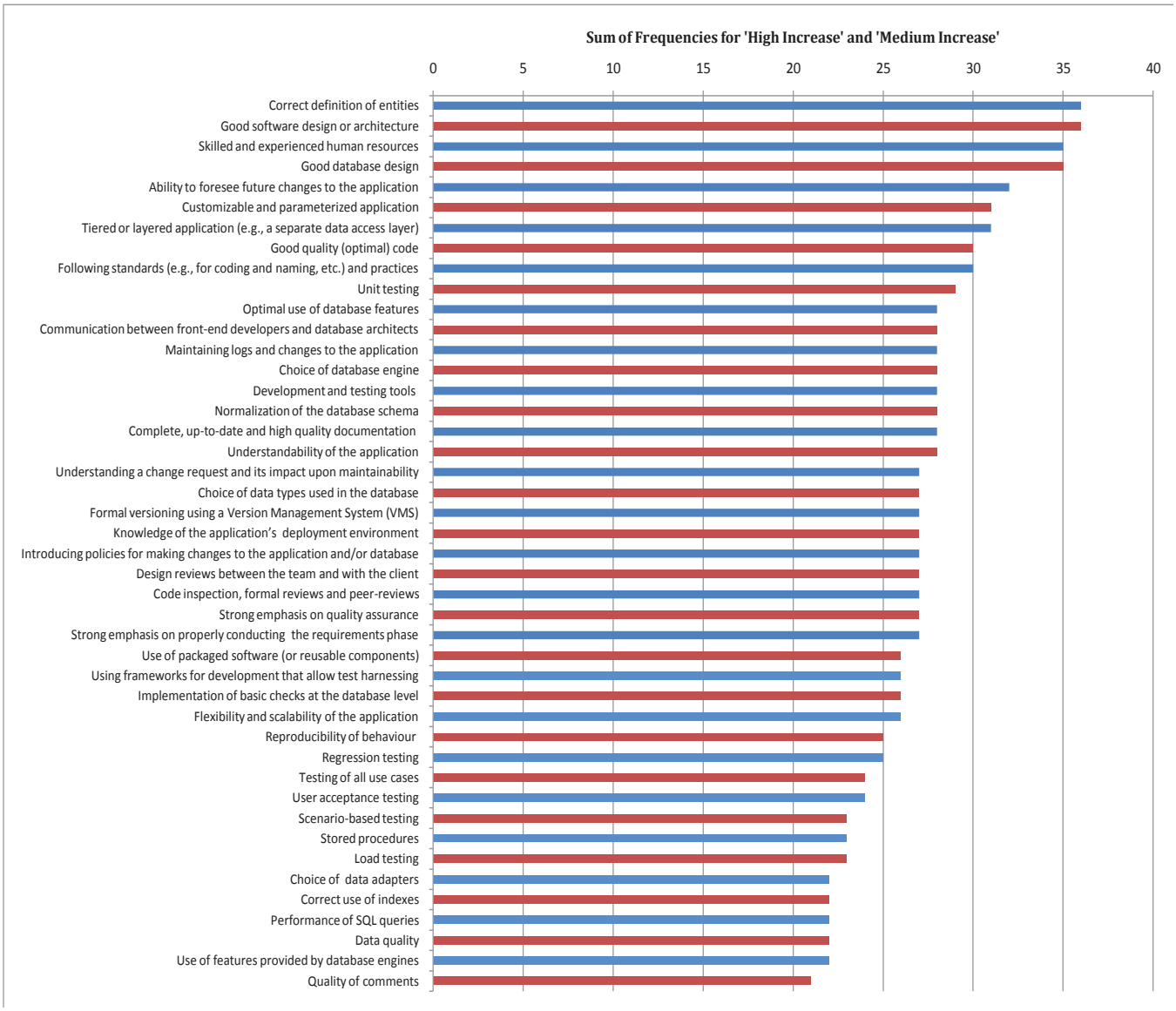


Figure 1. Predictors of maintainability for relational database-driven software applications

The percentages of predictors selected for each category presented in Table I.

TABLE I. PERCENTAGE OF MAINTAINABILITY PREDICTORS PER CATEGORY

Category	Frequency	Total in Category	Selection Percentage
Database Schema	4	19	21%
Relational Database-Driven Applications	7	15	47%
Design Aspects of Database-Driven Applications	4	9	44%
Quality, QA, and Testing	17	20	85%
Change and Maintenance	4	10	40%
Development and Deployment Environment	4	7	57%
Organizational Culture and Policies	2	4	50%
Team, Team Communication, and Project Management	2	7	29%

The results given in Table I show that the most important category of predictors was 'Quality, Quality Assurance, and Testing', both in terms of total number of predictors selected for overall analysis and number of predictors selected from the total number of predictors within the category. The second important category, perhaps not in terms of the percentage of factors selected from the category, but the number of factors contributing to maintainability prediction is 'Relational Database-Driven Applications'. Other important categories relate to 'Development and Deployment Environment' and 'Organizational Culture and Policies'. The results clearly indicate that the factors related to quality, database schema, relational database-driven application play an important role in predicting maintainability. However, the results also indicate that the factors related to development environment and organizational culture are also considered important by

practitioners in order to determine the maintainability of relational database-driven applications.

B. Ranking of the Aspect of Relational Database-Driven Applications for Ease of Maintenance

The first section of the third part of the survey asked the respondents to assign a ranking of 1(easiest) to 3(hardest) to each of three listed aspects of the relational database-driven applications i.e., application's interface, application's data model, and application's algorithmic objects or computation intensive modules. The results (see Figure 3) show that the application's interface was considered the easiest to maintain by 33 respondents, followed by the application's data model (24 respondents) and algorithmic objects (23 respondents). As such, there was not too much difference between the number of respondents for second and third rankings, which suggest that the data model of a relational database-driven software application can be as difficult to maintain as algorithmic or computation intensive objects.

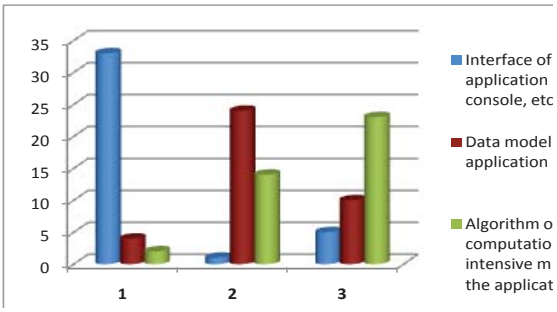


Figure 2. Ranking of the aspects of relational database-driven applications as per ease of maintenance

In addition to the ranking against the application's aspects, this part of the survey also asked if the respondents wanted to report any other factors they believed were important for predicting maintainability of relational database-driven applications. Only three respondents added their responses, which included software development process level used in an organization; non-technical aspects related to customer behavior which drives the timelines for performing maintenance tasks; and conformance of DBMS to standards, reliability of WANs, and impact of other software installed on the system such as anti-virus, operating system etc. As such, these factors were not considered significant as part of the analysis as each of these were reported by only one respondent and would have ultimately been eliminated from the list of top maintainability predictors.

C. Measures of Maintainability for Relational Database-Driven Software Applications

The second section of the third part of the survey was aimed at recording the most used measure of maintainability among time to carry out a maintenance task, own experience and judgment, and an equation or a more sophisticated method. Thirteen respondents provided no answer to this question. Among the 27 responses received, 20 favored expert judgment, 4 favored time to carry out a maintenance task, 1 favored a more sophisticated method, and two reported a combination of time to carry out a maintenance task and their judgment. The one respondent who mentioned an equation or more sophisticated method also added a note to indicate that there is no use of an equation or model but the company uses mature practices so it has its own parameters to assess maintainability.

Overall, their own experience and judgment was the measure reported by most surveyed practitioners to measure the maintainability of relational database-driven applications. This suggests that maintainability of relational database-driven applications is determined by subjective assessment in practice.

V. Discussion

This paper presents the results of a survey conducted with software professionals. The survey was informed by a previously conducted SR [12] and the results of interviews conducted with software professionals [13]. It was aimed at identifying maintainability predictors that had the highest impact on the maintainability of relational database-driven software applications. A list of 120 predictors belonging to 10 different categories was used on which expert opinions from software professionals were recorded. The survey results suggest that the top 5 predictors related to 'Database Design', 'Design Aspects of Relational Database-Driven Applications', 'Team, Team Communication and Project Management', and 'Change and Maintenance'. The number of predictors selected by more than 50% respondents was highest for the category 'Quality, Quality Assurance, and Testing' followed by the predictors related to 'Relational Database-Driven Application'.

The main contribution of this paper is that it identifies the top predictors of maintainability for relational database-driven applications. Other contributions include establishing and supporting by evidence that the type of applications most developed in software organizations have a relational database backend, as discussed in Section III-C; interface of a relational database-driven application is considered easiest to change by software practitioners whereas data model and computation intensive modules are considered equally difficult to maintain in comparison to interface; and maintainability in practice is measured using expert judgment.

The results of the survey presented some interesting findings in terms of the factors considered important by software practitioners. From the results of the SR, the most frequently mentioned predictors related to coupling, complexity and size related measures. However, the survey results suggested that the factors related to 'Size, Scope, and Complexity of Relational Database-Driven Applications' were not ranked higher than predictors belonging to other categories by the surveyed software practitioners. In fact, the list of predictors ranked higher by more than 50% of the respondents does not include any factor from this category. This is a very important finding suggesting a huge gap between research and practice with regards to the perception of maintainability predictors in the context of relational database-driven applications. The possible reasons for this can be that most research experiments reported in literature are not performed in an industrial setting and the applications used in research experiments are generally small in scope. This makes it easier to gather metrics and experiment on these applications whereas

in practice there is lesser room for experimentation due to tight deadlines. In addition, in practice no sophisticated techniques for maintainability prediction are used and therefore, practitioners rely more on the visible factors on which that they can more easily formulate a judgment on maintainability. Also, the sample size for this research is not large so there is also the possibility of lack of external validity. We believe that irrespective of the reasons, there is a need to further investigate maintainability predictors of for relational database-driven applications. In this regard, our work in progress involves gathering project specific data on the predictors identified herein and creating maintainability prediction models based on the gathered data. This, in addition to providing prediction models, will also be a means to validating the predictors and will suggest a need to consider predictors other than the ones selected herein.

The maintainability predictors for software applications suggested in the literature involved measures related to coupling, complexity and size of the application. The results of the survey suggest that size related measures are not important predictors whereas predictors related to database design are very important for maintainability prediction of relational database-driven applications. This suggests that the maintainability of relational database-driven applications is different from that of the software applications.

It may appear that the data analysis method used is not very sophisticated and rigorous. However, the purpose of this survey was to identify top predictors from the list of 120 predictors such that these could be used for future work involved in this research. The analysis technique used perfectly suited the purpose and the use for a more sophisticated technique was hence, not required. Moreover, there are examples in the literature on software prediction where frequency analysis has been used to rank the predictors to be used further for the creation of prediction models [10].

The possible limitations of this research are related to unequal number of respondents from the two countries and smaller sample size. While equal number of respondents from both countries would have provided better opportunity for data analysis and comparison of results, the research itself was context free as established by the results from the interviews [13]. Therefore, we believe that the unequal number of respondent in this case is not a threat to the validity of the research. In regards to small sample size, we believe that it poses threats to the external validity of the research as the chances of obtaining different results from a larger sample size cannot be ignored. However, considering the number of companies to which invitations to participate were emailed, the length of the survey, and the time frame within which the survey had to be completed; we believe that the sample data is of a considerable size to derive valuable conclusions.

VI. CONCLUSIONS

This paper presents the results of a survey conducted with software professionals. The survey was informed by the results of a previously conducted SR and interviews conducted with software professionals. The aim of the survey was to rank the 120 maintainability predictors of relational database-driven applications belonging to 10 different categories for their strength in predicting maintainability. Out of 120 total factors, 44 factors were found to have an impact of either 'high increase' or 'medium increase' on maintainability, as per more than 50% of the respondents. The most important category of predictors was 'Quality, Quality Assurance' and Testing' whereas the top predictor ‘Correct definition of entities’ belonged to the category 'Database Design'. Our results also suggest that the practitioners measure maintainability using expert judgment and consider computation intensive modules of relational database-driven applications as easy or difficult to maintain as the data models.

Future work involves creating maintainability prediction models for relational database-driven software applications. In this regard, the work on gathering project related data on the predictors is already in progress.

REFERENCES

[1] P. Bhatt, et al., “Influencing Factors in Outsourced Software Maintenance”, ACM SIGSOFT Soft. Eng. Notes, 31, 3, pp. 1 – 6, 2006.

[2] A. Cleve, T. Mens, and J. Hainaut, “Data-Intensive System Evolution”, Computer 43, 8 (August 2010), pp. 110-112.

[3] E.H. Ferneley, “Design Metrics as an Aid to Software Maintenance: An Empirical Study”, J Softw Maint: Res. Pract., 11, pp. 55-72, 1999.

[4] S.K. Gardiokiotis, “A Structural Approach towards the Maintenance of Database Applications”. IDEAS ’04, 277-282, 2004.

[5] M. Genero, J. Olivas, M. Piattini, F. Romero, “Using metrics to predict OO information systems maintainability”, ICAISE, pp. 388-401, 2001.

[6] IEEE Std. 610.12-1990, “Standard Glossary of Software Engineering Terminology”, IEEE Computer Society Press, Los Alamitos, CA, 1993.

[7] G.M. Kapfhammer and M.L. Soffa, “A Family of Test Adequacy Criteria for DB-Driven Applications”, ESEC/FSE 2003, pp.98-107.

[8] J.S. Lim, S.R. Jeong, S.R. Schach, “An empirical investigation of the impact of the object-oriented paradigm on the maintainability of real-world mission-critical software”, J Syst Software, 77, pp.131-138, 2005.

[9] A. Maule, W. Emmerich, and D.S. Rosenblum, “Impact Analysis of Database Schema Changes”, ICSE ’08, pp. 451-460, 2008.

[10] E. Mendes, N. Mosley, and S. Counsell, “Investigating Web size metrics for early Web cost estimation”, J. Syst. Softw. 77, 2, pp. 157-172, 2005.

[11] P. Oman, and J. Hagemeister, “Construction and Testing of Polynomials Predicting Software Maintainability”. J Syst Software, 24, 1994.

[12] M. Riaz, E. Mendes, and E. Tempero, “A Systematic Review of Software Maintainability Prediction and Metrics”, ESEM 2009, pp. 367-377, 2009.

[13] M. Riaz, E. Mendes, and E. Tempero, “Towards Maintainability Prediction for Relational Database-Driven Software Applications – Evidence from Software Practitioners”, ASEA 2010, pp. 110-119, 2010.

[14] M. Riaz, “Survey of software maintainability prediction for relational database-driven software applications”, http://www.cs.auckland.ac.nz/survey_2010.

[15] M. Seltzer, “Beyond Relational Databases”, Communications of the Acm, (July 2008), 51, 7, pp. 52-58, 2008.

How Annotations are Used in Java: An Empirical Study

Henrique Rocha*, Marco Tulio Valente*

*Department of Computer Science
Federal University of Minas Gerais (UFMG)
Email: henrique.rocha@gmail.com, mtov@dcc.ufmg.br

***Abstract*—Since 2004, Java provides support to general purpose annotations (also known as metadata) that allows developers to define their your own annotation types. However, seven years after their inception in the Java language, we still do not have empirical evidence on how software developers are effectively using annotations in their systems. Therefore, this paper presents an empirical study on how annotations are used on a corpus of 106 open-source Java systems. On total, we have evaluated more than 160,000 annotations that have been applied to the source code of such systems. Our main findings can be summarized as follows: (a) the so-called annotation-hell phenomena affects many of the evaluated systems; (b) developers are using both pre-defined annotations and annotations defined by external frameworks, mostly annotations dedicated to persistence and testing; (c) most of the evaluated annotations have been employed to annotate methods (more than 90%); (d) although Java does provide not support to annotations for anonymous classes, several programs from our corpus have applied annotations to such classes.**

***Index Terms*—Annotations, Empirical Studies, Qualitas Corpus.**

I. Introduction

Annotations have been one of the features introduced in J2SE 5.0 released in 2004 [3], [1]. Annotations are a metadata feature that provides the ability to associate additional information to Java elements (like classes and methods). Annotations have no direct effect on the code they annotate, and they do not change the way in which the source code is compiled. There are at least the following possible uses for annotations [3], [1], [6]:

- Compiler information (using compiler annotations) to suppress warnings and to detect errors. For example, developers can use the `@SuppressWarning("unchecked")` annotation to suppress the unchecked warning that can be raised when using an old code written before J2SE 5.0.
- Generation of additional files such as source code, XML, help files, and so on. For example, developers can annotate some system's functions with `@Help("...")` and later an annotation processing tool can read this metadata and generate a help file.
- Source code documentation by javadoc and similar tools (using the `java.lang.annotation.Documented` annotation).
- Since annotations can be processed at runtime, they can be used for logging, testing and other utilities.
- Some APIs (like Java Persistence API) relies on annotations as a declarative way to provide their services. Instead of calling methods or declaring objects, an annotation is placed in the program elements that the developers want to be annotated. For example, to define a class as a `JPA` entity, we just need to put the `@Entity` annotation on the class.

Although annotations have several useful applications, their use could massively contribute to code pollution. This phenomenon is usually called *annotation hell*, i.e. the source code becomes cluttered with so many annotations leading to poor code readability and understandability.

The ultimate goal of our research is to empirically investigate the use of annotations in Java programs. Annotations have been a part of the Java language for almost seven years, and now it is reasonable to assume that they are being used in most Java applications. We are particular interested in providing answers to the following questions: (a) Which are the most used annotations? (b) Which program elements (i.e. methods, classes) are annotated more frequently? (c) Are there many systems suffering from the *annotation hell* phenomenon?

In order to answer such questions, we present the results of an in-depth empirical study including the inspection of annotations in 106 open-source Java systems that are a part of the Qualitas Corpus. The Qualitas Corpus is a collection of open-source Java sytems designed to help code analysis in empirical studies [9], [8]. Basically, we developed two approaches to extract the annotation data from the Qualitas Corpus. The results from both approaches are shown and compared, providing a better analysis to the use of annotations and even showing some tendencies that programmers are following.

The remainder of this paper is organized as follows. In Section II, we provide an introduction to the use of annotations in Java, explaining their most recomended uses and underlying syntax. Section III describes the methodology used for the empirical study, including the two approaches we have followed to retrieve annotations in Java systems. Section IV analyzes and discuss the obtained results. In Section V we present this study's threats to validity. Section VI discusses related work and Section VII concludes the paper, summarizing its main

contributions and outlining future work.

II. Annotations

Annotations are a metadata feature that does not directly affect the code they annotate. As mentioned in the Introduction, they can be used for several purposes like runtime testing, generating additional files, providing information to compilers, and providing a simple and declarative form to use API features [3], [1], [6].

In terms of implementation, annotations can be considered a special kind of interface [3], [1]. To distinguish an annotation declaration from an interface, annotations are declared using the `@interface` keyword. Listing 1 shows a code fragment that declares an annotation called `Author`. Annotations can make use of access modifiers just like interfaces (`public`). In the example, the `Author` annotation has the *element* `name` which has a `String` type and by default has the `"[unassigned]"` value. Annotations *elements* are a special kind of method declaration.

```
1 public @interface Author {
2     String name() default "[unassigned]";
3 }
```

Listing 1. Code fragment declaring an Annotation

Listing 2 shows how the `Author` annotation can be used to annotate a class. This code also shows some `JPA` annotations. For example, the `@Entity` denotes that the `Customer` class will be made persistent by `JPA`. The `@Id` annotation is used to define the entity's primary key.

```
1 @Author{name="Henrique"} @Entity
2 class Customer implements Serializable{
3
4     @Id
5     public Integer getCustomerId(){
6       ...
7     }
8 ...
```

Listing 2. Annotations Example

Annotations can be divided based on their retention policy into three types: Source, Class and Runtime annotations [3]. Source annotations are processed during compilation (usually by a compiler plug-in), but they have no effect on the generated code. Class annotations are also processed during compilation and they are stored in the generated class files. Runtime annotations are stored in class files and can be recovered at runtime through the Java Reflection API.

There are some predefined annotations in Java, including the following compiler annotations: `@Deprecated`, `@Override`, and `@SuppressWarning`. The `@Deprecated` annotation indicates that the marked component is deprecated and should no longer be used. The `@Override` is an annotation for methods only, and it indicates that the method is supposed to override a superclass method. Finally, the `@SuppressWarning` annotation disables a specific warning the compiler would otherwise generate.

The additional data provided by annotations can be processed by the compiler, other tools or, in the case of runtime annotations, they can be examined at runtime using reflection. The most common tool to work with annotations is the annotation processing tool (`apt`) provided in the Java development toolkit. The `apt` retrieves annotations in source files, using custom factories and visitors so that programmers may process annotations according to their needs.

III. Methodology

In this section we describe the target systems used in the study (Section III.A) and the methodoly we followed in the research (Section III.B).

A. Target Systems

We defined the scope of our study to be the systems contained in the Qualitas Corpus [9]. The Qualitas Corpus is a *curated* collection of open-source Java systems whose ultimate goal is to reduce the effort in finding, obtaining, and organizing code sets to be used in empirical studies [9]. We used the Qualitas Corpus version `20101126r`, that contains only the most recent versions of the 106 systems.

B. Study Design

To retrieve the annotations used by the systems in the Qualitas Corpus, we implemented a factory class for the `apt` tool, that is a part of the Java SE Development Kit (JDK) version 1.6.0 update 24. The `apt` parses the source files and provides to the factory class each annotation found, as well as details about the annotation. Basically, our factory class saves all the relevant information about the annotations in a relational database for latter analysis.

Since `apt` is the official Java tool to process annotations, our original plan was to use it to examine all annotations used by the systems included in the Qualitas Corpus. However, the `apt` tool has stopped with runtime errors while trying to process 15 systems – Table I shows the name of these systems. The errors occurred on the tool itself, and even bug report information has been raised to contact the manufacturer.

TABLE I
Qualitas Corpus systems the APT has not been able to Process

System	System	System
Castor	javacc	nakedobjects
Cayenne	jboss	netbeans
gt2	jrefactory	springframework
Hibernate	maven	struts
jFin_DateMath	myfaces	tapestry

Due to the `apt` errors, we created a textual search program to find annotations in source files. Originally, this textual search approach has been used on the systems that the `apt` tool has not been able to process. Latter we expanded the textual search to analyze all the systems to better compare the differences between both approaches.

TABLE II
ANNOTATIONS FOUND ON THE QUALITAS CORPUS

System	KLOC	APT		Textual Search	
		Total	Density	Total	Density
ant	107.770	4	0.04	4	0.04
antlr	25.243	984	38.98	995	39.42
aoi	111.725	21	0.19	21	0.19
argouml	194.859	2038	10.46	2047	10.51
aspectj	412.394	889	2.16	893	2.17
azureus	453.433	39	0.09	40	0.09
castor	115.543	–	–	929	8.04
cayenne	127.529	–	–	3473	27.23
checkstyle	23.316	1413	60.60	1529	65.58
cobertura	51.860	12	0.23	12	0.23
compiere	400.257	3621	9.05	3752	9.37
derby	592.817	22	0.04	22	0.04
drjava	62.380	123	1.97	177	2.84
eclipse_SDK	2282.511	798	0.35	916	0.40
findbugs	109.096	2122	19.45	2387	21.88
fitlib	27.539	625	22.70	658	23.89
freecol	81.671	205	2.51	278	3.40
freecs	23.012	5	0.22	6	0.26
gt2	446.863	–	–	7001	15.67
heritrix	61.681	150	2.43	177	2.87
hibernate	163.858	–	–	6755	41.22
hsqldb	123.268	22	0.18	22	0.18
htmlunit	40.004	5881	147.01	5959	148.96
informa	9.722	3	0.31	3	0.31
ireport	221.490	5454	24.62	5549	25.05
itext	76.369	371	4.86	371	4.86
jFinDateMath	4.807	–	–	64	13.31
jasperreports	170.064	99	0.58	106	0.62
javacc	13.772	–	–	8	0.58
jboss	281.643	–	–	2582	9.17
jchempaint	90.831	8923	98.24	8923	98.24
jedit	107.469	486	4.52	523	4.87
jena	70.948	1148	16.18	1474	20.78
jgrapht	11.931	53	4.44	53	4.44
jgroups	96.325	1436	14.91	1436	14.91
jhotdraw	75.958	3143	41.38	3570	47.00
jmeter	81.010	1164	14.37	1234	15.23
jre	914.867	1182	1.29	1197	1.31
jrefactory	113.427	–	–	44	0.39
jruby	160.360	5139	32.05	5217	32.53
jspwiki	43.326	158	3.65	160	3.69
jung	37.989	422	11.11	432	11.37
junit	6.164	171	27.74	213	34.56
lucene	113.223	106	0.94	112	0.99
marauroa	13.823	424	30.67	434	31.40
maven	54.336	–	–	880	16.20
megamek	258.957	1649	6.37	1839	7.10
myfaces_core	119.529	–	–	2844	23.79
nakedobjects	110.378	–	–	4578	41.48
netbeans	1890.536	–	–	57820	30.58
picocontainer	9.259	196	21.17	206	22.25
pmd	60.875	769	12.63	786	12.91
poi	143.507	286	1.99	294	2.05
proguard	55.567	45	0.81	45	0.81
quartz	26.819	32	1.19	33	1.23
roller	50.980	96	1.88	97	1.90
rssowl	73.230	1639	22.38	2434	33.24
spring	160.302	–	–	7883	49.18
squirrelsql	6.944	4	0.58	15	2.16
struts	74.670	–	–	1848	24.75
sunflow	21.648	17	0.79	25	1.15
tapestry	53.367	–	–	4263	79.88
tomcat	166.478	2717	16.32	2897	17.40
trove	2.196	3	1.37	4	1.82
weka	224.356	21	0.09	21	0.09
Total		56330		160570	

IV. RESULTS

In this section we present the results from our study. Table II shows the systems, their Kilo non-comment Lines of Code (KLOC), the total annotations found and the annotation density as retrieved by the `apt` tool and by the textual search. The annotation density is calculated by dividing the number of annotations by the KLOC. Therefore, this density represents an average of how many annotations are present in every one thousand lines of code. As mentioned, the `apt` tool crashed with an internal bug on 15 systems, these systems are shown on the table with a "–".

Two more aboservations should be made about the data in Table II, first the table shows that the size of the evaluated systems has no impact on the runtime errors raised by the `apt` tool. For example, the `castor` system has 115.543 KLOCs and caused a crash in the `apt`, while the `Eclipse_SDK` system has 2,282.490 KLOCs and has been sucessfully processed.

Second, Table II also shows a slight difference between the number of annotations found by the `apt` and by the textual search. Investigating further, we discovered that Java does not support annotations in the scope of anonymous classes. Because of that, the `apt` has completely ignored those annotations. On the other hand, the implemented textual search has count such annotations.

To better inspect the different measurements regarding annotations in anonymous classes, we will use the `jhotdraw` system as an example. As can be observed on Table III, the measurements have a difference of 427 annotations in their count. Table III also shows the particular annotations present in the `jhotdraw` system. As we can observe, the main difference is in the `@java.lang.Override` annotation, followed by the `@java.lang.SupressWarning` annotation. In fact, this seems to be case not only for the `jhotdraw`, but for every system in the Qualitas Corpus with a different annotation count between the `apt` and the textual search.

TABLE III
JHOTDRAW ANNOTATIONS

Annotation	APT	Textual Search
@java.lang.Override	2909	3323
@java.lang.SuppressWarnings	89	102
@org.jhotdraw.annotations.Nullable	95	95
@org.jhotdraw.annotations.NotNull	42	42
@java.lang.Deprecated	4	4
@java.lang.annotation.Documented	2	2
@java.lang.annotation.Target	2	2
Total	**3143**	**3570**

From all the 106 analyzed systems, 41 did not have a single annotation – Table IV shows these systems and the year of their release date. Analysing the years on this table, we can

conclude that seven systems were release before 2004 (i.e. before the introduction of annotations in Java) and another seven systems have been release in 2004.

TABLE IV
QUALITAS CORPUS SYSTEMS WITHOUT ANNOTATIONS

System	Year
axion	2003
c_jdbc	2005
colt	2004
columba	2005
displaytag	2008
drawswf	2004
emma	2005
exoportal	2006
fitjava	2004
galleon	2006
ganttprj	2009
ivatagrp	2005
jag	2006
james	2004

System	Year
jasml	2006
jext	2004
jfreechart	2009
jgraph	2009
jgraphpad	2006
jmoney	2003
joggplayer	2002
jparse	2004
jpf	2007
jrat	2003
jsXe	2006
jtopen	2010
log4j	2010
mvnforum	2010

System	Year
nekohtml	2010
openjms	2007
oscache	2007
pooka	2008
quickserver	2006
quilt	2003
sablecc	2005
sandmark	2004
velocity	2010
webmail	2002
xalan	2007
xerces	2010
xmojo	2003

Finally, Table V shows the top ten Qualitas Corpus systems with annotations using both measurements. Based on this table it is possible to conclude that some of the systems the `apt` was not able to process have a large number of annotations, for example, `netbeans`, `spring framework` and `hibernate`.

TABLE V
TOP TEN QUALITAS CORPUS SYSTEMS WITH THE HIGEST NUMBER OF ANNOTATIONS

APT		
System	Total	Density
jchempaint	8923	98.24
htmlunit	5881	147.01
ireport	5454	24.62
jruby	5139	32.05
compiere	3621	9.05
jhotdraw	3143	41.38
tomcat	2717	16.32
findbugs	2122	19.45
argouml	2038	10.46
megamek	1649	6.37

Textual Search		
System	Total	Density
netbeans	57820	30.58
jchempaint	8923	98.24
spring	7883	49.18
gt2	7001	15.67
hibernate	6755	41.22
htmlunit	5959	148.96
ireport	5549	25.05
jruby	5217	32.53
nakedobjects	4578	41.48
tapestry	4263	79.88

In the remainder of this subsection we first discuss the results using the `apt` tool (Section IV.A). Second, we show the results provided by the textual search approach (Section IV.B). And finally, we discuss both results in order to clarify our main findings (Section IV.C).

A. APT

Table VI shows the top ten analyzed systems with the highest annotation density. For example, the `htmlunit` system – the system that presented the highest annotation density – has a 147.01 density, which means that on average on each one thousand lines of code there will be 147 annotations.

Table VII show the total number of annotations from all systems processed by the `apt` classified by program element. In this classification, a type denotes packages, classes, enums or interfaces. As we can see, more than 92% of the annotations are used to annotate methods.

TABLE VI
TOP TEN QUALITAS CORPUS SYSTEMS WITH HIGHEST ANNOTATION DENSITY AS PROCESSED BY THE APT

System	KLOC	Annotations	
		Total	Density
htmlunit	40004	5881	147.01
jchempaint	90831	8923	98.24
checkstyle	23316	1413	60.60
jhotdraw	75958	3143	41.38
antlr	25243	984	38.98
jruby	160360	5139	32.05
marauroa	13823	424	30.67
junit	6164	171	27.74
ireport	221490	5454	24.62
fitlib	27539	625	22.70

TABLE VII
ANNOTATIONS BY PROGRAM ELEMENT

Element	Annotations	Percentage
Type	2459	4.36
Constructor	612	1.09
Field	865	1.54
Method	51931	92.19
Parameter	463	0.82
Total	**56330**	**100**

Finally, Table VIII presents the most used annotations in the processed systems. As expected, the three compiler annotations are used very frequently in such systems.

TABLE VIII
MOST USED ANNOTATIONS IN THE QUALITAS CORPUS SYSTEMS PROCESSED BY APT

Annotation	Number	Percentage
@java.lang.Override	28723	50.99
@Test	11327	20.10
@org.jruby.anno.JRubyMethod	2762	4.90
@org.openscience.cdk.annotations.TestMethod	2735	4.85
@java.lang.SuppressWarnings	2699	4.79
@java.lang.Deprecated	1783	3.16
@com.grlsoft.htmlunit.BrowserRunner.Alerts	1523	2.70
@org.openscience.cdk.annotations.TestClass	443	0.78
@RunWith	359	0.63
@org.jgroups.annotations.Property	290	0.51
@BeforeClass	279	0.49

B. Textual Search

Table IX shows the ten analyzed systems with the highest annotation density, as processed by our text-based search approach. By comparing Tables VI and IX, it is possible to conclude that some systems the `apt` has not been able to process have a high density value and they appeared on the textual search (`tapestry`, `spring framework`, `nakeobjects`, `hibernate`). The system that presented the highest and second highest densities were respectivelly the `htmlunit` and `jchempaint`.

Due to limitations inherent to the textual search approach, we could not determinate to which program element the annotations belong to. Finally, Table X presents the most used annotations in all systems. We can observe that, as in the `apt` tool based results, the compiler annotations are widely used.

TABLE IX
TOP TEN QUALITAS CORPUS SYSTEMS WITH THE HIGHEST ANNOTATION DENSITY AS PROCESSED BY THE TEXTUAL SEARCH

System	KLOC	Annotations	
		Total	Density
htmlunit	40.004	5959	148.96
jchempaint	90.831	8923	98.24
tapestry	53.367	4263	79.88
checkstyle	23.316	1529	65.58
springframework	160.302	7883	49.18
jhotdraw	75.958	3570	47.00
nakedobjects	110.378	4578	41.48
hibernate	163.858	6755	41.22
antlr	25.243	995	39.42
junit	6.164	213	34.56

TABLE X
TEXTUAL SEARCH ANNOTATIONS MOST USED IN THE CORPUS' SYSTEMS

Annotation	Number	Percentage
@Override	92275	57.46
@Test	20875	13.00
@SuppressWarnings	7714	4.80
@Deprecated	3377	2.10
@Column	2836	1.76
@JRubyMethod	2755	1.71
@TestMethod	2735	1.70
@Alerts	1509	0.93
@Entity	1121	0.69
@Id	951	0.59

C. Analysis

Based on the results presented on this paper we can present some interesting findings about the use of annotations.

First, by inspecting Tables VI and IX, we can observe that the annotation density values for these systems (specially those on Table IX) are a strong indication they might be suffering from *annotation hell*. For example, when we analyze the `htmlunit` system, it scored the highest density in both approaches, having almost 150 annotations per KLOC (or approximatly one annotation at 7 lines of code).

A second interesting finding is about the annotations used on the systems. We initially expected that all three compiler annotations to be amongst the most used ones. But the second most used annotation was the `@Test` annotation. The `@Test` annotation is used to test methods dynamically through reflection. Apparently, this has become a popular standard used by several programmers.

Moreover, we can see on Table X that the annotations `@Column`, `@Entity` and `@Id` are used very frequently. These annotations are used by the Java Persistence API to map Java classes to relational database tables.

Another annotation used frequently, as presented on Tables VIII and X, is the `@org.jruby.anno.JRubyMethod`. This annotation is used by the `jruby` system to specify methods signatures using Java code and to use them as method calls in Ruby code. To illustrate the use of `@JRubyMethod`, Listing 3 shows an example of a `readchar` method marked with this annotation. As mentioned, the goal is to mark this method callable from code within Ruby.

```
1 @JRubyMethod(name = "readchar")
2 public static IRubyObject readchar(
      ThreadContext context, IRubyObject recv)
      {
3 ...
```

Listing 3. JRubyMethod Annotation Example

A third interesting finding comes from the observation of Table VII. As we can see, most annotations were used to annotate methods (over 90%), which is a huge difference to the second most used annotation, the types annotations with almost 5%. Very few annotations have been used to mark constructors, fields and parameters.

Finally, another interesting finding was that although Java does not support annotations for anonymous classes, many programs are using it. The reason why the annotation count differ from the `apt` and the textual search is mainly because of those anonymous class annotations. Analyzing the table we can see that in 38 out of 50 systems evaluated by both approches we have anonymous class annotations.

Investigating this occurence further, we discovered another reason for such form of use. Some programming IDEs warn the programmer to use annotations or even generates code with annotations, regardless they are located in anonymous class. We tested this feature using the `Eclipse` IDE build ID 20110218-0911 to generate the code for an anonymous inner class for a button event, and the IDE generated the method with an `@Override` annotation.

V. THREATS TO VALIDITY

Two main factors can impact on the validity of the results presented in this paper. First, our chosen systems might not be representative of all annotations used by the entire Java community. To minimize this threat we used the Qualitas Corpus that has 106 open-source systems from which 65 had annotations.

The second threat is related to the `apt` bugs. We were not able to determine what caused the bug that made the `apt` to crash while processing 15 systems from the Qualitas Corpus. To minimize this threat we developed a second approach based on a textual search. In this way the results obtained by the `apt` could be compared and validated to the results obtained by the textual search. However, using the textual search approach it was not possible to extract the annotations' full qualified name, nor which Java program element have been annotated.

VI. RELATED WORK

Annotations can be considered a relative new feature in the Java language. Probably because of that, there is not many works related to studying the use of annotations in existing systems. However, there are studies related to creating tools to verify the correct use of frameworks and APIs annotations. One of those efforts presents the AnnaBot tool [1], that demonstrate its usefulness using Java Persistence API examples. Another proposed tool to verify frameworks and

APIs annotations is the ModelAn [6], which is a model based approach, and uses the Fraclet as a case study.

On the field of empirical studies about Java constructions and abstractions, there are many related work. First, there is an empirical study of UpgradeJ [10] which is a variant of the Java language. UpgradeJ is a language that allows multiple versions of classes to co-exist, thus supporting dynamic software updatings. The empirical study has evaluated different versions of classes on open-source Java applications and estimated how many of the classes changes could be handle by UpgradeJ dynamically. The results show that most changes could be supported without a significant code rewrite.

There is also a work aiming to evaluate the use of non-private fields in Java applications since there is little empirical evidence to this practice [8]. This study has relied on 100 open-source Java applications (that we could consider to be an ancestor to the Qualitas Corpus [9]). The study has concluded that it is not uncommon for systems to declare non-private fields and do not take advantage of that access.

Another empirical study that uses a Qualitas Corpus release aims to find how programmers are overriding methods [7]. This work relies on a set of metrics to analyze the systems and how well their overriding implementation is. This study found that most subclasses override at least one method. It also found some questionable uses of overriding in the Qualitas Corpus systems.

The analysis of class coupling and the choice of refactoring (or removing) them in latter releases of the systems is the topic of another empirical study [5]. This study shows that the size of the source code does not affect the redesign choices, and a strong tendency that some classes have towards their removal.

Annotations have also been investigated in the scope of aspect-oriented software development. For example, annotations are often mentioned as an alternative to design more robust pointcuts. Kiczales and Mezini recommend using annotations when: (i) it is difficult to write a stable regular expression or enumeration-based pointcut; (ii) the name of the annotation is unlikely to change; (iii) the annotation denotes a well-defined semantic property (and not properties that are only true in some configurations of the system) [4]. Eaddy and Aho propose using annotations at the statement level for exposing join points needed by heterogeneous concerns and for enabling fine-grained advising [2]. However, their proposal can lead to a widespread use of annotation and thus can increase the code scattering and tangling phenomenon usually associated to Java annotations.

VII. Conclusions

We have presented a large empirical study on the use of annotations in open-source Java systems. Our motivation for this work was mainly because as a new language feature, it is difficult to find studies analyzing the use annotations. After analyzing the systems using the official Java annotation processing tool, we had to develop a textual search program to process the systems where the `apt` hast failed. Using the `apt` we were able to process 50 systems with 56,330 annotations; and using the textual search we processed 65 systems with 160,570 annotations.

We have also shown that some systems have a very high annotation density. Moreover, some of the discoveries found by our analysis from both approaches confirm that the most used annotations are the compiler ones. Also that methods are the most annotate element, and that the `JPA` annotations are popular. We also found that programmers are using the `@Test` as a standard annotation to mark methods that will be dynamically tested by reflection.

Finally, although the Java language does not support annotations inside anonymous classes, programmers are still annotating them. Some programming IDEs generate code with annotations even in anonymous classes. This indicates a tendency that maybe should be included on latter versions of the Java language.

As future work we hope to inspect even further the annotations on Java applications, processing and analyzing more annotated data.

References

[1] I. Darwin. Annabot: a static verifier for Java annotation usage. In *2nd International Workshop on Defects in Large Software Systems*, pages 21–28, 2009.

[2] M. Eaddy and A. V. Aho. Statement annotations for fine-grained advising. In *ECOOP Workshop on Reflection, AOP, and Meta-Data for Software Evolution*, pages 89–99, 2006.

[3] J. Gosling, B. Joy, G. Steele, and G. Bracha. *Java Language Specification, 3rd Edition*. Addison-Wesley, 2005.

[4] G. Kiczales and M. Mezini. Separation of concerns with procedures, annotations, advice and pointcuts. In *19th European Conference on Object-Oriented Programming (ECOOP)*, volume 3586 of *Lecture Notes in Computer Science*, pages 195–213. Springer, 2005.

[5] A. Mubarak, S. Counsell, and R. M. Hierons. An empirical study of "removed" classes in Java open-source systems. In *Advanced Techniques in Computing Sciences and Software Engineering*, pages 99–104. Springer, 2010.

[6] C. Noguera and L. Duchien. Annotation framework validation using domain models. In *Model Driven Architecture Foundations and Applications*, volume 5095 of *Lecture Notes in Computer Science*, pages 48–62. Springer, 2008.

[7] C. S. Tempero, E. and J. Noble. An empirical study of overriding in open source Java. In *33rd Australasian Computer Science Conference (ACSC)*, volume 102, pages 3–12, 2010.

[8] E. Tempero. How fields are used in Java: An empirical study. *Software Engineering Conference, Australian*, 0:91–100, 2009.

[9] E. Tempero, C. Anslow, J. Dietrich, T. Han, J. Li, M. Lumpe, H. Melton, and J. Noble. Qualitas corpus: A curated collection of Java code for empirical studies. In *Asia Pacific Software Engineering Conference (APSEC)*, pages 336–345, dec 2010.

[10] E. Tempero, G. Bierman, J. Noble, and M. Parkinson. From Java to UpgradeJ: an empirical study. In *1st International Workshop on Hot Topics in Software Upgrades*, pages 1–5, 2008.

Automated Extraction of Data Lifecycle Support from Database Applications

Kaiping Liu[1], Hee Beng Kuan Tan[1], Xu Chen[2], Hongyu Zhang[3] and Bindu Madhavi Padmanabhuni[1]

[1] School of Electrical and Electronic Engineering, Nanyang Technological University
{kpliu, ibktan}@ntu.edu.sg
padm0010@e.ntu.edu.sg

[2] Institute of Computing Technology, Chinese Academy of Sciences
Beijing, China, 100190
chenxu@software.ict.ac.cn

[3] School of Software, Tsinghua University
Beijing, China, 100084
hongyu@tsinghua.edu.cn

***Abstract*—Database application is one of the most common types of systems. Grounded on the simple concept of data lifecycle—any data in database is created from insertion, used via selection and modification and terminated at deletion—this paper proposes a novel approach to reverse engineer the data lifecycle automatically from the source code of database applications. The extracted information can be used for the selection of open-source database applications for adaptation. It can also be used for maintenance and verification of database applications. A tool has been developed to implement the proposed approach for PHP-based database applications. Case studies have also been conducted to evaluate the use of the proposed approach.**

Keywords-data lifecycle; extraction; reverse engineering; maintenance; verification

I. INTRODUCTION

Due to the complexity of software development and absent or incomplete associated documentation through most stages of the whole software development life cycle, recovery of system inner features automatically from program source code is important in system comprehension, verification and maintenance. Reverse engineering is the process of analyzing a subject system to create representations of the system at a higher level of abstraction [1], and is a suitable approach for this purpose.

During the past decades, much effort has been focused on this area. Michael L. Nelson conducted a survey on the reverse engineering and program comprehension in [2]. He presented various approaches to automating reverse engineering including syntactic analysis, graphing methods and execution and testing method. In 2000, H. A. Muller.et al [3] presented a roadmap of reverse engineering building on the program comprehension theories. They discussed the code reverse engineering and explored the spectrum of reverse engineering tools.

Database application is one of the most common types of systems, it is important to provide support to analyze the code of the database applications automatically. The specific characteristic of database applications is data manipulation and the data maintained in the database is always dynamic. The flow of data processing, from insertion to being used to finally deletion, indicates a lifecycle of data in the database, which can be a representative aspect of database applications.

Based on the basic concept of data lifecycle, this paper proposes a novel approach to reverse engineer the data lifecycle automatically from the source code of database applications. This basic but representative information is an easy to use and understand indicator that benefits the selection, maintenance and verification of database applications.

The paper is organized as follows. Section 2 depicts the data lifecycle. Section 3 discusses the proposed approach. Section 4 reports our evaluation. Section 5 presents the related work and Section 6 concludes the paper.

II. DATA LIFECYCLE

Data maintained in a database is usually dynamic. Once the data is created from insertion, the data should be used, updated, and finally removed once it becomes obsolete. This implies that an attribute and table defined in the schema have a lifecycle that starts from being inserted, via being used as and when needed, and optionally being updated in any number of times, to being deleted. We call such characteristics data lifecycle of attribute and table respectively. Fig. 1 depicts the data lifecycle using state transition diagram.

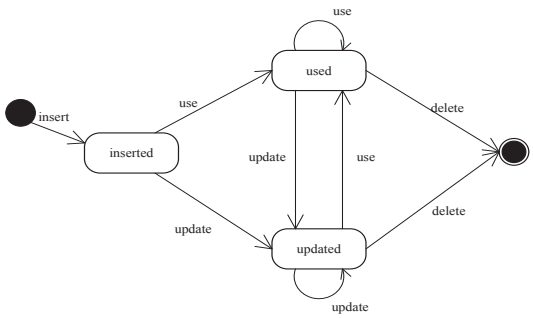


Figure 1. Data lifecycle of attribute and table

The property of the data lifecycle implies directly that any database application must include related operations to support the data lifecycle of attribute and table. Any missing of database operation for supporting the data lifecycle is very serious. Programs in database applications provide support to data lifecycle by performing four database operations: INSERT, SELECT, UPDATE and DELETE.

TABLE I. TYPES AND CLASSIFICATION OF DATA LIFECYCLE SUPPORT

SELECT	INSERT	UPDATE	DELETE	Category	IMPLICATION
T	F	T	T	Undefined	Data is updated, used and deleted without defined
T	F	T	F	Undefined	Data is updated and used without deleted and defined
T	F	F	T	Undefined	Data is used and deleted without updated and defined
F	F	T	T	Undefined	Data is updated and deleted without used and defined
T	F	F	F	Undefined	Data is used without updated, defined and deleted
F	F	T	F	Undefined	Data is updated without used, deleted and defined
F	F	F	T	Undefined	Data is deleted without updated, used and defined
F	T	T	T	Redundant	Data is defined, updated and deleted without used
F	T	F	T	Redundant	Data is defined and deleted without updated and used
F	T	T	F	Redundant	Data is defined and updated without used and deleted
F	T	F	F	Redundant	Data is defined without used updated and deleted
T	T	T	F	Undeletable	Data is defined, used and updated without deleted
T	T	F	F	Undeletable	Data is defined and used, without updated and deleted
T	T	F	T	Non-modifiable	Data is defined, used and updated without updated
F	F	F	F	Reserved	Data is specified without defined, used, updates and deleted
T	T	T	T	Complete	Data is specified then defined, used, updates and deleted

In a database application, most of the attributes and tables should be provided with a complete data lifecycle. That is, there are programs in the application which perform insertion, selection, updating and deletion for these attributes and tables. However, we should note that there are some special cases on updating and deleting the data, e.g., the application of initial public offer of a company stock. The record can't be updated or cancelled when it is confirmed.

Table I shows all the sixteen possible combinations of data lifecycle support provided in a database application with SELECT, INSERT, UPDATE and DELETE operations. We can classify the sixteen possible combinations according to their implications. For example, if insertion, selection, updating and deletion are all provided for an attribute, then the data lifecycle for that attribute is basically complete. If none of these database operations is provided, then basically the attribute or table is just to be reserved for future use. In addition to the details of data lifecycle support types, Table I also shows the classification of data lifecycle support provided in a database application.

III. EXTRACTION OF DATA LIFECYCLE SUPPORT

In this section, we give detailed description on the extraction of data lifecycle support from the source code of database applications.

Considering different purposes, our approach extracts data lifecycle support of the attribute and table respectively. The data lifecycle support of attribute provides detailed information about the lifecycle support of each attribute and its category. The data lifecycle support of table is the combination of data lifecycle of attributes within the table and provides detailed information about the lifecycle support of each table and its category. Next, we describe the extraction process.

In the extraction, for each SQL query, we need to identify: the operation type (SELECT, UPDATE, INSERT or DELETE); attribute names and table names that are involved. In practice, an SQL query is usually formed as a string literal or more often a string variable, and is then passed to some specific query functions, e.g. mysql_query in PHP if MySQL is used. Based on our observation, the value of the SQL string variable commonly varies according to some conditions, i.e. the actual value taken by the query function is subject to different program paths chosen during execution. Fig. 2 shows a simple snippet of PHP code as an example for this.

```
1. <?php
2. if (…)
3.     $attr = 'col1';
4. else
5.     $attr = 'col2, col3';
6. mysql_query('SELECT ' . $attr . ' FROM table');
7. ?>
```

Figure 2. A PHP code snippet

From Fig 2, it can be observed that the SQL query in line 6 is constructed by concatenation of three strings, the second of which is a variable with different execution values under different conditions. For the extraction, first, we compute the control flow graph and then perform control and data dependency analysis.

A. Construction of Control Flow Graph and Control Dependence Analysis

All SQL query function call sites in the source code should be identified, and parameters they take can then be evaluated. We compute standard control flow graph (CFG) for each program. Fig. 3 shows the CFG for the code in Fig. 2.

Starting from the entry block, the control flow graph is traversed and each basic block encountered is checked to see whether it contains query function calls. We can locate one such function call in the node 5 in Fig. 3 representing line 6 in the code in Fig. 2. The parameter which contains the SQL query is extracted. For each query, it would be either one single

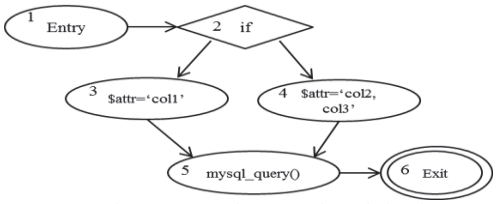


Figure 3. CFG for example code in Fig. 2

variable, or the concatenation of several string literals and variables. In Fig. 2, the parameter only includes one variable: $attr. The actual value of the variables vary according to different condition values taken in the predicate nodes before the function, whose different successive paths would assign different values to the variables that are used directly or indirectly by the query function. To include all possible values for the variables for the precision of the extraction process, we need to further adopt data dependence analysis.

B. Data Dependence Analysis

For each basic node in CFG that contains one or more SQL query function calls, we extract paths in the CFG starting from the entry node and passing through the function call node. For example, from the CFG in Fig. 3, we obtain two paths for node 5. One is 1 –> 2 – >3 –> 5 and the other is 1 –> 2 –> 4 –> 5.

For each such path, data dependence analysis is performed for every variable that is used in the function call. We trace each variable's direct or indirect definition from the function call node backward along the path, until either 1) a constant assignment is encountered, for example: $attr = 'col1'; 2) assignment with values that are unable to be determined, such as return value of user input functions. If the string value of one variable can be determined, this variable is replaced by its actual value; otherwise it is replaced by an empty string. All these strings are concatenated in the same order as they are listed in the function call. By this way, the string value of the parameter used in SQL query function is extracted.

In this way, for the above example, from the first path, we generate the SQL query: "SELECT col1 FROM table"; from the second path, we produce "SELECT col2, col3 FROM table". These queries are put into a set for further analysis.

C. Data Lifecycle Generation

After the possible values of the SQL string variable used in the query function are retrieved, the set of SQL query strings are formed. When we parse each SQL query, the operation type of the query, i.e., either SELECT, UPDATE, INSERT or DELETE, is first determined. Next, the attribute names and table names are obtained from the strings according to the standard SQL grammar specification. For some set of SQL queries, the operation type, attribute names and table names are identical, and the changing part is the values that are used by the attributes, for example, values in the conditions in WHERE clause. In our approach, these different SQL queries are treated as duplicated and we only keep one of them.

For the example in Fig. 2, we finally extract two selection operations, the first one selects attribute "col1" from table "table"; the second one selects attributes "col2" and "col3" from the same table. We count the two operations into the corresponding data lifecycle information for the attributes and the table.

Finally, with these retrieved attribute operations and their occurrence sites, we integrate them together to count operations on each attribute in each table. Based on these, data lifecycle for attributes and tables as described above can be generated accordingly. Fig. 4 shows the above mentioned data lifecycle extraction process.

```
Algorithm extractDataLifecycle:
Input: source code files of one software system
Output: Data lifecycle information
Begin:
for each source code file F do:
    Generate control flow graph G for F;
    Traverse G and identify query function invocations;
    for each invocation node I do:
        Extract parameter list from I;
        for each variable V do:
            for each path p in G that passes I do
                Perform data dependency analysis for V on P;
                Trace definitions backward along P;
                if the last traceable definition is string
                    literal then:
                    Replace V with the string value;
                else:
                    Replace V with empty string;
                endIf;
            endFor;
        endFor;
        Generate a SQL query by concatenation of string
            values of all variable;
        Parse the SQL query and record
            operation type, attribute names and table names;
    endFor;
endFor;
Integrate all extracted information for each attribute of each
table;
Generate data lifecycle;
End.
```
Figure 4. The pseudo-code of data lifecycle extraction

IV. EVALUATION

In this section, we evaluate our proposed approach for the selection, maintenance and verification of database applications with case studies. We start by first giving a description of our prototype tool.

A. Prototype Tool

Many database applications are developed using PHP. Hence, we implemented a prototype tool in Java for PHP applications in order to verify our proposed approach. PHP is the widely used scripting language for developing web application. We used Pixy [4], an open-source PHP analyzer to parse the PHP source code.

The tool analyzes the source code and extracts the data lifecycle support automatically. It works in the following steps: first it scans through the PHP system and finds all the PHP source code files; second, it parses each file to generate corresponding control flow graph, and then identifies the query

execution functions; third, the tool computes data dependency for the variables involved in the query functions, and based on that possible SQL queries are generated, the operation types and table and attribute names involved are determined; finally, according to the information extracted, the data lifecycle support is generated automatically. Though some variables' values cannot be decided during data dependency analysis, we observed that these variables are mostly used as the values of attributes, and in most cases the operation types, attribute and table names involved in the queries can be resolved from string literals.

Fig. 5 shows a diagram depicting the workflow of the prototype tool.

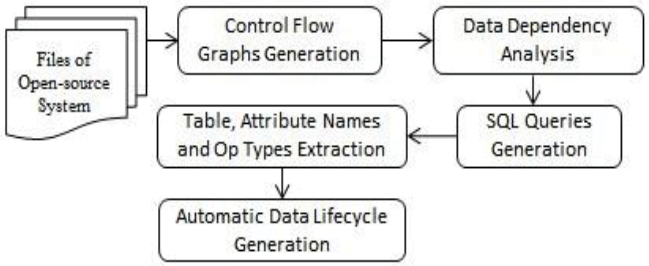


Figure 5. The workflow of the prototype tool

B. Case Studies

1) Selection

Due to the lack of documentation, selection of an appropriate open-source database application for use from the huge repository is not an easy task. Our approach extracts easy to use and understand lifecycle support information automatically to assist the users in the selection process.

To aid the selection process, we give summarized information of the system. With this information, one can assess the quality of the whole system directly from the analysis of data lifecycle completeness. Furthermore, with the information of data lifecycle of attributes and tables, the user can look into detailed information of implementation and be clear about the data lifecycle for each attribute and table.

For the selection process, we evaluated our proposed approach on four real-world PHP DB applications from sourceforge.net. They are School Mate (v1.5.4, a PHP/MySQL school administration and management), Open-School (pre-alpha release, a web-based School Management Software), LAMP School (v0.3 beta, an online school register system) and School Admin (v0.3, a web-based software for schools and colleges). The four systems provide similar functionalities, but differ in their completeness and maturity. Our tool is applied to analyze them and discover their data lifecycle support information.

Due to the space limit, summarized information of two systems is shown in Table II. From the experiment, we know that the percentage of complete category is highest for School Mate (45.3%) followed by LAMP School (42.0%) whereas School Admin has the lowest percentage (1.2%) of complete attributes. The percentage of undefined attribute of School Mate is the lowest (21.3%) while that of Open-School's is the highest (46.0%). Besides, the percentage of the reserved attributes of School Mate (4.3%) is lower than that of the other three (12.4%, 28.0% and 35.3% for Open-school, LAMP School and School Admin respectively). It also has no undeletable attribute in it. Both the Open School and Lamp School have no redundant attributes or tables while School Admin and School Mate have 8.2% and 6.8% redundant attributes respectively. Furthermore, only in School Admin, the percentage of undeletable category is 17.6% while there are no undeletable attributes and tables amongst the other three.

TABLE II. SUMMARIZED DATA LIFECYCLE SUPPORT

School Mate	**Table**		**Attribute**	
	#	**%**	**#**	**%**
Complete	11	73.3	53	45.3
Redundant	0	0	8	6.8
Non-modifiable	2	13.3	26	22.2
Undeletable	0	0	0	0
Undefined	2	13.3	25	21.3
Reserved	0	0	5	4.3
Total	15	100	117	100
School Admin	**Table**		**Attribute**	
	#	**%**	**#**	**%**
Complete	1	6.3	1	1.2
Redundant	3	18.8	7	8.2
Non-modifiable	1	6.3	6	7.1
Undeletable	3	18.8	15	17.6
Undefined	1	6.3	26	30.6
Reserved	7	43.8	30	35.3
Total	16	100	85	100

The above mentioned information suggests that each system has some advantages against others as well as some disadvantages. Usually a more complicated and better established system would have more complete data lifecycles. On the contrary, systems that have not been fully developed may have poor data lifecycles.

The empirical results can facilitate the user when selecting a suitable database application for adaptation. Considering the huge amount of open-source systems available on the Internet, the data lifecycle support is a good way to start and help the selection process.

2) Maintenance

The data lifecycle support for attributes and tables is also useful in database application maintenance since it reveals the completeness of functions from inside the source code. In this section, we demonstrate how the data lifecycle support can be used in the maintenance process for cases of undefined, non-modifiable, undeletable, redundant and reserved respectively. We use Table III and Table IV for demonstration.

- *Undefined*

If the data lifecycle support information provided for an attribute or table in a database application is classified as undefined, it is certainly not adequate and additional programs must be coded to address this flaw.

TABLE III. DATA LIFECYCLE OF ATTRIBUTE FOR SCHOOL ADMIN*

Attribute	Table	S	U	I	D	Category
admid	admission	T	F	T	T	Non-modifiable
cls	classes	T	F	T	F	Undeletable
reasonofrej	admission	F	T	T	T	Redundant
email	emp	F	T	F	F	Undefined
empid	emp	F	F	F	F	Reserved
statid	admission	T	T	T	T	Complete

* S = Select, U = Update, I = Insert, D = Delete. This is only a part.

TABLE IV. DATA LIFECYCLE OF TABLE FOR SCHOOL ADMIN*

Table	S	U	I	D	Category
assign	F	F	F	F	Reserved
rght	F	F	F	F	Reserved
rolerght	F	F	F	F	Reserved

* Only a part of the table is shown.

From Table III, we found that the attribute "email" in table "emp" is undefined, which means there is no insertion for this attribute, but there is an update operation for it. Although there is a case that an attribute has a default value and its insertion is not compulsory, we note that this might still be a flaw in design. However, by manual inspection, we found that in the project School Admin, the attribute "email" in fact has no default value, so this is actually an error or incompleteness of implementation. An additional function for this is thus needed.

- *Non-modifiable/Undeletable*

If the data lifecycle support information provided for an attribute or table in a database application is classified as non-modifiable or undeletable, then its adequacy depends on the application characteristics or the requirements; developers can examine whether additional programs are needed be coded for the adequacy of data lifecycle.

It can be seen from Table III that attribute "admid" in table "admission" is non-modifiable. This indicates that once a record is inserted, its attribute "admid" can never be changed. After investigation, we found that this attribute is not an attribute that requires special control to disallow from changes. The value of this attribute can be changed from time to time.

Besides, the attribute "cls" in table "classes" is undeletable. It is also possible that some attributes are not allowed to be deleted, but it is not the case for this attribute either.

Hence, additional functions should be added to the application to enable the users to modify or delete the values of the two attributes.

- *Redundant*

If the data lifecycle support information provided for an attribute or table in a database application is classified as redundant, a developer should examine whether it is really needed for the application. This may lead to the introduction of additional programs to use it or removal of unnecessary attribute or table.

As can be seen from Table III, the attribute "reasonofrej" in table "admission" is redundant for it is never used. According to this information, developers can decide either to add functions that use this attribute, or remove it and its other operations correspondingly.

- *Reserved*

If the data lifecycle support information provided for an attribute or table in a database application is reserved, a developer can decide whether he still wants to retain the reserved attribute/table.

From Table III, we learnt that attribute "empid" in table "emp" is reserved. Also from Table IV that, tables "assign", "rght" and "rolerght" are reserved. This is resulted from that there is not any operation on them. Consequently, whether to delete the attribute or tables from the schema depends on the real adaptation and developers. We found that these tables could be retained because the system is not fully developed yet and further enhancement and extension could be made.

As the above cases demonstrated, data lifecycle information reveals overall status and defect or incompleteness of implementation. Problems with the code could thus be located more easily and quickly. Since software maintenance could often be costly and time-consuming, especially when a new developer tries to have a broad picture of the existing system, data lifecycle can thus be a cost-effective facilitation for the maintenance process.

3) Verification

Software verification seeks to provide objective evidence that the design outputs of a particular phase of the software development life cycle meet the specification for that phase. When development reaches a stage, e.g. a milestone, developer would like to check the correctness of the specified programs in that stage. We show a simple example to demonstrate the use of data lifecycle in database application verification.

TABLE V. DATA LIFECYCLE OF ATTRIBUTE IN LAMP SCHOOL

Attribute	Table	S	U	I	D	Category
idalunno	alunni	T	F	F	T	Undefined
idcattedra	cattedre	T	F	F	T	Undefined

As is shown in Table V, the attribute "idalunno" in table "alunni" is undefined, i.e. there is no insertion operation for it. However, in the source code file "alu_conf.php", we found the following SQL query:

```
"SELECT * FROM alunni WHERE idalunno='$c'"
```

It can be seen that the attribute "idalunno" in table "alunni" is used as the condition in WHERE clause. However, since its value has never been defined, the result of this selection would be always empty and thus this operation would be inappropriate. This indicates design flaws or incompleteness of implementation against the original design.

Besides, there is another SQL query in a different source code file "del_cat.php" shows the same flaw:

```
"DELETE  FROM cattedre WHERE (idcattedra=
$_GET[idcl])"
```

The cases above are only two similar flaws example among all in the source code files. Data lifecycle support helps expose

such flaws with straightforward information. Reviews can thus be conducted to draw these problems.

V. Related Work

Software reverse engineering plays a critical important role in many aspects such as software maintenance and software comprehension. Over the past decades, a large number of studies have been conducted and quite a number of tools have been developed. In [1], Elliot J. Chikofsky and James H. Cross II et al provided taxonomy of software reverse engineering, in which six definitions of terms were given. Bellay and Gall reported an evaluation of four reverse engineering tools that analyze C source code [5]. They used a number of assessment criteria derived from Brown and Wallnau's Technology Delta Framework [6].

Reverse engineering has long been employed in software maintenance process. Rainer Koschke surveyed software visualization in software maintenance, reverse engineering and re-engineering in [7], where several representations for both software and specification were described. H. A. Müller presented using reverse engineering approach to aid subsystem identification, which could be used to provide better understanding and maintenance of a large software system [8]. P. Benedusi et al showed the use of hierarchical data flow diagrams in reverse engineering for software maintenance [9]. They focused on methodology at different levels of abstraction. Roger H. L. Chiang et al proposed extraction of ERR model from a relational database through data schema and data instance analysis [10]. However, we focus on extraction of data lifecycle from source code analysis.

Reverse engineering can also be used to enhance the understanding of software systems, and better understanding can result in better maintenance. S. Rugaber gave introductions of methodology, representation and tools in this area of research [11]. In [12], the authors described the results of the use of Rigi project in reverse engineering, which builds mental models from the discovered abstractions. Eleni Stroulia employed dynamic behavior analysis in reverse engineering to understand the system's process and uses [13]. Instead, our approach uses static analysis. J. Henrard et al described techniques in database reverse engineering [14]. However, they focused on generic methodology for construction of representation from different patterns, while our approach focuses on data lifecycle usages.

VI. Conclusion

Though the proposed approach is simple and useful, to the best of our knowledge, no such approach has been proposed in the literature. For a database application, there are four database operations supporting the data lifecycle: INSERT, UPDATE, SELECT and DELETE. We classify the possible sixteen combinations from these four operation types into six categories. We propose an approach to reverse engineer the data lifecycle automatically from the source code of the database applications. To verify the proposed approach, we developed a tool for PHP database applications and conducted case studies to evaluate the proposed approach using the tool.

The first benefit of our approach is the automatic data lifecycle extraction can lighten the burden of analyzing the code manually when the users try to understand the systems' database usage and behavior. Users can infer design flaws and completeness of the database from the extracted data lifecycle support directly. Secondly, the straightforward information provided by our approach can facilitate the selection, maintenance and verification of database applications.

Acknowledgment

We would like to take this opportunity to express our thanks to the developers of Pixy for releasing their tool as open source software, which helps a lot in our prototype tool development.

References

[1] Elliot J. Chikofsky, James H. Cross II, "Reverse Engineering and Design Recovery: A Taxonomy," *IEEE Software*, vol. 7, no. 1, pp. 13-17, Jan./Feb. 1990

[2] Nelson, M., A Survey of Reverse Engineering and Program Comprehension , *Arxiv preprint cs/0503068,* 2005

[3] Müller, H.; Jahnke, J.; Smith, D.; Storey, M.; Tilley, S. & Wong, K. Reverse engineering: A Roadmap, *Proceedings of the Conference on the Future of Software Engineering,* 2000, 47-60

[4] Jovanovic, N., Kruegel, C. and Kirda, E., Pixy: A Static Analysis Tool for Detecting Web Application Vulnerabilities, *Proc. IEEE Symp on Security and Privacy*, Oakland, CA, May 2006

[5] Bellay, B. & Gall, H., A Comparison Of Four Reverse Engineering Tools, *Proceedings of the Fourth Working Conference on Reverse Engineering, 1997*, 2002, 2-11

[6] A. Brown and K.Wallnau., A Framework For Evaluating Software Technology, *IEEE Software*, page 39-49, September 1996.

[7] Koschke, R. Software Visualization in Software Maintenance, Reverse Engineering, and Re-Engineering: A Research Survey, *Journal of Software Maintenance and Evolution: Research and Practice*, John Wiley & Sons, 2003, 15, 87-109

[8] Müller, H. A., Orgun, M. A., Tilley, S. R. and Uhl, J. S. (1993), A Reverse-Engineering Approach To Subsystem Structure Identification, *Journal of Software Maintenance: Research and Practice*, 5: 181–204

[9] Benedusi, P.; Cimitile, A. & De Carlini, U., A Reverse Engineering Methodology to Reconstruct Hierarchical Data Flow Diagrams for Software Maintenance, *Proceedings of International Conference on Software Maintenance, 1989.*, 2002, 180-189

[10] Chiang, R. H. L.; Barron, T. M. & Storey, V. C. Reverse Engineering of Relational Databases: Extraction of An EER Model From A Relational Database, *Data & Knowledge Engineering*, 1994, 12, 107 – 142

[11] Rugaber, S., Program Comprehension For Reverse Engineering, *AAAI Workshop on AI and Automated Program Understanding*, 1992

[12] Muller, H.; Wong, K. & Tilley, S., Understanding Software Systems Using Reverse Engineering Technology, *Object-Oriented Technology for Database and Software Systems*, 240-252

[13] Stroulia, E. & Systä, T., Dynamic Analysis For Reverse Engineering And Program Understanding, *ACM SIGAPP Applied Computing Review*, ACM, 2002, 10, 17

[14] Henrard, J.; Englebert, V.; Hick, J.; Roland, D. & Hainaut, J., Program Understanding In Databases Reverse Engineering, *Database and Expert Systems Applications*, 1998, 70-79

An Empirical Study on the Importance of Quality Requirements in Industry

Jose Luis de la Vara[1], Krzysztof Wnuk[2], Richard Berntsson Svensson[2], Juan Sánchez[1] and Björn Regnell[2]

[1]Centro de Inv. en Métodos de Producción de Software
Universidad Politécnica de Valencia
Camino de Vera s/n, 46022, Valencia, Spain
{jdelavara, jsanchez}@pros.upv.es

[2]Department of Computer Science
Lund University
PO Box 118, SE-22100 Lund, Sweden
{krzysztof.wnuk, richard.berntsson_svensson, bjorn.regnell}@cs.lth.se

Abstract— **Quality requirements can constrain many aspects of a software system and have a strong influence on its success. Therefore, they play a major role in the development of any software system. As software systems are designed to achieve various goals and perform different functions, the order of importance of quality requirements may vary depending on these goals and functions. Within this context, this paper aims to gain new insights into how industry deals with quality requirements. More specifically, its purpose is to analyze the importance of different types of quality requirements. An empirical study has been performed in order to determine practitioners' perspectives on this issue. The paper reports on a survey in which product managers, project leaders and programmers with different backgrounds have participated. Consequently, the paper provides further evidence about the perceived importance of types of quality requirements and how the importance may be different depending on the practitioners' roles, the types of project, the orders of magnitude of the requirements or the application domains.**

Keywords- requirements engineering, empirical study, survey, quality requirement, non-functional requirement, industry.

I. INTRODUCTION

The requirements of a software system can be considered the main indicators of its success [16] and of its quality [10]. As a result, the requirements engineering (RE) process has been widely recognized as the most important development stage. Not adequately addressing it can lead a software system to failure and increase its development time and cost [7].

Among the types of requirements for a software system [14], quality (aka non-functional) requirements (QRs) specify restrictions on a software system and how well it shall perform its functions. Some typical examples of types of QRs are usability, performance and security. The influence of these requirements on the success of a software system has been acknowledged both in academia [5] and in industry. Organizations can face many problems when dealing with QRs [4], and they can strongly constrain decisions related to important aspects of the development of a software system such as its architecture [23], planning [19] and evolution [8].

An important issue about QRs is that the importance of the types of QRs is not always the same. For example, it can vary among stakeholders' roles [11], types of project [2] and application domains [8]. If practitioners disregard this issue, then they may develop software systems whose quality is insufficient and/or does not meet the actual needs of its stakeholders. As a consequence, a project or a system may fail.

This paper aims to gain new insights into the importance of QRs in industry. For this purpose, an empirical study in the form of a questionnaire-based survey is presented. 31 practitioners from 25 organizations participated in the survey and determined the most important types of QRs for them and the rationale behind their decisions. The respondents also indicated their role, the types of project in which they participate, and the order of magnitude of the requirements [20] and the application domain of the projects.

The results of the study provide novel empirical evidence about the perceived importance of types of QRs in industry and how it may vary. Results are presented and analyzed for: a) product managers, project leaders and programmers; b) development of internal, contract-driven and market-driven software; c) small, medium and large-scale RE, and; d) public administration, telecommunications and software development application domains. The value of the results is twofold. First, they can help practitioners to determine the types of QRs that may be more important for a given software system and possible conflicting priorities on the QRs. Second, they can be very useful for academia to formulate hypotheses for new studies and to identify areas for further research.

The paper is organized as follows. Section II presents background and related work. Section III explains the research methodology that was followed. Section IV presents and discusses the results of the study. Finally, Section V summarizes our conclusions and future work.

II. BACKGROUND AND RELATED WORK

Gaining insights into RE practice has been and is an important goal for the RE research community [17], and empirical studies are essential to achieve it. Important contributions have been made up to date, but more studies are necessary yet. The first significant empirical study on RE practice [6] was published in 1988. Since then, many issues of RE in industry have been analyzed, including QRs.

This work has been developed with the support of the Spanish Government under the project PROS-Req TIN2010-19130-C02-02 and the program FPU AP2006-02324, the Valencia Regional Government under the project ORCA PROMETEO/2009/015, and ERDF.

Several facets of QRs such as elicitation, dependencies, metrics, cost estimation and prioritization have been addressed in empirical studies. A systematic literature review on them is presented in [3]. Among the studies listed in the systematic review and others published after the search of the systematic literature review was performed, we have found publications that addressed priorization and/or focused on the importance of types of QRs in industry. These works are the following ones.

Haigh [11] performed the existing survey with the largest sample. 318 students and alumni of a MBA ranked the importance of types of QRs on the most important software system in developing their work responsibilities. The factors analyzed were the stakeholders' role and the type of system. In a similar line, Berntsson Svensson et al. [2] examined how QRs of embedded systems were handled. They interviewed product managers and project leaders from five organizations, and analyzed the importance, interdependencies, quantification, dismissal, challenges and non-challenges of QRs.

Leung [15] conducted a user survey on QRs of intranet applications for which he obtained 30 valid responses. Another study on specific products was presented in [22]. A process framework for customization of software quality models was proposed and validated with two software products. The importance of types of QRs in the products was determined.

A survey with respect to cost and lead-time impact in development of software platforms is presented in [12]. This study analyzed 34 responses from two organizations and addressed the architect, system designer and marketing roles. Ameller and Franch [1] also analyzed QR-related practices of architects. 60 subjects participated in a web survey, and they indicated the importance of types of QRs in their projects.

Finally, Ernst and Mylopoulos [9] studied the importance of QRs along project lifecycle and the interest on them in different projects. They analyzed mailing list discussions, bug reports and commit logs of eight open-source products.

The contribution of this paper on understanding the perceived importance of QRs in relation to these works are that 1) it provides new insights about factors and treatments that have been studied previously (e.g., importance for project leaders), and 2) analyzes factors and treatments that have never been addressed (e.g., order of magnitude of requirements and software for public administration). With regard to the results of these works about common factors, they are used in Section IV for comparison with the results of this paper.

III. Research Methodology

The empirical study that is presented corresponds to a part of a survey on RE practices and perspectives in industry (hereafter referred to as wider survey). It was performed using a mixed research approach [21], i.e., there are parts of the study that correspond to quantitative (fixed) research whereas others correspond to qualitative (flexible) research.

On the one hand, some parts of the study and its analysis were pre-specified, e.g., overall rating of the importance of the types of QRs on the basis of the percentage of respondents that selected the types. These parts aimed to obtain quantitative data for comparison of variables and corresponded to the fixed side of the study. On the other hand, analysis of other parts evolved during the research process, e.g., analysis of some factors was not initially intended but was later decided. In addition, we aimed to investigate and understand phenomena within its real context and to seek new insights, ideas and possible hypotheses for future research. These conditions allowed us to regard the study as qualitative [18].

The following two subsections present the research process that was performed and discuss the validity of the survey.

A. Research Process

The study was performed through 6 stages [13].

Objectives definition: The first stage of the study involved brainstorming and meetings to identify the areas of interest. Of all the objectives of the wider survey, this paper addresses further understanding of the importance of QRs in industry. The research questions analyzed are: **RQ1:** What types of quality requirements are considered the most important ones by practitioners? **RQ2:** Is there any difference in the perspectives about their importance among roles, types of project, orders of magnitudes of the requirements or application domains?

Survey design: The survey was cross sectional [13] and explorative [25], and was administered by means of a web-based self-completion questionnaire. As explained above, a mixed approach was followed.

Development of a survey instrument: Relevant literature has been presented in Section II. The survey instrument was created with inspiration from [2]. We defined 21 types of QRs on the basis of Lauesen's comparison between ISO 9126 and McCall quality factors [14]. In addition to background information (about the possible factors and treatments to analyze), respondents were asked to indicate 1) the top five most important types of QRs, 2) the most important type, and 3) why they regarded that type as the most important one. Before listing the questions, a description of the survey was provided. It must also be indicated that more questions were asked for the wider survey.

Since the first and the fourth authors were initially responsible for data collection, it was expected that most of the respondents worked in Spain. Nonetheless, it was also planned to obtain responses from other countries. As a result, a Spanish version and an English version[1] of the (same) questionnaire were created. The expected time for completing the questionnaire (wider survey) was between 10 and 15 minutes.

Instrument evaluation: Four pilot studies were conducted for pre-testing, two for each version of the questionnaire. Time for completion was within our expectations, and some minor changes were made in question wording.

Data collection: Respondents were selected through a combination of maximum variation, convenience and snowball sampling [18] within our industrial collaboration network. We mainly aimed to obtain responses from product managers, project leaders and programmers of different organizations,

[1] http://hci.dsic.upv.es/jdelavara/REsurvey_instrument_en.pdf

TABLE I. SUMMARY OF RESPONSES AND RESULTS

Factor	Treatment	n	Top 5 types			#1 type	
			SEL	N_SEL	J_SEL	SEL	J_SEL
Role	Product manager	6	PERF (1), USAB (2), STAB (2)	MAIN	-	-	-
	Project leader	14	USAB (1), MAIN (2), SUIT (3) FLEX (3), SECU (5)	-	-	SUIT (1), USAB (2)	SUIT
	Programmer	11	MAIN (1), PERF (2), USAB (2)	-	-	-	-
Type of project	Internal software	6	MAIN (1), USAB (2), RELI (2), SUIT (2)	STAB	-	-	-
	Contract-driven	9	MAIN (1), USAB (2), PERF (3), CORR (4)	-	-	-	-
	Market-driven	10	PERF (1), USAB (1), RECO (3), RELI (3)	-	RECO	USAB (1)	-
Order of magnitude	Small	11	MAIN (1), USAB (2), PERF (3), FLEX (4)	-	-	USAB(1)	-
	Medium	8	USAB (1), MAIN (2), RELI (2)	-	ANAZ	MAIN (1)	MAIN
	Large	7	USAB (1), MAIN (1), PERF (3), SECU (4)	-	-	-	-
Application domain	Public admin.	9	USAB (1), MAIN (2), RELI (3)	-	CORR	-	-
	Telecom.	4	PERF (1), FAUL (2)	-	-	-	-
	Sw. development	4	USAB (1)	-	-	-	-

Types of QRs: analyzability (ANAZ), correctness (CORR), fault tolerance (FAUL), flexibility (FLEX), maintainability (MAIN), performance (PERF), recoverability (RECO), reliability (RELI), security (SECU), stability (STAB), suitability (SUIT), usability (USAB).

Perspectives: selected in the treatment (SEL), not selected just in that treatment of the factor (N_SEL), just selected in that treatment of the factor (J_SEL).

thus we focused on potential respondents' role for sampling. The role was the only treatment for the people contacted that we knew for all of them before analysis. They worked for regional, national and multinational organizations.

Analysis: This stage was performed by the first and the fourth author. 35 responses were obtained, but four were rejected because of inconsistent or lack of information. One was modified because the respondent indicated software testing as application domain ("other" option), which was considered to be part of software development. Therefore, we finally analyzed 31 valid responses (data points), 22 from the Spanish version of the questionnaire and 9 from the English one. The respondents worked for 25 different organizations[2]. Three responses from two organizations, two from two organizations and one from the rest of organizations were obtained. One respondent worked for several organizations, but it was regarded as a single organization for analysis.

After overall rating of the types of QRs, partition of responses involved determination of the factors (e.g., respondents' role), treatments of the factors (e.g., project leader) and perspectives in a treatment (e.g., usability is a top five type) that should be reported and thus analyzed according to the scope of the paper. Some criteria for determination of results to report and analyze were defined (Table II).

Criteria for respondent's perspectives aimed to determine selections and no selections of perspectives to analyze. P1 is related to the importance of a perspective in a treatment in general. P2 is related to the non-importance of a perspective in comparison to other treatments of a factor, and P3 in comparison to other perspectives of a treatment. P4 is related to the importance of a perspective just for a treatment. T1 and T2 aimed to determine the treatments to analyze, and F1 to determine the factors. These criteria apply to the whole questionnaire. Some extra ones for analysis and report of specific questions are described in Section IV.

Table I shows a summary of the data points (n) of the treatments (twelve) of each factor (four) that was considered for analysis. Most of the factors had less than 31 data points due to answer exclusions, i.e., the rest of data points until 31 correspond to answers that were not part of any of the treatments defined. Finally, it must be indicated that statistical tests [25] were not used for analysis. We did not use probabilistic sampling methods, thus statistical inferences could be made in the survey [13].

TABLE II. CRITERIA FOR DETERMINATION OF RESULTS TO ANALYZE

Criterion	Description
T1	A treatments is not analyzed if it has less than four data points (less than 10% of the sample)
P1	A perspective in a treatment that fulfils T1 is analyzed if at least three participants of the treatment (75% of the smallest sample for a treatment) selected the perspective
P2	A perspective in a treatment that fulfils T1 is analyzed if no participant selected the perspective but P1 is fulfilled in the perspective for all the other treatments of the same factor
P3	A perspective in a treatment that fulfils T1is analyzed if no participant selected the perspective but P1 is fulfilled in all the other perspectives of the same treatment
P4	A perspective in a treatment that fulfils T1 is analyzed if it fulfills P1 and P1 is not fulfilled in the perspective for all the other treatments of the same factor
T2	A treatment that fulfils T1 is not analyzed if it does not have any perspective that fulfils P1, P2, P3 or P4
F1	A factor is not analyzed if it all its treatments fulfill T1 or T2

B. *Validity*

This section discusses how threats to validity were addressed. We consider the four perspectives proposed in [25].

Construct validity: This validity is concerned with the relationship between a theory and its observation. Mono-operation bias was avoided by collecting data from respondents with different backgrounds, and evaluation apprehension by guaranteeing anonymity and confidentiality when publishing the results. A remaining validity threat was the provision of lists for determination of background information and selection of types of QRs. It may have been easier for respondents to use the list items provided than to propose others.

Conclusion validity: This validity is concerned with the relationship between a treatment and the conclusions drawn from it. For reliability of measures, pilot studies were

[2] Although identification of respondents' organization is not necessary for analysis in this paper, it is for some parts of the wider survey.

conducted and all the types of QRs and concepts that may have been unknown to the respondents (e.g., small-scale RE) were briefly described in the questionnaire. Reliability of treatment implementation was achieved by providing the same questionnaire to all respondents. We also consider that the criteria defined for analysis increased the possibility to draw correct conclusions. A remaining validity threat is the fact that the treatments determined for each factor have different data points. The influence of this fact on overall rating of the types of QRs is further discussed in Section IV.

Internal validity: This validity is concerned with the causal relationship between a treatment and its results. Maturation was addressed by designing an instrument whose completion should take between 10 and 15 minutes. Pilot studies and development of the questionnaire with close reference to literature and influence of an administered and validated existing instrument reduced instrumentation threats.

External validity: This validity is concerned with the generalization of the conclusions. Given the exploratory and qualitative nature of the study, it does not aim to generalize conclusions beyond its actual setting, but to explain and understand the phenomena under analysis. Nonetheless, understanding the phenomena may help in understanding other cases. Interaction of selection and treatment was avoided by just contacting product managers, product leaders and programmers for data collection. In addition, many results and conclusions coincide with the studies outlined in Section II.

IV. Results and Interpretation

This section describes and discusses the results of the study. They have been divided into selection of the set of top five types of QRs and selection of the most important type.

A. Top Five Types of QRs

This section presents the results that were obtained when respondents selected the top five types of QRs for them. When reporting the results (in Table I and in the text), the ranking of the types is put in brackets. The ranking was determined by ordering the types of QRs on the basis of the percentage of respondents that selected them. It must be noted that the respondents did not rank the types when selecting the top five ones, but just selected the five types of QRs that according to them were the most important ones.

Figure 1 shows the overall rating of the types of QRs as top five types in the survey. Usability (1), maintainability (2), performance (3), reliability (4) and flexibility (4) are the five types with the highest overall frequencies. Therefore, they are the overall top five types of QRs in the survey.

The number of times that these types were selected as top five in the twelve treatments is: eleven times for usability, nine for maintainability, seven for performance, four for reliability, and two for flexibility. This order is equal to the overall rating except for flexibility. Just security (sixth type) and suitability (eighth type) were selected the same number of times (two treatments in total) than some of the overall top five types. Therefore, we consider that the threat related to not having balanced data points did not strongly affect the overall rating.

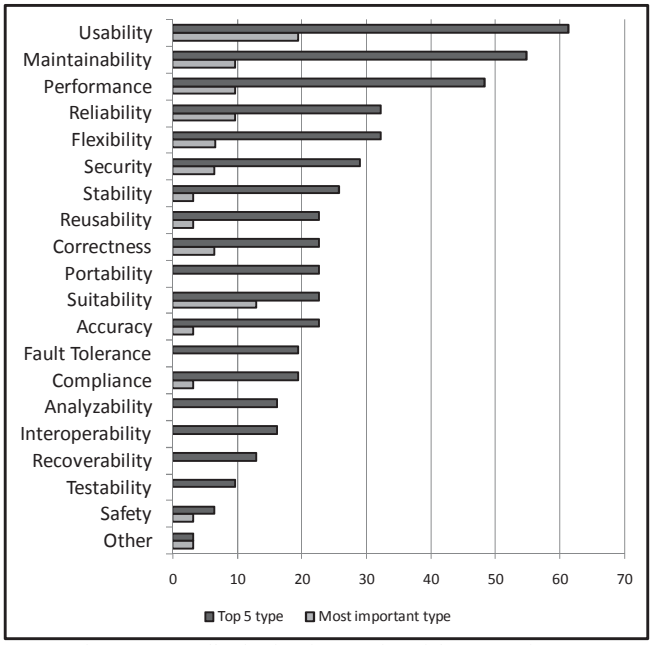


Figure 1. Overall selection frequencies of the types of QRs.

No respondent selected installabilty or replaceability, and one selected the "other" option. He specified *"hard core performance (speed and uptime, 99.999)"*, what we consider a specialization and combination of performance and reliability. This indicates that the types of QRs used in the survey can be combined when specifying requirements, and suggests that a clear cut between the types may not always exist in industry.

The overall rating is in line with related work. The most different results were the high ranking of accuracy and the low rankings of performance and flexibility in [11], and the low ranking of usability in [12]. The focus on software systems that respondents used and software platforms, respectively, may be the reasons. The results of [15] have not been considered for comparison because it just studied users' perspectives.

Table I shows a summary of the results for the treatments of each factor. The "SEL" column corresponds to perspectives that fulfill the criterion P1 (see Section III), the "N_SEL" column to perspectives that fulfill P2, and the "J_SEL" column to perspectives that fulfill P4. Less than five types of QRs are reported if P1 is not fulfilled (e.g., just one respondent selected the fifth type) or if more than five types should be reported (e.g., the fourth and seventh types have the same frequency).

When analyzing respondents' role, we can see that usability is a top five type for the three roles, and performance and maintainability for two. The difference in the selection of other types of QRs indicates that perspectives may vary among stakeholders. In relation to related work, performance, usability and stability were top five types of QRs for product managers in [2], and usability, maintainability and flexibility were for project leaders. In [11], maintainability and usability were top five types for developers, and maintainability and stability were for managers of development. The most different result is that no project leader selected suitability in [2].

For types of project, again just usability is ranked as a top five type for all the types of project, whereas maintainability,

performance and reliability are for two. Four types of QRs are top five types in several types of project, thus it seems that there is less variation in this factor than in others. Nonetheless, this factor is the only one in which the criteria P2 and P4 are fulfilled. The high importance of usability, performance and reliability for market-driven development was reported in [2][9][12]. However, recoverability was lowly ranked in [2].

With regard to the order of magnitude, maintainability and usability are top five types in all the scales, and performance is in two. The impact of the order of magnitude of requirements on the importance of QRs has not been previously analyzed. Therefore, no comparison is possible. Nonetheless, we think that the results suggest that it is a factor that can have influence, although the coincidences in perspectives may indicate that the influence of the factor is lower than of others.

Concerning specific application domains, we can see that just usability is a top five type for more than one domain. This is the factor with the lowest number of coincidences between treatments. Although the low number of data points of the treatments and of perspectives reported affects them, the results suggest that this factor has a higher influence on the variation of the importance than others. The importance of performance and fault tolerance in telecommunications was acknowledged in [8], and usability was reported as a top five type of QRs for development tools in [11].

In summary, the results provide new evidence about the fact that some types of QRs are more important than others in industry and that their importance can vary depending on stakeholders' roles, types of project, orders of magnitude of requirements and application domains. It also seems that some factors may have more influence on variation than others, and that the types of QRs used in the survey can be combined for specification. Finally, many hypotheses can be formulated from these results and could be tested in new studies. For example, that the importance of recoverability is higher in market-driven development than in other types of project.

B. Most Important Type of QRs

This section presents the results that were obtained when respondents were asked to select the type of QRs that was the most important one for them and to justify the selection. The first question allowed us to further analyze the results about the top five types, whereas the second one allowed us to gain new insights into the perceived importance of QRs.

Figure 1 shows the overall selection frequency of the types of QRs as the most important one. Usability (1), suitability (2), maintainability (3), performance (3) and reliability (3) are the types most frequently selected. This result is very similar to the overall rating of the types of QRs as top five types. The main difference is the presence of suitability. Further analysis of this fact shows that more than half the respondents that selected suitability as a top five type also selected it as the most important one. No other type has reached such a high frequency as most important type in relation to its selection as a top five type (if the "other" option is not considered). This suggests that when practitioners care about suitability, it is usually really important for them.

Table I shows a summary of the results. They further indicate the importance of suitability for project leaders and of usability in general. Usability is the only type that is ranked as the most important type of QRs and whose selection is reported for several treatments. With regard to the result about maintainability, we consider that it may need further study.

Maintainability was highly ranked in all the orders of magnitude of requirements when asking the top five types, but its selection as the most important type just in medium-scale RE suggests that the type is more important in that scale. This may have happened because of different reasons. For example, it may be thought that organizations that deal with large-scale RE need (and thus have) more mature software development processes, which impose execution of practices targeted at facilitating maintainability. Once these practices have been introduced, maintainability may be easier and consequently its perceived importance may decrease. Although important, the type would not be the most important one.

When reviewing the reasons behind selection of the most important type of QR, we realized that answers could be divided into three categories: product, project and customer-oriented. An example of product-oriented answer is *"flexibility is necessary for adaptation of a product to new use cases"*, of project-oriented answer is *"highly confidential data are managed in the projects"*, and of customer-oriented answer is *"security is the main point that clients demand"*. This division allowed us to perform two analyses that had not been planned and have not been addressed in any previous publication.

First, we analyzed the frequencies of the categories. Customer-oriented answers are the most frequent ones (at least half the answers had this orientation) for project leaders, market-driven development and small-scale RE, whereas product-oriented answers are for medium-scale RE. No product manager gave a project-oriented answer, and the perspective fulfilled the criteria P2 and P3. Finally, no product-oriented answer was obtained from small-scale RE and the perspective fulfilled P3. This analysis indicates that focus on products, projects or customers when dealing with QRs may vary depending on some factors.

Second, we realized that another analysis was possible. On the basis of existing works [24], we classified the types of QRs selected as the most important one into more important for users (customer-related) and more important for developers (development-related). Maintainability, flexibility, stability and reusability were considered development-related types of QRs, and the rest of types selected as the most important one were considered customer-related types.

Figure 2 shows the theoretical and actual frequencies of customer-related and development-related answers. The theoretical one was calculated on the basis of the previous classification and the selection frequency of the types of QRs as the most important one, whereas the actual one was calculated on the basis of the frequency of the categories of justification answers. The actual frequency of development-related answers was considered to be the sum of the frequencies of product-oriented and project-oriented answers, i.e., we considered that product-oriented and project-oriented answers were development-related.

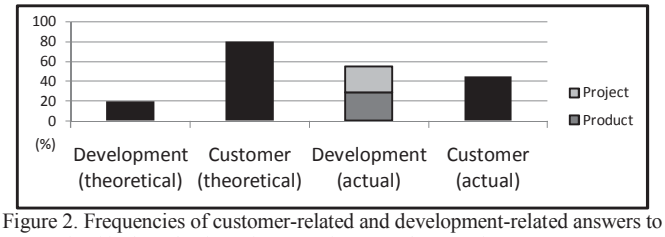


Figure 2. Frequencies of customer-related and development-related answers to justify the selection of a type of QRs as the most important one

Although the actual frequency of customer-related answers is the highest one in the survey, it is lower than in theory. This result suggests that there is a gap between theory and practice on the importance of types of QRs for customers or for development. Difference in the perception of types between academia and industry was discussed in [2], although not in relation to their importance for development and customers.

In summary, the results are in line with those from top five types of QRs. An exception is suitability, whose importance seems to be especially high for the practitioners that focus on it, and maintainability seems to be more important for medium-scale RE. The results also suggest that product, project and customer orientation when considering a type as the most important one may vary, and that there is a difference on this issue between theory and practice. Again, hypotheses can be formulated, such as that market-driven development mainly focuses on customers when prioritizing QRs.

V. CONCLUSIONS

This paper has presented an empirical study on QRs in industry. By means of a questionnaire-based explorative survey, practitioners' perspectives on the importance of types of QRs were obtained.

The results of the survey provide more and new evidence about how the importance of QRs may vary in industry. In general, usability, maintainability and performance are the three most important types for product managers, project leaders and programmers. Nonetheless, the importance of all the types may vary depending on different factors. Among them, evidence of variation has been provided for practitioners' roles, types of project, orders of magnitude of the requirements and application domains. In addition, although practitioners usually think of customer issues when selecting a type of QRs as the most important one, they may also think of development issues even when dealing with customer-related types.

The most immediate future work is to analyze the answers to the rest of questions of the wider survey. New empirical studies would also allow us to gain more insights into the importance of QRs in industry. These studies should aim to test hypotheses formulated from the results of this paper or to further analyze some phenomena by means of different empirical studies. For example, case studies would allow us to analyze the importance of QRs in project development settings.

ACKNOWLEDGMENT

The authors would like to thank all the people that participated and collaborated in the study, and Jose Ignacio Panach and Jose Luis Hervás for their comments on analysis.

REFERENCES

[1] D. Ameller and X. Franch, "How do software architects consider non-functional requirements: a survey", in REFSQ 2010, LNCS 6182, R. Wieringa and A. Persson, Eds. Heidelberg: Springer, pp. 276-277.

[2] R. Berntsson Svensson, T. Gorschek, and B. Regnell, "Quality requirements in practice: an interview study in requirements engineering for embedded systems". in REFSQ 2009, LNCS 5512, M. Glinz and P. Heymans, Eds. Heidelberg: Springer, pp. 218-232.

[3] R. Berntsson Svensson, M. Höst, and B. Regnell, "Managing quality requirements: a systematic review", in SEEA 2010, pp. 261-268.

[4] A. Borg, A. Yong, P. Carlshamre, and K. Sandahl, "The bad conscience of requirements engineering: an investigation in real-world treatment of non-functional requirements", in SERPS'03, pp. 1-8.

[5] L. Chung, B.A. Nixon, E. Yu, and J. Mylopoulos, Non-Functional Requirements in Software Engineering. Boston: Kluwer, 2000.

[6] B. Curtis, H. Krasner, and N. Iscoe, "A field study of the software design process for large systems", Commun. ACM, 31(11), pp. 1268-1287, 1988.

[7] A.M. Davis, Software requirements: objects, functions and states. Upper Saddle River: Prentice-Hall, 1993.

[8] C. Ebert, "Putting requirement management into praxis: dealing with non-functional requirements". Inf. & Softw. Technol., 40(3), pp. 175-185, 1998.

[9] N. A. Ernst and J. Mylopoulos, "On the perception of software quality requirements during the project lifecycle", in REFSQ 2010, LNCS 6182, R. Wieringa and A. Persson, Eds. Heidelberg: Springer, pp. 143-157.

[10] A. Finkelstein, "Requirements engineering: a review and research agenda", in APSEC'94, pp. 10-19.

[11] M. Haigh, "Software quality, non-functional software requirements and IT-business alignment", Softw. Qual. J., 18(3), pp. 361-385, 2010.

[12] E. Johansson, A. Wesslén, L. Bratthall, and M. Höst, "The importance of quality requirements in software platform development - a survey", in HICSS 2001.

[13] B.A. Kitchenham and S.L. Pfleeger, "Personal opinion surveys", in Guide to Advanced Empirical Software Engineering, F. Shull, J. Singer, and D.I.K. Sjoberg, Eds., London: Springer, pp. 63-92, 2008.

[14] S. Lauesen, Software Requirements - Styles and Techniques. London: Addison-Wesley, 2002.

[15] H.K.N. Leung, "Quality metrics for intranet applications", Inf. & Manage., 38(3), pp. 137-152, 2001.

[16] B. Nuseibeh and S. Easterbrook, "Requirements engineering: a roadmap", in Future of Software Engineering, ICSE'00, pp. 35-46.

[17] B. Paech, T. Koenig, L. Borner, and A. Aurum, "An analysis of empirical requirements engineering survey data", in Engineering and Managing Software Requirements, A. Aurum and C. Wohlin, Eds. New York: Springer, pp. 427-452.

[18] M.Q Patton, Qualitative Research and Evaluation Methods, 3rd ed., London: Sage Publications, 2002.

[19] B. Regnell, R. Berntsson Svensson, and T. Olsson, "Supporting roadmapping of quality requirements", IEEE Softw., 25(2), pp.42-47, 2008.

[20] B. Regnell, R. Berntsson-Svensson, and K. Wnuk, "Can we beat the complexity of very large-scale requirements engineering?", in REFSQ 2008, LNCS 5025, B. Paech and C. Rolland, Eds. Heidelberg: Springer, pp.123-128.

[21] C. Robson, Real World Research, 2nd ed. Oxford: Blackwell, 2002.

[22] M. Sibisi and C.C. van Waveren, "A process framework for customising software quality models", in AFRICON 2007, pp. 547-554.

[23] M. Svahnberg, "An industrial study on building consensus around software architectures and quality attributes", Inf. & Softw. Technol., 46(12), pp. 805-818, 2004.

[24] K.E. Wiegers, Software Requirements, 2nd ed. Redmond: Microsoft Press, 2003.

[25] C. Wohlin, P. Runeson, M. Höst, C. Ohlson, B. Regnell, and A. Wesslén, Experimentation in Software Engineering: An Introduction. Boston: Kluwer, 2000.

An Empirical Study on Classification of Non-Functional Requirements

Wen Zhang, Ye Yang, Qing Wang, Fengdi Shu
Laboratory for Internet Software Technologies
Institute of Software, Chinese Academy of Sciences
Beijing 100190, P.R.China
{zhangwen,ye,wq, fdshu}@itechs.iscas.ac.cn

***Abstract*—The classification of NFRs brings about the benefits that NFRs with respect to the same type in the system can be considered and implemented aggregately by developers, and as a result be verified by quality assurers assigned for the type. This paper conducts an empirical study on using text mining techniques to classify NFRs automatically. Three kinds of index terms, which are at different levels of linguistical semantics, as N-grams, individual words, and multi-word expressions (MWE), are used in representation of NFRs. Then, SVM (Support Vector Machine) with linear kernel is used as the classifier. We collected a data set from PROMISE web site for experimentation in this empirical study. The experiments show that index term as individual words with Boolean weighting outperforms the other two index terms. When MWEs are used to enhance representation of individual words, there is no significant improvement on classification performance. Automatic classification produces better performance on categories of large sizes than that on categories of small sizes. It can be drawn from the experimental results that for automatic classification of NFRs, individual words are the best index terms in text representation of short NFRs' description and we should collect as many as possible NFRs of software system.**

***Keywords*—Non-Functional Requirements, Automatic Classification, Support Vector Machine.**

I. Introduction

Non-functional requirements (NFRs) describe the expected qualities of a software system such as usability, security and look and feel of user interface. These qualities do impact on the architectural design of the software system [1] and satisfaction of stakeholders relevant with the software system [2] [3] [4]. NFRs are put forward by stakeholders with no expertise in software engineering and the number of NFRs is often very large. Moreover, NFRs are scattered over the functional requirements and the types of the NFRs are unknown because they are written in natural language. These charateriestics of NFRs make its classification a labor-intensive and time-consuming job without the adoption of automatic techniques. Nevertheless, software developers cannot ignore NFRs because they are decisive on the success of the software system.

Compared with functional requirement, NFRs tend to be properties of a system as whole, and hence cannot be verified for individual components or modules [19] [23]. This makes the classification of NFRs necessary in development because , on the one hand,we need to handle NFRs in a different way from functional requirements and, different types of NFRs should be handled by developers with different expertise.

For instance, in a security critical, mission critical, or economy vital software systems, formal method with model checking of the correctness of specification on system security is used to determine whether or not the ongoing behavior holds of the system's structure. However, a serious (and obvious) drawback of model checking is the state explosion problem because the size of the global state graph is (at least) exponential in the size of the program text [21]. Thus, an exigent demand from system developers is to identify the security-related NFRs from the requirement document and express them using formal specifications, and then to conduct verification of the satisfaction of system's properties on those security specifications.

On the other hand, the classification of NFRs benefits an overall decision on the systems' satisfaction on each category of NFRs and the progress of system development developers have made. Functional requirements can be measured as either satisfied or not satisfied, but NFRs can not merely measured by a linear scale as degree of satisfaction [25]. In system testing, for example, we can customize our test strategy based on the classification of NFRs and thus system behaviors of NFRs were reported directly to the project manager.

Most, if not all, projects have resource limitations and time constraints, with different requirements having different concerns of software systems. Even after the requirements are elicited and collected, they are still just an unorganized set of data. As such, it is necessary to categorize the requirements into different types so as to match problems and solutions in separate domain of discourse, especially for large-scale software projects with hundreds or even thousands of requirements.

In human resource allocation and optimization [22], different developers possess different expertise in handling various aspects of software development. Different tasks in development may need different expertise and capability from the developers. Thus, a match of developers and tasks is at the core of the success of software development. The identification of different types of NFRs results in the formation of different types of development tasks. Consequently, those tasks are assigned to developers aggregately according to their expertise and capability level. For instance, the NFRs with "performance" type and the NFRs with "maintainability" type should be dealt with by developers with different expertise.

We forward NFRs of the type "look and feel" to UI (User Interface) design experts of the system so that the satisfaction of these NFRs can be ensured throughout the whole system.

Methods for the elicitation of NFRs include questionnaires, checklists and templates for inquiring stakeholders concerning quality issues [24]. Basically, NFRs are made of textual sentences whose contents concern the expected qualities of a software system. For instance, "*The application shall match the color of the schema set forth by Department of Homeland Security. LF*" is a typical NFR record fetched out from the data set and it comprises two parts: textual description and NFR type as "*LF*". The description of NFR is very simple and easy to understand. It does not need professional knowledge of engineering aspects. Thus, text mining, which combines both natural language processing (NLP) and statistical machine learning, can be used for the task of automatic classification of NFRs.

Although there is substantial literature on NFR classification [2] [4] [5] [17] , we find two problems. First, most methods are purely intuitive and derived without theoretical support or mathematical model. Also, some methods are extremely labor-intensive and time-consuming, and others are qualitative, without quantitative analysis. Second, the index terms used in existing automatic classification of NFRs are keywords extracted directly from requirements without feature selection. This would cause huge dimensionality of NFR vectors and consequently bring about huge computation when quantitative methods are used. Most importance, we are uncertain that other index terms than keywords are more appropriate for automatic classification of NFRs.

The questions devised for this empirical study are: 1) among existing indexing methods, which one is the best performance for automatic classification of NFRs, and 2) is it possible to produce better performance with SVM than those derived in previous work?

The remainder of this paper is organized as follows. Section 2 describes the research approach employed in this paper. Section 3 conducts experiments of using SVM and different index terms to classify NFRs. Section 4 present the related work and Section 5 concludes this paper.

II. Research Approach

The research approach adopted in the empirical study is shown in Figure 1 to automate the classification of NFRs. First, NFRs' textual description are represented using different types of index terms as N-grams, individual words, and MWEs, respectively. Then, we transfer NFR textual description into numerical vectors in different feature space determined by different types of index terms. Second, support vector machine (SVM), which is a popular classifier in machine learning [6] [9], is used to classify NFR vectors. Finally, performance of each classification is evaluated.

A. Index Terms

The requirement data set we collected from the PROMISE web site (http://promisedata.org/repository) and it contains 625 records of both functional and non-functional requirements.

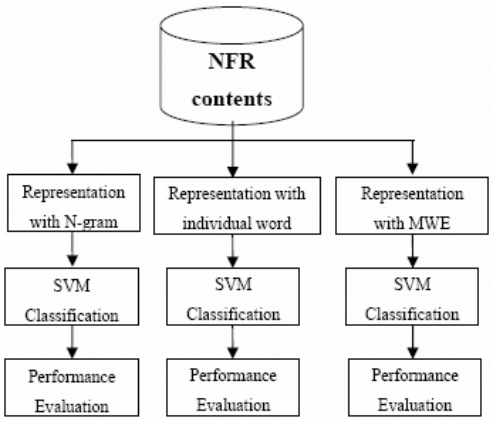


Fig. 1. Procedures of automatic classification of NFRs.

N-gram Representation. N-gram is proposed in text mining to categorize documents with textual errors such as spelling and grammatical errors, and it has been proved as an effective technique to handle these kinds of errors [7]. An N-gram is an N-character contiguous fragment of a long string. Since every string is decomposed into small fragments, any errors that are present in words only affect a limited number of those fragments. If we measure the similarity of two strings based on their N-grams, we will find that their similarity is immune to most textual errors. We used bi-gram (344 2-grams) and tri-gram (1,295 3-grams) for representation of NFRs, respectively.

Word Representation. The method of using individual words of a text content to represent the text can be traced to Salton et al[10]. We follow this idea to use words in NFRs for representation. First, we eliminated stop words from NFRs' description [1]. Second, word stemming [2] was conducted to map word variants to the same stem. We set the minimum length of a stem as 2. That is, only the stems which have more than 2 characters will be accepted as stems of words. Thus, 1,127 individual word stems are produced from the data set.

MWE Representation. We used the method proposed by Justeson and Katz [12] for MWE extraction from NFRs. The basic idea behind this method is that a MWE should include 2 to 6 individual words and it should occur in a collection of documents more than twice. Nevertheless, the part of speeches of individual words of a MWE should meet the regular expression described in formula 1.

$$((A \mid N)^{+} \mid (A \mid N)^{*}(NP)^{?}(A \mid N)^{*})N \qquad (1)$$

Here, *A* denotes an adjective, *N* denotes a noun and *P* denotes a preposition. Using the method adopted from [11], we extracted 93 MWEs from the data set.

[1]We obtain the stop words from http://ftp.uspto.gov/patft/help/stopword.htm

[2]we use Porter stemming algorithm that is available at http://tartarus.org/martin/PorterStemmer/

B. Feature Selection

In this paper, we adopt information gain (IG) [8], which is a classic method for feature selection in machine learning, to select informative index terms for automatic classification of NFRs. IG is defined as the expected reduction in entropy caused by partitioning NFRs according to a given term. The formula of IG is presented in Equation 2 and the formula of entropy is depicted in Equation 3.

$$Gain(S, A) = Entropy(S) - \sum_{v \in Value(A)} \frac{\mid S_v \mid}{\mid S \mid} Entropy(S_v) \tag{2}$$

$$Entropy(S) = \sum_{i=1}^{c} -p_i \log p_i \tag{3}$$

Here, *S* is the collection of the types of all NFRs such as PE (performance) and US (usability). *Value(A)* is a set of all possible values of index term *A*. S_v is a subset for which *A* value *v*, *c* is the number of categories of all NFRs, and p_i is the proportion of the NFRs that belong to category *i*. To observe the performances of textual features on automatic classification dynamically, the different feature sets of textual features are constructed at different removal percentages of low IG value terms (see also removal ratio in Section 3.2).

C. SVM Classifier

The classifier used for automatic classification is support vector machine (SVM) [6] [9], that is introduced in statistical machine learning. We selected this method in this paper because Gokyer et al [9] used it to transfer NFRs to architectural concerns and preferred effectiveness was achieved. In this paper, the linear kernel (u*v) is used for SVM training because it is superior to non-linear kernels for classifying textual contents validated by prior research [8] [13].

III. Experiment

A. Categories of the data set

Table 1 lists the categories of both functional and non-functional requirements in the data set and their numbers. Functional requirements occupy the largest proportion in the data set, amounting to 40.8% (255/625). Moreover, there is a skew distribution among the NFR categories in the data set. For instance, the largest category "usability" has 67 cases while the smallest category "Palm Operational" has only one case. We adopt binary classification in this paper because skew distribution often deteriorates the performance of multi-class classification [28] .

In binary classification, if the number of positive (negative) cases is much larger than the number of negative (positive) cases, the performance of automatic classification may be not convincing because of the biased distribution of data points. For example, if 90% cases in a data set are positive, then the accuracy of automatic classification should be greater than 90% if the classifier predicts all the data points with positive labels. For this reason, the categories "usability", "security", and "Look And Feel" with relatively medium sizes in the data set are used as the positive classes for binary classification one by one and meanwhile, the NFRs in remaining categories are used as negative classes.

TABLE I
Categories of requirements and their numbers in the data set

Abbr.	Full Name	Num
F	Functional	255
US	Usability	67
SE	Security	66
O	Operational	62
PE	Performance	54
LF	Look And Feel	38
A	Availability	21
SC	Scalability	21
MN	Maintainability	17
L	Legal	13
FT	Fitness	10
PO	Palm Operational	1

B. Classification Process

There are actually two kinds of work involved in representing textual contents: indexing and term weighting. Indexing is the job of assigning index terms for textual contents and term weighting is job of assigning weights to terms, to measure the importance of index terms in textual documents. We employ Boolean values as term weights for the index terms for the reason that most NFRs in the data set are very short and contain less than 20 index terms so they do not need complex weighting schemes such as those mentioned for document representation [15].

IG is employed to change the percentages of index terms used for representation. The removal ratio is predefined to remove the index terms with small entropy. For instance, if we set the removal ratio to 0.1 for representation with individual words, then a percentage of 90% of individual words with smaller entropy will be eliminated from the feature set and we only use the remaining 10% of individual words for the representation of NFRs. The purpose of varying different percentages of index terms for representation is that we want to observe the robustness of classification performances when the set of index terms become smaller and smaller. This is especially important for deciding which type of index terms should be used for representing NFRs when computation capacity is not enough to support large dimension of vectors in training classifier. In representation with MWEs, we use all the 1,127 individual words and a proportion, which is is defined by the removal ratio, of MWEs as the index terms.

The experiments in this paper are carried out with 10-fold cross-validation technique. In each experiment, we divide the whole data set (for both positive and negative classes) into 10 subsets. The 9 of 10 subsets are used for training and the remaining one subset is used for testing. We repeat the experiment 10 times and the performance of the classification is measured as average precision and recall [2] of the 10 repetitions.

C. Experimental Results

The outcome of our experiments shows that the precisions of all the automatic classification tasks are significantly higher than those of the classification approach proposed by Cleland-Huang et al [2]. Yet, the recalls of the automatic classifications in this paper are not comparable to the classification proposed by Cleland-Huang et al. We do not list the precisions and recalls produced in our experiments due to space limitation (Readers who have an interest in the precisions and recalls are welcome to ask the authors for more details).

We conjecture that the high recalls of the results of Cleland-Huang et al [2] can be attributed to the small size (within the range between 10 and 20) of index terms they employed in their experiments, which is much smaller than the sizes of index terms (within the range between 100 and 1,000) used in our experiments. When small size of index terms is used, larger number of NFRs is classified as relevant NFRs of the category. Consequently, more irrelevant NFRs will be misclassified as relevant. Considering an extreme case of classifying relevant NFRs with "Usability", if all the NFRs are regarded as relevant with "Usability", then the recall of automatic classification will be 100%. However, this classification result may be of less help for automating the task of classification. Thus, we argue that for automatic classification of NFRs, precision should be given more importance if recall is acceptable.

The F-measure [16] described in Equation 4 combines both precision and recall for performance evaluation. We used F-measure as the indicator for performance evaluation. In general, the larger the F-measure is, the better the classification result is. Here, for the purpose of comparison, we make out the F-measures of Cleland-Huang et al [2] as the baseline performance in the experiments.

$$F - measure = \frac{2 \times precision \times recall}{precision + recall} \quad (4)$$

Figures 2-4 show the experimental results of classifying the assigned three categories "Usability", "Security", and "Look And Feel" at different removal ratios using four types of index terms (2-gram, 3-gram, word and MWE) to represent NFRs.

First, it can be seen that the representation with individual words has the best performance measured by F-measure nearly on all three categories. In some cases, representation with terminologies improves the precision of automatic classification of representation with individual words (when the removal ratio arrives at 0.9). That is, even if most index terms are removed from the feature set, their performances do not decline drastically. However, in most cases, representation with terminologies is not able to improve the performances of automatic NFR classification compared with representation with individual words, even worse than that of 3-gram representation. Both representation with individual words and terminologies produce very robust classification.

This outcome implies that for automatic classification of NFRs, representation with individual words is sufficient to produce a desirable performance. This outcome is very different from the experimental results produced by Zhang et al

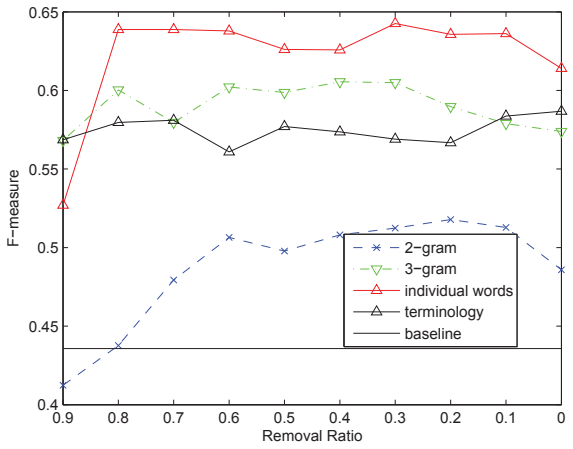


Fig. 2. Classification performances on category "Usability".

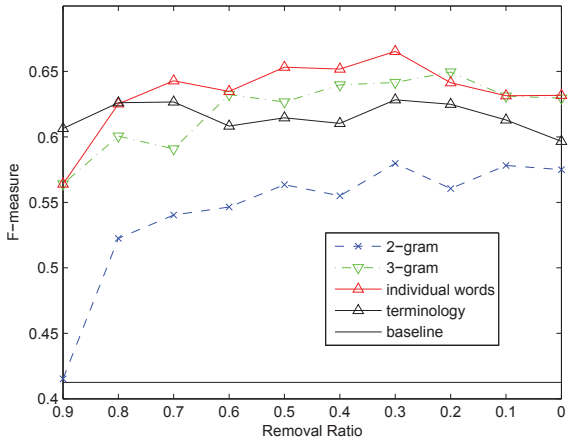


Fig. 3. Classification performances on category "Security".

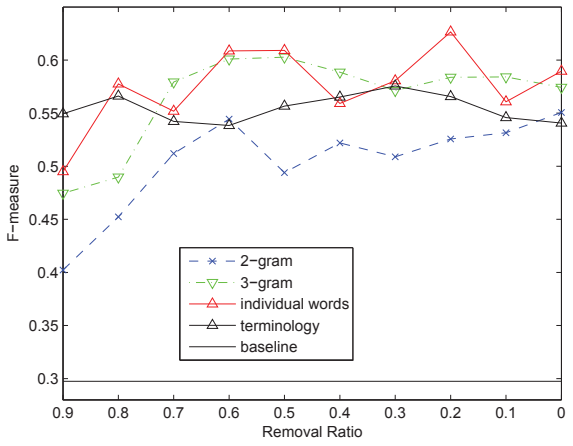


Fig. 4. Classification performances on category "Look And Feel".

[8]. They argued that MWEs are superior to individual words for classifying news documents automatically. We explain that the lengths of NFRs in the data set are much shorter (20 individual words on average) than those of Reuters-21578 news documents (80 individual words on average) so the semantics inherent in textual contents of NFRs is not so much important as those inherent in news documents.

Second, for representation with N-grams, representation with 3-grams outperforms that with 2-grams. This outcome can be attributed to that the number of 3-grams is much larger than that of 2-grams (see Section 2.1). Moreover, the robustness of the 3-gram representation is better than the 2-gram representation because the magnitudes of variations of F-measures of the 3-gram representation are smaller than those produced by the 2-gram representation.

Thirdly, for classification on different categories, the performance on "Security" is as almost the same as that on "Usability" but outperforms that on "Look And Feel". The number of NFRs in category "Security" (66) is approximately equal to that in category "Usability" (67), which is much larger than the number of "Look And Feel" (38). We explain that automatic classification would produce more favorable performance on categories those having large sizes than those having small sizes.

Based on the experimental results, the answer for question 1 devised for this case study in Section 1 should be that to date, keywords are the most effective index terms for automatic classification of NFRs and for question 2, our answer is that the machine learning technique, at least SVM, can improve the classification significantly.

IV. Related Work

NFR has attracted much interest of researchers from software engineering field. Much work has been invested in managing NFRs. Tran and Chung [20] proposed a prototype tool for explicit representation of NFRs. They aim to consider NFRs in a more goal-oriented perspective and argue that NFRs have much influence on the design of the solution as well as reinforce engineering process. In order to formulate implicit relationships of NFRs , ontology and graphic visualization are used in their tool to express softgoals and their interdependencies explicitly.

Cleland-Huang et al [4] introduce a goal-centric approach to managing the impact of change upon the NFRs of a software system. They partition NFRs into different softgoals of a system and construct a softgoal interdependency graph (SIG) to trace both direct and indirect impacts of software changes on NFRs. Probabilistic network model is employed to enable the traceability of impacts. However, the job of constructing SIG is labor-intensive and time-consuming because of the lack of automatic approaches. If we can classify NFRs into different softgoals automatically, it would be much easier to identify the relations between subgoals and softgoals. That is, human workload involved in constructing SIG will be reduced to a great extent.

In another work, Cleland-Huang, et al.[2] proposed an information retrieval method to discover and identify NFRs from system specification. Their basic assumption is that different types of NFRs are characterized by distinct keywords (index terms) that can be learned from documents of that type. However, their method of selecting index terms is ad-hoc and does not consider any linguistic properties of those index terms. Moreover, the classifier used in their method, which is based on the additive weights of index terms on a given NFR type, is quite simple without any theoretical ground.

Rosenhainer [14] proposed aspect mining to identify cross-cutting concerns in requirements specifications. Two techniques are suggested to be used for aspect mining: identification through inspection and identification supported by information retrieval (IR). The former one is manual and the later one is semi-automatic. Rosenhainer argued that IR-based technique is more promising than manual method and their experimental results on interactions between functional and non-functional requirements have validated this argument. This work encourages our study in this paper to use IR techniques for automatic classification of NFRs.

Casamayor et al [24] employed naïve Bayes and EM (Expectation Maximization) algorithm as a semi-supervised learning approach to classify non-functional requirements in textual specifications. They used the same data set as used in this paper and reported that their algorithm produced an average accuracy above 70%. In fact, the performance of SVM in our experiments is better than theirs because we exclude those categories of small number of data points. That is, unbalanced distribution of data points is purposely alleviated in our experiments. Moreover, our conjecture is also validated by their experimental results that those category of large number of data points such as "usability", "security" and "Look And Feel".

V. Concluding remarks

NFR is crucial to the success of a software system as it describes necessary qualities of system to avoid devastating effects and system failure [17]. In this empirical study, vector space model and machine learning technique are employed to classify NFRs automatically. We used different index terms to transfer NFRs into numeric vectors and examined their performances on automatic classification of NFRs. The machine learning classifier we adopted in this paper is SVM with linear kernel, which is widely recommended as a promising classifier for text mining. Information gain for feature selection and SVM for automatic classification is introduced. We conducted experiments using the data set collected from PROMISE data set.

The experimental results show that individual words, when used as the index terms, have the best performance in classifying NFRs automatically. We noticed that, in most cases, the more samples in a category in data set, the better performance the automatic classifier will produce on the category. This outcome illustrated that the number of NFRs in the data set is an important factor for automating the classification of NFRs.

This inference suggests that we need to collect NFRs as many as possible if automatic classification is desired.

This work can be applied in at least two aspects in software engineering currently. First, it can be used to identify NFRs in requirement specification. Usually, customers, testers, and stakeholders of a software system will mix their desirable qualities in a specification. Whereas functional requirements describe what the system needs to do, NFRs describe constraints on the solution space and capture a broad spectrum of properties, such as usability and security [18]. Because the solutions of functional requirements and NFRs are different, or the time to consider these two kinds of requirement in system design is different, we must differentiate these two kinds of requirements. Second, different NFRs are often handled by different designers and developers with different background knowledge in architecture, it is crucial to classify the NFRs into different categories so that the NFRs in the same category can be processed as a comprehensive requirement in system design.

ACKNOWLEDGMENTS

This work is supported by the National Natural Science Foundation of China under Grant Nos. 90718042, 60873072, 61073044, and 60903050; the National Science and Technology Major Project; the National Basic Research Program under Grant No. 2007CB310802; the Scientific Research Foundation for the Returned Overseas Chinese Scholars, State Education Ministry.

REFERENCES

[1] Y. Yang, J. Bhuta and D. Port and B. Boehm, Value-Based Processes for COTS-Based Applications, IEEE Software, 22(4), 54-62, July/August, 2005.

[2] J. Cleland-Huang, R. Settimi and X. Zou and P. Solc, The Detection and Classification of Non-Functional Requirements with Application to Early Aspects, In Proceedings of the 14th IEEE International Requirements Engineering Conference, 36-45, 2006.

[3] B. Nuseibeh, Weaving Together Requirements and Architecture, IEEE Computer, 34(3), 115-117, 2001.

[4] J. Cleland-Huang, et al, Goal-Centric Traceability for Managing Non-Functional Requirements, In Proceedings of the 27th International Conference on Software Engineering, 362-371, 2005.

[5] D. Zhang and J. J. P. Tsai, Machine Learning and Software Engineering, Software Quality Journal, 11, 87-119, 2003.

[6] V. Vapnic, The Nature of Statistical Learning Theory, Springer, New York, 1995.

[7] W. B. Cavnar and J. M. Trenkle, N-Gram-Based Text Categorization, Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval, 161-175, 1994.

[8] W. Zhang, T. Yoshida and X. J. Tang, Text classification based on multi-word with support vector machine, Knowledge-based Systems, 21(8), 879-886, 2008.

[9] G. Gokyer, et al, Non-functional Requirements to Architectural Concerns: ML and NLP at Crossroads, Proceedings of International Conference on Software Engineering Advances, 2008.

[10] G. Salton and A. Wong and C. S. Yang, A Vector Space Model for Automatic Indexing, Communications of the ACM, 18(11), 613-620, 1975.

[11] W. Zhang, T. Yoshida and X. J. Tang, Improving effectiveness of mutual information for substantival multiword expression extraction, Expert Systems with Applications, 36(8), 10919-10930, 2009.

[12] J. S. Justeson and S. M. Katz, Technical terminology: some linguistic properties and an algorithm for identification in text, Natural Language Engineering, 1(1), 9-27, 1995.

[13] Y. M. Yang and X. Lin, A re-examination of text categorization methods, In Proceedings on the 22nd Annual International ACM Conference on Research and Development in Information Retrieval, 42-49, 1999.

[14] L. Rosenhainer, Identifying Crosscutting Concerns in Requirements Specifications, In Proceedings of OOPSLA Early Aspects 2004: Aspect-Oriented Requirements Engineering and Architecture Design Workshop, 2004.

[15] G. Salton and C. Buckley, Term weighting approaches in automatic text retrieval, Information Processing Management, 24, 513-523, 1998.

[16] J. Han and M. Kamber, Data Mining Concepts and Techniques, Morgan Kaufmann Publishers, New York, 2006.

[17] F. Brooks, No sliver bullet-essence and accidents of software engineering, IEEE Computer, 20(4), 10-19, 1987.

[18] I. Sommerville and P. Sawyer, Viewpoints: Principles, Problems, and a Practical Approach to Requirements Engineering, Annals of Software Engineering, 3, 101-130, 1997.

[19] B. Nuseibeh and S. Easterbrook, Requirement Engineering: A Roadmap, Proceedings of the Conference on The Future of Software Engineering(ICSE'00), 35-46, 2000.

[20] Q. Tran and L. ChungL, NFR-Assitant: Tool Support for Achieving Quality, Proceedings of IEEE Symposium on Application-Specific Systems and Software Engineering and Technology (ASSET'99),1999.

[21] E. A. Emerson, The beginning of model checking: A personal perspective, 25 Years of Model Checking, 27-45, 2008.

[22] J. Xiao, L. J. Osterweil, Q. Wang and M. Li, L, Disruption-Driven Resource Rescheduling in Software Development Processes, In Proceedings of 4th International Conference on Software Process, 234-247, 2010.

[23] L. Chung, J. Prado Leite, On Non-Functional Requirements in Software Engineering, A.T. Borgida et al. (Eds.): Mylopoulos Festschrift, LNCS 5600, 363-379, 2009.

[24] A, Casamayor, D. Godoy and M. Campo, Identification of non-functional requirements in textual specifications: A semi-supervised learning approach, Information and Software Technology, 52, 436-445, 2010.

[25] F. Tsui and O. Karam, Essentials of Software Engineering (Second Edition), Jones and Bartlett Publishers, Sudbury, Masshachusetts, 2011.

[26] G. Salton and M.J. McGill, Introduction to Modern Information Retrieval, McGraw-Hill Book Company, New York, NY, USA, 1986.

[27] J. Davis and M. Goadrich, The relationship between precision-recall and ROC curves, In Proceedings of the 23rd international conference on Machine learning, 233-240, 2006.

[28] J. Weston and C. Watkins, Multi-class support vector machines, In Proceedings of 7th European Symposium on Artificial Neural Networks, 219-224, 1999.

Assessing the Impact of Aspects on Design By Contract Effort: A Quantitative Study

Henrique Rebêlo[1] Ricardo Lima[1] Uirá Kulesza[2] Cláudio Sant'Anna[3]
Roberta Coelho[2] Alexandre Mota[1] Márcio Ribeiro[1,4] César Oliveira[1]

[1]Federal University of Pernambuco, PE, Brazil
{hemr, rmfl, acm, mmr3, calo}@cin.ufpe.br
[2] Federal University of Rio Grande do Norte, RN, Brazil
{uira, roberta}@dimap.ufrn.br
[3] Federal University of Bahia, BA, Brazil
santanna@dcc.ufba.br
[4] Federal University of Alagoas, AL, Brazil
marcioribeiro@ic.ufal.br

Abstract

Although it is assumed that the implementation of design by contract is better modularized by means of aspect-oriented (AO) programming, there is no empirical evidence on the effectiveness of AO for modularizing non-trivial design by contract code in well-understood modularity attributes. This paper reports a quantitative case study of the adequacy of aspects for modularizing design by contract concern. The study consisted of refactoring a real-life application so that the code responsible for implementing the contract enforcement strategies was moved to aspects. Our analysis was driven by fundamental modularity attributes, such as separation of concerns, coupling, and size. We have found that AO techniques improved separation of concerns between the design by contract code and base application code. However, contradicting the general intuition, the AO version of the system did not present significant gains regarding four classical size metrics we employed.

1 Introduction

Design by Contract (DbC), originally conceived by Meyer [12], is a technique for developing and improving functional software correctness. The key mechanism in DbC is the use of the so-called "contracts". A contract formally specify an agreement between a client and its suppliers. Client classes must satisfy the supplier class conditions before calling one of its methods. When these conditions are satisfied, the supplier class must guarantee certain properties, which constitute the supplier class's obligations. However, when a client breaks a condition (client violation), a runtime error occurs. The use of such pre- and postconditions and invariants to specify software contracts dates back to Hoare's 1969 paper on formal verification [7]. The novelty with DbC is to make these contracts executable. This is useful for isolating errors during debugging, and for validating contracts that are used as documentation or for increasing code reliability and correctness [1, 15].

It is assumed that the contracts of a system is de-facto a crosscutting concern that can be better modularized by the use of aspect-orientation [8, 10, 11]. Recent studies [10, 8, 1, 15, 16] have shown that object-oriented abstractions are not able to modularize the main features of design by contract methodology, such as invariants and pre- and postconditions, and tend to lead to programs with poor modularity (scattered and tangled DbC code).

To the best of our knowledge, Lippert and Lopes [10] conducted the most well-known systematic study that explicitly investigated the use of AO to implement classical design by contract features such as pre- and postconditions of a large OO framework, called JWAM. Among other things, they compared the contracted Java and AspectJ implementations of such OO framework. According to their findings, the AspectJ implementation improved the modularity of design by contract concern. Also, they argue that the use of AO drastically reduced the number of contracts (e.g., precondition) and lines of code (LOC). However, the authors presented their findings in terms of a qualitative as-

sessment. Quantitative evaluation consisted solely of counting LOC. Hence, there is no empirical evidence that AO techniques promote a superior solution in well-understood modularity attributes such as separation of concerns, coupling, and size, when used for modularizing non-trivial homogeneous and heterogeneous design by contract code.

This paper complements Lippert and Lopes work [10] by performing quantitative assessments of OO and AO implementations for invariants and pre- and postconditions of a real-life web-based information system, called Health Watcher (HW) [6]. The OO version was implemented in Java, whereas the AO version was implemented in AspectJ. Our evaluation focused upon on well-known modularity attributes such as separation of concerns, coupling, and size [17, 5]. We have found that the AO solution improved the separation of design by contract concern of the HW system. Moreover, the use of aspects have exhibited significant reuse of DbC features such as preconditions. However, the AO implementation of HW has not presented significant gains regarding four classical size metrics.

This paper is structured as follows: Section 2 describes our experimental settings and justifies the decisions made to ensure the study is valid. The results gathered from applying the modularity metrics are discussed in Section 3. Section 4 analyzes the obtained results and points some constraints on the validity of our study. Finally, Sections 5 concludes this paper by summarizing this paper's findings.

2 Experimental Settings

This section describes the configuration of our study. Section 2.1 briefly exemplifies and explains how we moved design by contract code to aspects. The choice of the target system is discussed in Section 2.2. In addition, the metrics used in the assessment process (Section 2.3), and our assessment procedures (Section 2.4) are described.

2.1 Aspectizing Design By Contract

Our study focused on the placement of contracts. We moved all the JC.requires, JC.ensures, and JC.invariant calls in the selected portions of the selected target system to aspects. These methods are declared in the JC class which encapsulate all the Java contract operations. As such, the methods calls JC.requires, JC.ensures, and invariant denotes pre- and postconditions, and invariants of the target system, respectively.

We used the Extract Fragment to Advice [13] refactoring to move contracts to aspects. Figure 1 illustrates this mechanics. It shows a trivial example of aspectization of preconditions using a **before** advice. Note that since the two methods of the class C have the same precondition α, we were able to refactor it to single advice, hence exploring

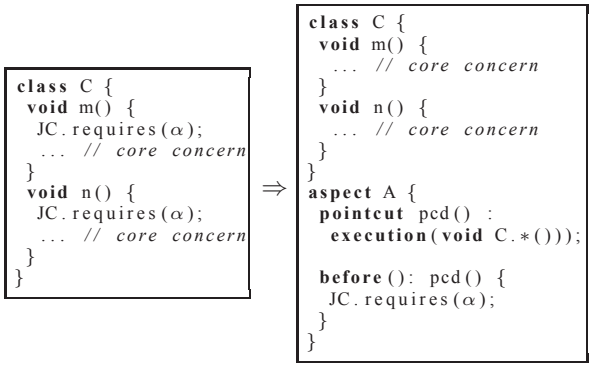


Figure 1. Refactoring DbC code to aspects.

reuse opportunities. Due to space constraints, we do not show how we extracted other DbC features to advice.

2.2 Target System Selection

The first major decision we had in our investigation was the selection of the target system. The chosen system is a real web-based information system, called Health Watcher (HW) [6]. The main purpose of the HW system is to allow citizens to register complaints regarding health issues. This system was selected because it met a number of relevant criteria for our intended evaluation. First, it is a real and non-trivial system with available OO and AO implementations with a number of recurring concerns and technologies common in day-to-day software development, such as GUI, persistence, concurrency, RMI, Servlets and JDBC [6]. Second, the original implementation of HW is composed by eleven use cases that are detailed described by an available requirements document, which is essential to understand its main functionalities [6]. Third, other qualitative and quantitative studies of the HW system have been recently conducted [9, 4, 6, 3], and so provided a solid foundation for this study.

2.3 The Metrics

In our study, a suite of metrics for separation of concerns (SoC), coupling, and size [17, 5] were selected to evaluate the OO and AO implementation versions of the HW system. This suite was adapted from classic OO metrics [2] to be applied to the AO paradigm. In addition, the chosen metrics have already been used in several empirical case studies [5, 9, 4, 6, 3]. For all the employed metrics, a lower value implies better results. Table 1 summarizes each metric used in this case study, and associates it with the relevant modularity attribute.

Separation of Concerns (SoC) metrics measure the degree to which a single concern (design by contract in our

Table 1. The Metrics Suite.

Attributes	Metrics	Definitions
Separation of Concerns (SoC)	Concern Diffusion over Components (CDC)	Number of classes and aspects that contribute to the implementation of a concern [5].
	Concern Diffusion over Operations (CDO)	Number of methods and advice that contribute to a concern's implementation [5].
	Concern Diffusion over LOC (CDLOC)	Counts the number of transition points for each concern through the lines of code. Transition points are points in the code where there is a "concern switch" [5].
Coupling	Coupling Between Components (CBC)	Number of classes and aspects declaring methods or fields that may be called or accessed by other components [2].
Size	Lines of Code (LOC)	Number of lines of code [2].
	Design By Contract Lines of Code (DbCLOC)	Number of lines of code that are relative to DbC.
	Number of Preconditions (NOPre)	Number of preconditions of each class or aspect.
	Number of Postconditions (NOPo)	Number of postconditions of each class or aspect.
	Number of Invariants (NOI)	Number of invariants of each class or aspect.
	Number of Attributes (NOA)	Number of attributes of each class or aspect [2].
	Number of Operations (NOO)	Number of methods and advice of each class or aspect [2].
	Vocabulary Size (VS)	Number of components of the system [2].

study) affects the system. The coupling metric CBC indicates the degree of dependency between components. Excessive coupling is not desirable, since it is detrimental to modular design. Size metrics are important to evaluate the complexity and different perspectives of the final system. In this way, the metric group includes metrics for both general system attributes (e.g., Number of Lines of Code) and quantities that are specific to design by contract such as Number of Preconditions (NOPre). The size metrics related to DbC are useful to quantify reuse of design by contract code in refactored versions of a particular target system. For further details about SoC, CBC, and size metrics, refer to [2, 17, 5].

2.4 Assessment Procedures

The main goal of this empirical case study is to answer how the HW system behaves regarding design by contract modularity when implemented with AO techniques. To this end, our study was divided into three major phases: (i) the implementation of the design by contract concern and alignment of the original HW according to its requirements document; (ii) the refactoring of the design by contract concern (developed in phase i) of HW to aspects, and (iii) the assessment of the two versions (the OO and AO versions developed in phases i and ii, respectively) of HW system.

In the first phase, we implemented the design by contract concern for the OO solution of the HW system, which is already available and implemented in Java. As aforementioned, HW comprises several classical crosscutting concerns, but no existing quantitative work have explored the design by contract one. We analyzed the entire available requirements document of the HW system to understand its functionalities and involved actors. This was fundamental to apply required preconditions, postcondition, and invariants for all the HW use cases. The implementation comprehends both homogeneous and heterogeneous contracts for the HW use cases. We found some inconsistences of the original HW implementation by its validation with contracts. Since this task is out of scope, we just mention that we made an alignment (fixing the found bugs) of the HW implementation to fulfil its requirements.

The second phase involved the refactoring of the design by contract crosscutting concern of HW system to aspects. After extracting all the contracts to aspects, we looked for reuse opportunities and eliminated identical contracts already moved to AspectJ advice. Basically, we implemented contracts in the aspects using before and after advice. Eventually, when we have *old expressions*, which refers to both pre- and post-state of a method execution, we used around advice. Further details on how to instrument old expressions with around advice, refer to [15, 16].

The goal of the third phase was to compare in a quantitative way the OO and AO versions of the HW system performed in the previous phases. In the measurement process, the data was partially gathered by the AJATO measurement tool [1]. It supports some metrics: LOC, NOA, NOO. Additionally, we used the AOP metrics tool [2] to collect CBC, LCOO, and VS. Eventually, we collected the SoC metrics (CDC, CDO, CDLOC) [17, 5] manually.

The data collection of SoC metrics (CDC, CDO and CDLOC) was preceded by what the metrics' authors call as "shadowing" process. In this process, the code implementing DbC was identified and shadowed in every class, interface and aspect. The metrics were, then, manually computed based on the shadowed code. The complete description of the gathered data, measurement tools, and shadowed code is available at [14]

[1] http://www.teccomm.les.inf.puc-rio.br/emagno/ajato/
[2] http://aopmetrics.tigris.org/

Table 3. Coupling and Size Metrics.

Metric			CBC	LOC	DbCLOC	NOPre	NOPo	NOI	NOA	NOO	VS
Before Refactoring	**OO**	Classes	342	7376	1272	467	346	383	183	530	89
		Aspects	-	-	-	-	-	-	-	-	-
		Total	**342**	**7376**	**1272**	**467**	**346**	**383**	**183**	**530**	**89**
After Refactoring	**AO**	Classes	279	6084	26	0	0	0	183	530	89
		Aspects	42	1150	1150	79	173	36	0	192	11
		Total	**321**	**7234**	**1176**	**79**	**173**	**36**	**183**	**722**	**100**
		Diff.	**-6.14%**	**-1.92%**	**-7.54%**	**-83.08%**	**-50%**	**-90.60%**	**0%**	**+26.59%**	**+11%**

Table 2. Separation of Concerns Metrics.

Metric			CDC	CDO	CDLOC
Before Refactoring	**OO**	Classes	51	297	1193
		Aspects	-	-	-
		Total	**51**	**297**	**1193**
After Refactoring	**AO**	Classes	1	4	0
		Aspects	11	192	0
		Total	**12**	**196**	**0**
		Diff.	**-76.47%**	**-34%**	**-100%**

3 Study Results

This section presents the results of the measurement process. The data have been collected based on the set of defined metrics (Table 1). We present the results by means of tables. Rows labelled "Diff." indicate the percentage difference between the original and refactored versions of the HW. A positive value means that the OO version fared better, whereas a negative value indicates that the AO version exhibited better results.

3.1 Quantifying Separation of Concerns

Table 2 shows the obtained results of the separation of concern metrics. The "Diff." row shows significant differences in favor of the AO implementation in terms of the Concern Diffusion over Components (CDC). This divergence is a direct consequence of the adopted strategy for creating new DbC aspects in HW system. For example, we created a new aspect whose sole responsibility was to implement all the non-null input parameters required to fulfil the HW requirements. This classical contract checking denotes an example of homogenous contracts of the HW system. In cases when we had heterogenous contracts, we create one aspect per layer to encapsulate the HW contracts. This scenario contributed to a better result in favor of the AO implementation regarding the CDC metric. The DbC concern is spread over 51 components (classes or interfaces) in the OO version, whereas in the AO solution it was only 12 components (in which 11 are aspects). This led us to a percentage reduction of 76.47% in favor of AO version.

Still regarding the AO solution, the DbC concern fared better for the CDO metric. It is scattered over 297 in the OO solution against to only 196 operations relative to the AO solution. As a result, we had a 34% percentage of reduction in favor to AO solution. Finally, the Concern Diffusion over LOC (CDLOC) was the metric where the AO implementation of HW system performed better against its counterpart in OO implementation. This implies that the design by contract concern is more tangled in the OO solution than in the AO implementation. The OO solution presents 1193 "concern switches" over the system code, whereas the AO solution had no occurrence of concern switches. This means that the DbC concern was completely untangled, localized, and isolated with aspects.

3.2 Quantifying Coupling and Size

Table 3 shows the obtained results for the coupling and size metrics. Regarding coupling, as observed, there is a small difference in favor of the AO implementation of HW system. Aspects reduced the coupling between system classes by removing the DbC-related code from them. However, the aspects still need to reference and, thus, are coupled to classes on which they introduce the DbC behavior. Hence, we had only 6.14% percentage reduction in favor of AO solution (see the CBC column in Table 3).

Contradicting the general intuition that aspects make programs smaller [10, 8] due to reuse, the OO version and its counterpart in AO did not present significant gains in relation to the four classical metrics: VS, NOO, LOC, and NOA. For instance, the Vocabulary Size (VS) grew as expected with 11% more components (classes + aspects) due to the introduction of design by contract aspects. Thus, AO version involved 100 components (with 11 created aspects), whereas the OO implementation included only 89 components to comprise the same functionalities. Moreover, the Number of Operations (NOO) grew significantly in the AO version due to the modularization of DbC with new mechanisms such as advice. As a result, we had 26.59% more method-like definitions in the AO version. In the HW system, the difference of the number of LOC was only 1.92% in favor to AO solution. Hence, even with significant reuse of design by contract code as we discuss next, the aspect code for realizing the DbC concern requires a lot of extra idioms which led to extra effort during implementation.

This finding, contradicts the Lipert and Lopes study [10], on which they had a reduction of more than 50% in LOC due to reuse. This happens because while HW system has homogeneous contracts that can be significantly reused, some heterogenous contracts can be harmful to the final LOC due to the poor reuse of such heterogeneous contracts and the extra aspect code needed to "aspectize" the design by contract concern. Finally, we had no difference between the two versions in relation to the Number of Attributes (NOA).

As aforementioned, in order to quantify the reuse of the design by contract code, we employed specific metrics to DbC concern: DbCLOC, NOPre, NOPo, NOI. The Number of LOC relative to DbC (DbCLOC) was the only specific size metric in which the benefits by the AO solution was less than 10% (7.54%). This happens because the DbC aspects used to modularize the HW contracts (e.g., invariants) have significant extra code (such as pointcuts to intercept the join points involved in the DbC concern realization and advice contextual information) to cope with the crosscutting concern modularization.

The remaining three DbC size metrics performed significantly better in favor to AO solution. For instance, the Number of Invariants (NOI) exhibited a reuse of 90.60%. This is understandable since an invariant implementation in Java needs to be invoked several times to fulfil its semantics. An invariant holds after every constructor execution and just before and after every instance method execution of a class [15]. With AO and AspectJ, we can modularize an invariant with two advice reducing the so scattered implementation of invariant calls to only 2 occurrences [15].

In relation to pre- and postconditions, we had a higher reuse of preconditions in the AO implementation (83.08%). Regarding postconditions, the AO solution fared better with a reuse of 50% against its counterpart in OO implementation. In fact, these numbers can substantially vary depending the degree of homogeneous and heterogeneous pre- and postconditions that can appear in a particular system. In the HW system, we observed that the postconditions are less reusable than preconditions, due to postconditions present more heterogenous contracts.

4 Discussion

This section makes a qualitative analysis of the obtained results (Section 3). Furthermore, we discuss the constraints on the validity of our empirical case study.

4.1 Empirical findings

Our empirical case study confirms some of the findings of the qualitative study conducted by Lippert and Lopes (LL) [10], which claims that the design by contract concern is better modularized with AO programming.

Despite the fact we had significant gains of SoC metrics in favor of the AO implementation of HW system, we found out that reusing contracts can be more difficult than usually advertised [10]. Contracts reuse depends directly on their types (e.g., postconditions) and mainly if such contracts are homogeneous or heterogenous.

For instance, since the nature of an invariant crosscuts several methods in a single class, it is naturally more reusable than pre- and postconditions that are relative to particular methods. However, pre- and postconditions can present significant reuse depends on their contracts. In other words, if several constrained methods present an intersection of common contracts, their reusability can be improved. As an example, similarly to LL [10], we found that several methods in HW present the following homogeneous postcondition: `JC.ensures( result != null)`. This postcondition states that every method using this contract must return an object that is non-null. The same situation also occurred for preconditions on input object parameters. In the HW system we observed that the reuse of postconditions was quite low (50%) when compared with preconditions (83.08%) and invariants (90.60%). This scenario happens due to postconditions in HW being more heterogenous than preconditions or invariants. We found more reusability opportunities for heterogenous preconditions than the heterogeneous postconditions. With this finding, we can conclude that the more heterogenous is a contract, its reuse with AO programming is minimized.

Another important finding of our study is related to the program size after refactoring to aspects. LL [10] discuss that by using AO programming they could reduce more than 50% of the total design by contract LOC due to the reuse. However, contradicting this general intuition that aspects make programs considerable smaller, we found that despite the higher reuse of DbC concern with the AO version of HW system, the gains in terms of the overall system LOC was only 1.92% and only 7.54% considering exclusively the LOC of DbC concern. This was a directly consequence for looking to reusability opportunities for heterogenous contracts. During modularization, we take into account the intercepted join points, contextual information and so forth.

4.2 Study Constraints

System. Although it can be argued that using a single system for such a study is a limiting factor, we claim that the HW system is representative in terms of the non-trivial applied contracts. The HW system is good candidate for empirical studies due to have a lot of documentation and resources available [6]. Naturally it is desirable to involve more systems and more approaches.

Metrics. The applicability, usefulness, and representative of the set of the metrics used in this study can be ques-

tioned. However, due to the nature of the study and the fact that separation of concerns is central to this study, the design by contract crosscutting concern was naturally the one which varied most. Hence, we used a set of metrics related to separation of concerns to better assess the SoC involving DbC. In addition, the SoC metrics described in Section 2.3 have already been proved to be effective quality indicators in several case studies [5, 9, 4, 6, 3].

Languages. In addition, the scope of our experience is limited to Java and AspectJ languages. In relation to design by contract features, our experience only considered the implementation of pre-, postconditions, and invariants.

5 Concluding Remarks

In this paper, we presented an empirical case study to assess various facets of design by contract modularity of object-oriented and aspect-oriented implementations of a real-life system. This study was the first to include a quantitative analysis of design by contract implementations using well-undertood and experimented modularity metrics such as separation of concerns, coupling, and size.

From this analysis we have discovered a number of interesting outcomes. Firstly, the use of aspect-orientation to modularize design by contract improved the separation of concerns when compared to its counterpart in object-orientation. Secondly, the use of aspects tended to present a significant reuse of design by contract features, specially invariants, which we had a reuse more than 90%.

Furthermore, even with the high reuse achieved by AO programming when modularizing DbC, we found out that the aspectization of crosscutting concern such as DbC does not necessarily makes a program drastically smaller than the non-modular one with OO. Aspect code involves much more than just encapsulate a call to a precondition. It is responsible to prepare the crosscutting behavior by intercept all the join points in a system, expose contextual information and so forth. The overall conclusion regarding design by contract modularity is that aspects achieve higher reuse than OO decompositions when handling homogenous contracts, however, not always with large code reduction.

One of the most immediate future work is to derive a predictive model for using aspects to implement design by contract, based on our experience of this study. Hence, developers may recognize the situations where it is advantageous to aspectize design by contract code. Moreover, we intend to conduct a scalability study to analyze how aspects scale up when the number of contracts grows.

Acknowledgements

This work is partially supported by INES, funded by CNPq and FACEPE, under grants 573964/2008-4 and APQ-1037-1.03/08. Ricardo Lima is also supported by CNPq under grant No. 314539/2009-3. Henrique Rebêlo is also supported by FACEPE under grant No. IBPG-1664-1.03/08. Cláudio Sant'Anna is also supported by CNPq under grant No. 480374/2009-0.

References

[1] L. C. Briand et al. Instrumenting contracts with aspect-oriented programming to increase observability and support debugging. In *Proc. of the 21st IEEE ICSM*, 2005.

[2] S. R. Chidamber and C. F. Kemerer. A metrics suite for object oriented design. *IEEE TSE.*, 20:476–493, June 1994.

[3] R. Coelho et al. Assessing the impact of aspects on exception flows: An exploratory study. In *Proceedings of the 22nd ECOOP*, Berlin, 2008. Springer-Verlag.

[4] F. C. Filho, A. Garcia, et al. Exceptions and aspects: the devil is in the details. In *SIGSOFT '06/FSE-14: Proceedings of the 14th ACM SIGSOFT FSE*, USA, 2006.

[5] A. Garcia et al. Modularizing Design Patterns with Aspects: A Quantitative Study. In *Proceedings of the 4th AOSD*, New York, NY, USA, March 2005. ACM Press.

[6] P. Greenwood et al. On the impact of aspectual decompositions on design stability: An empirical study. In *ECOOP*, pages 176–200, 2007.

[7] C. A. R. Hoare. An axiomatic basis for computer programming. *Commun. ACM*, 12(10):576–580, 1969.

[8] G. Kiczales et al. Getting started with aspectj. *Commun. ACM*, 44:59–65, October 2001.

[9] U. Kulesza et al. Quantifying the effects of aspect-oriented programming: A maintenance study. In *Proceedings of the 22nd IEEE ICSM*, USA, 2006.

[10] M. Lippert and C. V. Lopes. A study on exception detection and handling using aspect-oriented programming. In *Proceedings of the 22nd ICSE*, ICSE '00, USA, 2000. ACM.

[11] M. Marin et al. A classification of crosscutting concerns. In *Proceedings of the 21st IEEE ICSM*, pages 673–676, Washington, DC, USA, 2005. IEEE Computer Society.

[12] B. Meyer. Applying "design by contract". *Computer*, 25(10):40–51, 1992.

[13] M. P. Monteiro and J. a. M. Fernandes. Towards a catalog of aspect-oriented refactorings. In *AOSD'05*, 2005.

[14] H. Rebêlo et al. Design by contract as aspects. Available from: `http://cin.ufpe.br/~hemr/seke11`.

[15] H. Rebêlo et al. Implementing java modeling language contracts with aspectj. In *Proc. of the 2008 ACM SAC*, 2008.

[16] H. Rebêlo et al. The contract enforcement aspect pattern. In *Proceedings of the 8th SugarLoafPlop*, pages 99–114, 2010.

[17] C. Sant'anna et al. On the reuse and maintenance of aspect-oriented software: An assessment framework. In *Proceedings XVII SBES*, pages 19–34, Oct 2003.

A. Online Appendix

We invite researchers to replicate our case study. Source code of the OO and AO versions of the HW system, used measurement tools, shadowed code, and our results are available at [14].

Failure Prediction based on Log Files Using the Cox Proportional Hazard Model

Ilenia Fronza, Alberto Sillitti, Giancarlo Succi, Jelena Vlasenko
Free University of Bolzano - Bozen
Piazza Domenicani - Domenikanerplatz, 3,
I-39100 Bolzano - Bozen, Italy
{Ilenia.Fronza, Alberto.Sillitti, Giancarlo.Succi}@unibz.it, Jelena.Vlasenko@stud-inf.unibz.it

***Abstract*—Accurate failure predictions can help in mitigating the impact of computer failures. Resources, applications, and services can be scheduled to limit the impact of failures. However, providing accurate predictions with sufficient lead of time is challenging. Log files track changes of system state. A sequence or a pattern of messages may be used to predict failures. Here we describe an approach to predict failures based on the Cox Proportional Hazards (PH) model that has been applied successfully in various fields of research. We apply our approach to log files collected during approximately 3 months of work in a large Italian company. We compare the performance of the proposed model with Support Vector Machines.**

Keywords-failure prediction, Cox Proportional Hazard model, Log files

I. Introduction

The impact of failures can be substantial since the recovery process can require unexpected amounts of time and resources [1; 10]. Being able to forecast failures is extremely important, even when failures are inevitable – at least recovery or rescue actions can be taken [25].

Methods for the prediction of system failures based on events (in our case, the log messages) have been proposed in various engineering disciplines. These methods can be classified into design-based methods and data-driven rule-based methods. In a design-based method, the expected event sequence is obtained from the system design and is compared with the observed event sequence [19; 21; 24]. The major disadvantage of these methods is that in many cases events occur randomly and thus there is no system logic design information available.

Data-driven rule based methods do not require system logic design information. These methods are made of two phases: 1) identification of temporal patterns, i.e., sequences of events that frequently occur [16] and 2) development of prediction rules based on these patterns [14].

System log files consist of messages created while the system is running. The information recorded varies from general messages concerning user logins to more critical warnings about program failures [10]. Log file information can be analysed to find causes of failures. Although this type of forensic analysis is valuable, it is also possible to use the information contained in system log files for predicting events. There have been several approaches for predicting system failures using system log files.

Prediction methods using log files include standard machine learning techniques such as Support Vector Machines, Bayes networks, Hidden Markov Models, and Partially Observable Markov Decision Process [8; 9; 10; 11; 15; 20; 26; 28; 29]. The use of time-series analysis is common among these methods since a system message in isolation has been shown to be insufficient for predicting failure [29]. Unfortunaly, the large amount of information available in system log files makes finding the right pattern(s) difficult [10].

In this paper we introduce a new approach for predicting critical system events based on the Cox PH model. The actualization of such idea will set the path for the development of devices that read logs of running applications and signal the likely crash of such systems.

The Cox PH model has been applied mainly in biomedicine, often for the study of cancer survival [5; 31]. It has also been applied successfully in various fields of research, such as criminology [4], sociology [2], marketing [3]. There are limited uses of the Cox PH model in cybernetic [14] and also an application to software data [27] where, using as input code metrics, failure time data coming from bug report were analysed.

The paper is structured as follows: in Section 2, we present some background; in Section 3, we introduce our approach. In Section 4, we describe our experiments and results. In Section 5 we discuss our results.

II. Background

A. System Log Files

System log files are important for managing computer systems since they provide a history or audit trail of events. In this context, an event is a change in a system status, such as a user login or an application failure. Several studies have proposed different approaches for predicting system failures using system log files [8; 9; 10; 11; 14; 15; 20; 26; 28; 29].

System log files typically are text files that consist of messages sent to the logging service by applications. Applications can send information to the logging service process that stores the messages in a text file in an arrival

order. The logging service is primarily responsible for managing the log file while the message content is largely created by the application. In the dataset used in this study, each log entry consists of eight fields (Table I). The time field is the time when the message was recorded. One field stores the name of the application that was running; the machine sending the message is identified by two fields, server and computer (since the logging service serves multiple computers on different servers). The logged-in user is reported in the UsedName field, and Severity contains the level of severity of the message. In addition, some of the logs (not all), contain also an error name and description of the event that has occurred.

TABLE I. FIELDS OF A LOG MESSAGE IN THE ANALYSED DATASET.

Field Name	Description
Time	Time the message was recorded
Application	Running application
Server	Machine sending the message
Computer	
UserName	Name of the logged user
Severity	Level of severity of the message
Operation	Performed operation
Description	Some of the logs (not all) contain also the error name and the Description of the event that has occurred
ErrorName	

B. The Cox PH Model

Cox PH model (Cox 1972) gives an expression for the hazard at time *t* for an individual *i* with a given specification of *p* covariates *x*:

$$h(t|x) = h_0(t)e^{\sum_{i=1}^{p} \beta_i x_i} \quad (1)$$

The Cox model formula says that the hazard at time t is the product of two quantities. The first of them, $h_0(t)$, is called the baseline hazard function and is equal for all individuals; it may be considered as a starting version of the hazard function, prior to considering any of the x´s. Cox PH model focuses on estimating regression coefficients *ß´*s leaving the baseline hazard unspecified. *ß* is a vector of regression coefficients; in the p < n setting, *ß´*s are estimated by maximizing the log partial likelihood, which is given by:

$$l(\beta) = \sum_{i=1}^{n} \delta_i \left\{ x_i \beta - \log\left[\sum_{j \in R(t_i)} e_i^x \beta \right] \right\} \quad (2)$$

Where $R(t_i)$ is the risk set at time t_i, i.e. the set of all individuals who are still under study just prior to time t_i.

A parametric survival model is one in which survival time (the outcome) is assumed to follow a known distribution. The Cox PH model is not a fully parametric model; rather it is a semi-parametric model because even if the regression parameters ß´s are known, the distribution of the outcome remains unknown. The Cox PH model is a "robust" model, since the results obtained from it closely approximate the results of the correct parametric model.

The key assumption of the Cox PH model is proportional hazards; this assumption means that the hazard ratio (defined as the hazard for one individual over the hazard for a different individual) is constant over time.

Cox PH model is widely used because of its characteristics: 1) even without specifying $h_0(t)$, it is possible to find the *ß´*s, 2) no particular form of probability distribution is assumed for survival times, and 3) it uses more information – the survival times – than the logistic model, which considers a (0,1) outcome and ignores survival times and censoring. Therefore, it is preferred over the logistic model when survival time is available and there is censoring [13].

C. Performance Assessment

A contingency table (sometimes called confusion matrix) is a convenient way to tabulate statistics for evaluating the quality of a model. In Table II, TP, FP, TN and FN stand for true/false positive/negative counts, respectively; PP and PN stand for predicted positive/negative; and POS and NEG stand for actual positive/negative.

In this paper we will only consider metrics that can be defined in terms of the counts in a contingency table; this excludes, e.g., metrics that consider model complexity [7]. The most relevant metrics of this type are reported in Table III.

TABLE II. CONTINGENCY TABLE (A.K.A., CONFUSION MATRIX).

		Actual Value	
		Pos	***Neg***
Prediction Outcome	***PP***	TP	FP
	PN	FN	TN
PP=Predicted Positive, PN=Predicted Negative, Pos=Actual Positive, Neg=Actual Negative, TP=True Positive, FP=False Positive, TN=True Negative, FN=False Negative			

TABLE III. DERIVATIONS FROM A CONFUSION MATRIX: MOST RELEVANT METRICS FOR CLASSIFICATION PERFORMANCE EVALUATION [17].

Name *(alternative names)*	Definition
True Positive Rate *(TPR, Sensitivity, Recall)*	$\frac{TP}{TP+FN}$
False Positive Rate *(FPR, fall out)*	$\frac{FP}{FP+TN}$
Accuracy	$\frac{TP+TN}{TP+TN+FP+FN}$
True Negative Rate *(TNR, Specificity)*	$\frac{TN}{TN+FP}$
Balance	$1-\frac{\sqrt{(0-FPR)^2+(1-TPR)^2}}{\sqrt{2}}$

We use Receiver Operating Characteristics (ROC) space to analyse the classifier performance. False positive rate is plotted against the true positive rate. The graph (Fig. 1) shows

the trade-off benefits (TP) and costs (FP). The points (0,0) and (1,1) represent the training-free classifiers *AlwaysNegative* and *AlwaysPositive*; the point (0,1) represents the ideal classifier, and (1,0) represents the classifier which gets it all wrong. The ascending diagonal (0,0)–(1,1) represents random training-free behaviour: any point (*p*,*p*) can be obtained by predicting positive with probability *p* and negative with probability (1–*p*). The upper left triangle contains classifiers that perform better than random, while the lower right triangle contains those performing worse than random. The descending diagonal (0,1)–(1,0) represents classifiers that perform equally well on both classes (*TPR*=1–*FP*); left of this line we find classifiers that perform better on the negatives than the positives, to the right performance on the positives dominates [7].

Balance is defined in [17] as the Euclidean distance from the sweet spot FPR= 0, TPR=1 to a pair of (FPR; TPR) . For convenience, we 1) normalize balance by the maximum possible distance across the ROC square ($\sqrt{2}$), and 2) subtract this from 1, i.e. Hence, better and higher balances fall closer to the desired sweet spot of FPR= 0, TPR=1. In particular, classifiers having balance higher than 0.5 fall in the upper left triangle of the ROC space. Thus, we consider 0.5 as a threshold for balance.

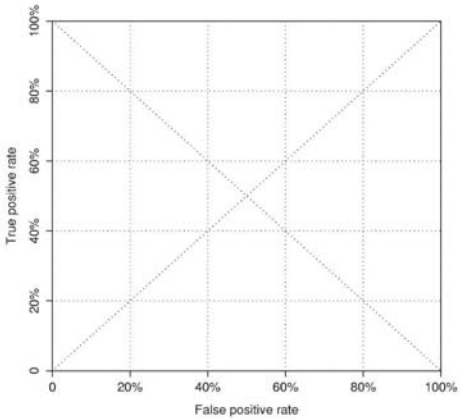


Figure 1. ROC space structure (from [7]).

Ideally, we seek predictors that maximize accuracy, TPR, and TNR. Note that maximizing any one of these does not imply high values for the others. Accuracy is a good measure of a learner's performance when the possible outcomes occur with similar frequencies, and is not suggested when class distributions are uneven [17]. Therefore, this paper will assess its learned predictors using balance, TPR, and FPR and not accuracy.

III. Approach

A. Structure of the approach

We propose a technique to predict the failure of a running software systems using log files. The idea is to develop devices that read logs of running applications and signal the likely crash of such systems. Fig. 2 presents a schematic view of the proposed approach. While the system is running, log data are collected to track the actual execution path. In this work, we look at the running system as a "black box", meaning that we do not have any other information about the system except the log files.

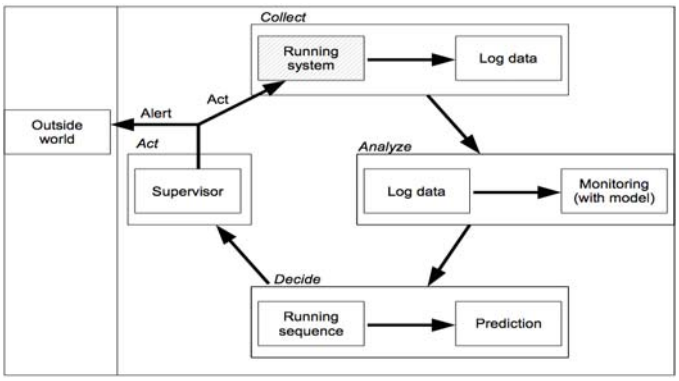


Figure 2. Schema of the proposed approach.

The monitoring process takes log data as input and, basing on the analysis performed, gives to the supervisor a message indicating the "likely failure" for the running application. The supervisor can act directly on the running system to avoid the predicted failure, or send an alert to the outside world. Possible actions could be to abort the running system, to restart it, to dynamically load components, or to inform the running system if it was a suitably structured autonomic system [18].

B. Structure of the Monitoring Process

1) Dimensional reduction of the problem and data preparation

While the system is running, data are *collected* [6; 22; 23] to track the actual execution path. Then, to get from raw logs to temporal event sequences:

1. data are parsed to extract Operation, Time and Severity fields;
2. duplicate rows are deleted together with logs that have missing information in one or several fields Operation, Time stamp, Severity;
3. sequences of activities are extracted: a new sequence starts either if there is a 'Log in' operation or if the day changes.
4. for each sequence, the number of occurrences of each operation is found.

Failures are defined as sequences containing at least one severity "Error".

2) Model Training and Analysis of the results

Following the guidelines of [12; 30; 31], model training includes the following steps:

1. the Schoenfeld test [13] is applied to select the operations satisfying the PH assumption.
2. the Cox PH model is applied and operations that are significantly associated to failures are identified.

3. for each sequence a risk score is evaluated according to the exponential value of a linear combination of the multiplicity of the operation, weighted by the regression coefficients derived from the aforementioned Cox PH model.
4. *m* is extracted, as the third quartile of risk scores of non failure sequences;
5. The risk score RS is then evaluated for the actual sequence of the running application, as in point 4. A message of "likely failure" about the running application is given as output to the supervisor when RS ≥ m.

IV. Experiments

A. Data

We use log files collected during approximately 3 months of work in an important Italian company that prefers to remain anonymous. Table IV presents some descriptive statistics about the six application in the dataset.

TABLE IV. Descriptive statistics.

App	Number of sequences	Pos* (%)
A1	12765	0.84
A2	718	15.88
A3	60	23.33
A4	343	12.83
A5	713	4.77
A6	8593	1.23
Pos = failures (percentage over n)		

After the preprocessing phase, sequences were sampled 1000 times using Monte carlo methods to obtainn training sets (60%) and test sets (40%). The proposed method was then applied as explained in Section III.

To compare the performance of the proposed model with SVMs, we applied SVM with linear, polynomial and radial basis kernel. The cost of misclassifying points was 100 in each case, to force the creation of a more accurate model.

B. Results

Table V shows the results of the proposed approach and compares its performance with the best performing SVM.

Our approach shows TNR higher than 0.70 in five cases. SVMs have almost perfect TNR: data are imbalanced, thus SVMs classify almost all instances as negative. Some specificity (i.e., TNR) is lost in our approach to improve the TPR.

TABLE V. Classification performance.

App	Classification Performance							
	TNR mean(std)		*TPR mean (std)*		*FPR mean (std)*		*Balance mean (std)*	
	Cox	*SVM*	*Cox*	*SVM*	*Cox*	*SVM*	*Cox*	*SVM*
A1	0.49 (0.25)	1.00** (0.00)	0.52 (0.20)	0.00** (0.00)	0.51 (0.25)	0.00** (0.00)	0.46 (0.03)	0.29** (0.00)
A2	0.75 (0.12)	0.98** (0.01)	0.97 (0.08)	0.91** (0.03)	0.25 (0.12)	0.02** (0.01)	0.82 (0.08)	0.93** (0.02)
A3	0.72 (0.14)	0.98* (0.05)	0.05 (0.09)	0.04* (0.09)	0.28 (0.14)	0.02* (0.05)	0.29 (0.07)	0.32* (0.07)
A4	0.70 (0.12)	0.96** (0.02)	0.38 (0.19)	0.33** (0.11)	0.31 (0.12)	0.04** (0.02)	0.50 (0.11)	0.52** (0.08)*
A5	0.93 (0.13)	0.99* (0.01)	0.86 (0.11)	0.73* (0.18)	0.07 (0.13)	0.01* (0.01)	0.88 (0.11)	0.81* (0.13)
A6	0.79 (0.16)	0.99* (0.01)	0.78 (0.18)	0.51* (0.12)	0.21 (0.16)	0.01* (0.01)	0.76 (0.14)	0.65* (0.08)
* Linear kernel; ** Radial kernel								

Fig. 4 show the ROC space of the proposed approach. The mean values of TPR and FPR of the 1000 experiments are plotted together with their standard deviation. Five points fall in the upper left triangle; thus the corresponding classifiers perform better than random. Three points are left of the descending diagonal (0,1)–(1,0); thus, these three classifiers perform better on the negatives than on the positives. It emerges from the comparison of this plot to Fig. 3 that A1 and A3 have now left the (0,0) point and approach the (0,1) point.

Fig. 5 shows that balance obtained with the proposed approach is higher than 0.50 in four out six cases, as for SVMs. Balance increases in three cases when applying our approach (in return of the above mentioned slight loss in TNR).

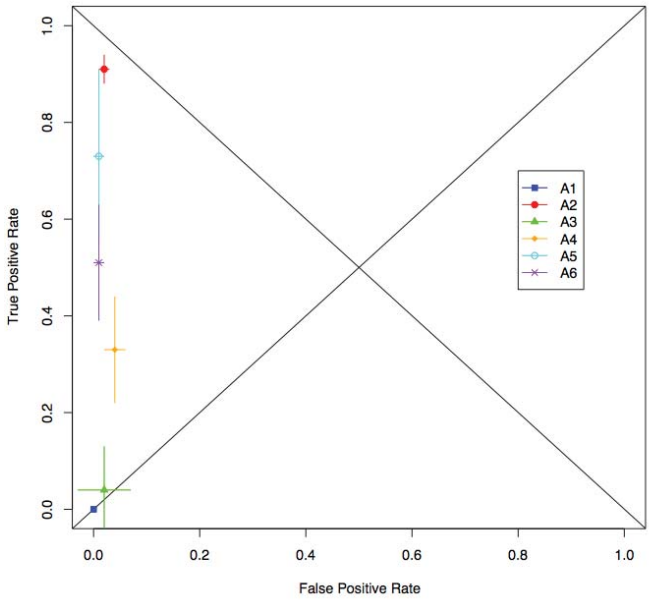


Figure 3. ROC space of the best performing SVMs.

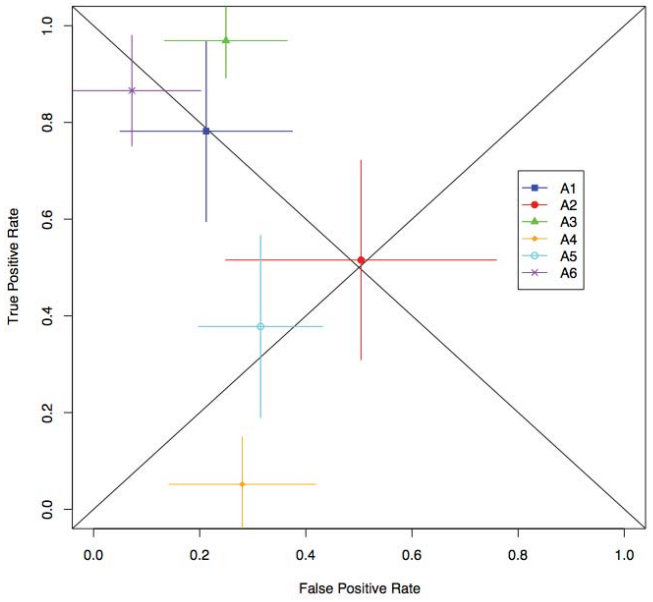


Figure 4. ROC space of the proposed approach based on the Cox PH model.

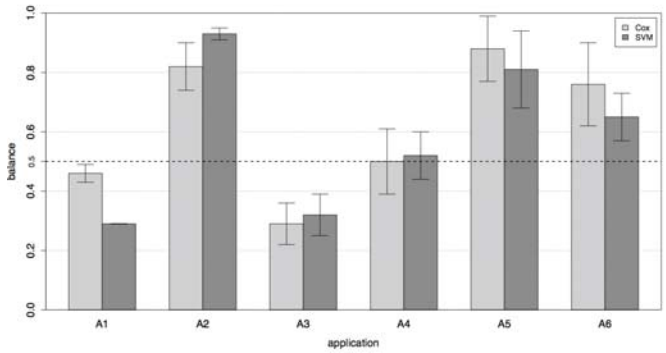


Figure 5. Balance of the proposed approach based on the Cox PH model and of SVMs: a comparison.

V. Conclusions

In this paper we introduce a new approach for predicting failures of a running system using log files; in particular, our approach is based on the Cox PH model. Each sequence of operations is assigned a risk score evaluated according to the exponential value of a linear combination of the multiplicity of the operation, weighted by the regression coefficients derived from the aforementioned Cox PH model. Classification as failure/non-failure is based on this risk score.

It emerges from the comparison with SVMs performance that the proposed approach increases balance between TPR and FPR in three cases. This result is gained in return of a slight loss in TNR which is almost perfect in SVMs. TNR may be so high because data are imbalanced; thus SVMs tends to classify everything as negative. Our approach improves on TPR inevitably sacrificing some specificity. Moreover, the approach based on Cox model gives good performances in the cases where SVMs give their worst results.

Overall, results from our experiments appear interesting and worth further investigations. Our goal now is to study more in-depth our promising model to determine if we can generalize our results. To this end, we plan to replicate the analysis on more industrial datasets. We will also work on broader comparison with other prediction methods to identify the specialities of our approach.

Another aspect that we will evaluate is the possibility of predicting the occurrence of a failure analysing only an initial portion of a sequence, so that there could be an early estimation of failure, providing additional time to take corrective actions. To this end, Cox PH model could be applied to predict survival and the estimate hazard ratio for the various features of the model may contribute to predict the "most dangerous" failures.

Finally, we are now investigating how we could consider other “black-box” properties or applications to predict failures; candidate properties include memory usage, number of open files, processor usage.

The proposed model could be particularly useful dealing with autonomic systems. Autonomic systems could be instructed to receive signals of likely failures and upon reception of such signals could start a suitable recovery procedure [18].

Acknowledgments

We acknowledge gratefully the support of the Free University of Bolzano/Bozen, and of the Province of South Tyrol.

References

[1] Adiga, N., et al. 2002. An Overview of the bluegene/l supercomputer. In *Proceedings of Supercomputing*, 2002.

[2] Agerbo, E. 2007. High income, employment, postgraduate education, and marriage : a suicidal cocktail among psychiatric patients. *Archives of General Psychiatry*, 64, 12, 2007, 1377-1384.

[3] Barros, C.P. and Machado, L.P. 2010. The length of stay in tourism. *Annals of Tourism Research*, 37, 3, 2010, 692-706.

[4] Benda, B. 2005. Gender differences in life-course theory of recidivism: A survival analysis. *International Journal of Offender Therapy and Comparative Criminology*, 49, 3, 2005, 325-342.

[5] Bøvelstad, H.M., Nygård, S., Størvold, H.L., Aldrin, M., Borgan, Ø., Frigessi, A., and Lingjærde, O.C. 2007. Predicting survival from microarray data a comparative study. *Bioinformatics*, 23, 16, 2080–2087.

[6] Coman, I.D., Sillitti, A., and Succi, G.: A case-study on using an Automated In-process Software Engineering Measurement and Analysis system in an industrial environment. In *Proceedings of the International Conference on Software Engineering* (Vancouver, Canada, May 16-24 May, 2009).

[7] P.A. Flach. 2003. The geometry of ROC space: understanding machine learning metrics through ROC isometrics. In *Proceedings of the 20th International Conference on Machine Learning*, 2003.

[8] Fu, S. and Xu, C.Z. 2007. Quantifying temporal and spatial fault event correlation for proactive failure management. In *Proceedings of Symposium on Reliable and Distributed Systems*, 2007.

[9] Fu, S. and Xu, C.Z. 2007. Exploring event correlation for failure prediction in coalitions of clusters. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis* (Reno, NV, USA, Nov. 15-21, 2007).

[10] Fulp, E.W., Fink, G.A., and Haack, J.N. 2008. Predicting computer system failures using support vector machines. In *Proceedings of the 1st Conference on Analysis of system logs*, 2008.

[11] Gross, K.C., Bhardwaj, V., and Bickford, R. 2002. Proactive Detection of Software Aging Mechanisms in Performance Critical Computers. In *Proceedings of the 27th Annual NASA Goddard Software Engineering Workshop*, 2002.

[12] Hao, K., et al. 2009. Predicting prognosis in hepatocellular carcinoma after curative surgery with common clinicopathologic parameters. *BMC Cancer*, 9, 2009, 398-400.

[13] Kalbfleisch, J.D. and Prentice, R.L. 2002. *The statistical analysis of failure time data*. Wiley, 2nd edition.

[14] Li, Z., Zhou, S., Choubey, S., and Sievenpiper, C. 2007. Failure event prediction using the Cox proportional hazard model driven by frequent failure sequences. *IEE Transactions*, 39, 3, 2007, 303-315.

[15] Liang, Y., Zhang, Y., Xiong, H., and Sahoo, R. 2007. Failure prediction in ibm bluegene/l event logs. In *Proceedings of the International Conference on Data Mining*, 2007.

[16] Mannila, H., Toinoven, H., and Verkamo, A.I. 1997. Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery*, 1, 1997, pp. 259 – 289.

[17] Menzies, T., Greenwald, J., and Frank, A. Data Mining Static Code Attributes to Learn Defect Predictors. *IEEE Transactions on Software Engineering*, 33, 1, 2007, 2-13.

[18] Müller, H.A., Kienle, H.M., Stege, U. 2009 Autonomic Computing: Now You See It, Now You Don't—Design and Evolution of Autonomic Software Systems. In: De Lucia, A., Ferrucci, F. (eds.): Software Engineering International Summer School Lectures: University of Salerno, LNCS 5413, Springer-Verlag, 32–54.

[19] Pandalai, D.N. and Holloway, L.E. 2000. Template languages for fault monitoring of timed discrete event processes. *IEEE Transactions on Automatic Control*, 45, 5, 2000, pp. 868 – 882.

[20] Salfner, F. 2008. *Event-based Failure Prediction: An Extended Hidden Markov Model Approach*. Berlin, Germany: Dissertation, Verlag.

[21] Sampath, M., Sengupta, R., and Lafortune, S. 1994. Diagnosability of discrete event systems. In *Proceeding of the 11th international conference on Analysis and Optimization of Systems Discrete Event System*s (Sophia, Antipolis, June 15 – 17, 1994).

[22] Sillitti A., Janes A., Succi G., and Vernazza T. 2004. Measures for Mobile Users: an Architecture. *Journal of Systems Architecture*, 50, 7, 393 - 405.

[23] Scotto M., Sillitti A., Succi G., and Vernazza T. 2006. A Non-Invasive Approach to Product Metrics Collection. *Journal of Systems Architecture*, 52, 11, 668-675.

[24] Srinivasan, V.S. and Jafari, M.A. 1993. Fault detection/monitoring using time petri nets. IEEE Transactions on System, Man and Cybernetics, 23, 4, 1993, 1155 – 1162.

[25] Schroeder, B. and Gibson, G.A.Understanding failures in petascale computers. *Journal of Physics*, 78, 2007.

[26] Stearley, J. and Oliner, A.J. Bad words: Finding faults in Spirit`s syslogs. In *Proceedings of the International Symposium on Cluster Computing and the Grid* (Lyon, France, May 19-22, 2008).

[27] Wedel, M., Jensen, U., and Göhner, P. 2008. Mining software code repositories and bug databases using survival analysis models. In *Proceedings of the 2nd ACM-IEEE international symposium on Empirical software engineering and measurement* (Kaiserslautern, Germany, October 09 - 10, 2008).

[28] Xue, Z., Dong, X., Ma, S., and Dong, W. A survey on failure prediction of large-scale server clusters. In *Proceedings of the International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing*, 2007.

[29] Yamanishi, K. and Maruyama, Y. Dynamic syslog mining for network failure monitoring. In *Proceedings of the International Conference on Knowledge Discovery in Data Mining*, 2005, pp. 499-508.

[30] Yanaihara, N., et al. 2006. Unique microRNA molecular profiles in lung cancer diagnosis and prognosis. *Cancer Cell*, 9, 3, 2006, 189-198.

[31] Yu, S.L. et al. 2008. MicroRNA Signature Predicts Survival and Relapse in Lung Cancer. *Cancer Cell*, 13, 1, 2008, 48-57.

Causal Networks Based Process Improvement

D. Günther[1], R. Neumann[2], K. Georgieva[2] and R. R. Dumke[2]
[1]CERT, Volkswagen AG, Wolfsburg, Germany
[2]Dept. of Computer Science, University of Magdeburg, Magdeburg, Germany

Abstract - **This paper includes a causal-based modelling of software process models in order to analyse the correct relationships between the different (key) process areas of these models. A first short description of causal network approaches shows the identified problems and possible benefits using these formal techniques in the software engineering area. The definition and extension of the causal modelling using causal networks helps to understand the relationships between the different software process artefacts and their causalities. Our causal network based process model (CNPM) concept describes the considered objects outside and inside the software processes or functions and their causalities expressed as roles. The description of first applications of the CNPM approach for the Capability Maturity Model Integration (CMMI) demonstrates the meaningfulness of this approach.**

Keywords - ***Software process improvement, causal network, process analysis and evaluation, software quality***

I. INTRODUCTION

Causal networks as a special kind of semantic networks are very expressive in order to see or analyze the relationships between process activities, areas and indicators in a logical manner. Typical results of such a modelling are

- The consequence of process activities to other ones involving different quality characteristics like correctness, completeness etc. (see [4] and [13])
- The repercussion of the chosen approaches for process evaluation and improvement (see [3] and [9])
- The overview about strong and weak process connections in order to keep quality improvements (see [2], [5] and [6])
- The application of (causal) model-based principles in order to reduce the process complexity and involvements (see [1], [7] and [10]).

In a general manner a causal network "is a directed acyclic graph arising from an evolution of a substitution system, and representing its history" [13]. The process evolution involves causal relationships between events, states, entities, objects, artefacts etc. which could be based on a special kind of empirical reasoning. In following we will characterize a causal network based approach that helps to identify incompleteness and mismatches of text-based process models in an explicit manner.

II. CAUSAL NETWORK-BASED PROCESS MODEL DESCRIPTION AND ANALYSIS

The causal network based process model (CNPM) concept is defined in the following four parts and components of this approach (see [2], [3] and [11]):

A. Causal Network Model Components

The causal network model M^{CNPM} is based on the following software process ingredients and involvements:

$$M^{CNPM} = \langle U^{CNPM}, V^{CNPM}, F^{CNPM} \rangle ,$$

where

- U^{CNPM} is a set of background variables that is determined by ***objects*** $o_{u,i}^{CNPM}$ $(i \in \{1,2,...,m\})$ as software process artefacts outside the considered model
- V^{CNPM} is a set $\{V_1^{CNPM}, V_2^{CNPM}, ..., V_n^{CNPM}\}$ of variables that are determined by ***objects*** $o_{v,i}^{CNPM}$ $(i \in \{1,2,...,n\})$ in the model – that is, variables or objects in $U^{CNPM} \cup V^{CNPM}$; and
- F^{CNPM} is a set of ***functions*** $\{f_1^{CNPM}, f_2^{CNPM}, ..., f_n^{CNPM}\}$ such that each f_i^{CNPM} $(i \in \{1,2,...,n\})$ is a mapping from (the respective domains of) $U^{CNPM} \cup (V^{CNPM} \backslash V_i^{CNPM})$ to V_i^{CNPM} and such that the entire set F_{CNPM} forms a mapping from U^{CNPM} to V^{CNPM}. In other words, each f_i^{CNPM} tells us the value V_i^{CNPM} given the values of all other variables in $U^{CNPM} \cup V^{CNPM}$, and the entire set F^{CNPM} has a unique solution V^{CNPM} *(o)*. Symbolically, the set of equations F^{CNPM} can be represented by writing

$$o_{v,i}^{CNPM} = f_i^{CNPM} (r_i^{CNPM}, o_{v,j}^{CNPM}, o_{u,j}^{CNPM}),$$

$$i,j = 1, \ldots, n, i \neq j$$

where r_i^{CNPM} is any realization of the unique minimal set of variables as ***roles***[1] R_i^{CNPM} in

[1] Against the causality in natural science, software processes are based on activities of subjects. Therefore, we use a description of subjects as *roles*. Note that the roles define the

$V^{CNPM} \backslash V_i^{CNPM}$ sufficient for representing f_i^{CNPM}.

Examples of this CNPM description are

1. $M_1^{CNPM} = \langle U_1^{CNPM}, V_1^{CNPM}, F_1^{CNPM} \rangle$ with U_1^{CNPM} = {'Object 1'}, V_1^{CNPM} = {'Object 2'}, F_1^{CNPM} = $\{o_1^{CNPM}, r_1^{CNPM}\}$, o_1^{CNPM} = {'Function 1'} and r_1^{CNPM} = {'Role 1'}

2. $M_2^{CNPM} = \langle U_2^{CNPM}, V_2^{CNPM}, F_2^{CNPM} \rangle$ with U_2^{CNPM} = {'Object 2'}, V_2^{CNPM} = {'Object 3'}, F_2^{CNPM} = $\{o_2^{CNPM}, r_2^{CNPM}\}$, o_2^{CNPM} = {'Function 2'} and r_2^{CNPM} = {'Role 1'}

Considering the different levels of causality as dependencies and improvements leads to different kinds of analysis and interpretations.

B. Causal Network Model Operations

The M^{CNPM} can be modified in the following manner considering the typical causal relationships between software process artefacts[2]:

- *Union* or *summarizing* of CNPM models consists of the union of the different model parts. A unified CNPM model M_3^{CNPM} could be built as $f\_join(M_1^{CNPM}, M_2^{CNPM})$ with U_3^{CNPM} = {'Object 1'}, V_3^{CNPM} = {'Object 2', 'Object 3'}, $F_3^{CNPM} = \{o_3^{CNPM}, r_3^{CNPM}\}$, o_3^{CNPM} = {'Function 1', 'Function 2'} and r_3^{CNPM} = 'Role1'

- *Partitioning* of a CNPM model consists of building sub models and special parts of models.

- *Restructuring* of a CNPM model is reasonable in different practical situations and consists of addition and extraction of any model parts.

C. Causal Network Model Analysis

The M^{CNPM} can be analyzed considering the typical causal relationships between software process artefacts in the following manner. The CNPM model could be considered as a directed graph where every node has some predecessors and any successors. Hence, it is possible to analyze or count these elements for a first level of CNPM analysis and evaluation. For instance, we obtain the number of all roles in the CNPM, the number of derived objects etc. Based on this idea, we can define the following function $f_{extract\_input}^{CNPM}$ of analysis as

$$f_{extract\_input}^{CNPM} : M_x^{CNPM} \times f_i^{CNPM} \rightarrow U_i^{CNPM}$$

where $M_x^{CNPM} = <U_x^{CNPM}, V_x^{CNPM}, F_x^{CNPM}>$, $f_i^{CNPM} \in F_x^{CNPM}$, $U_i^{CNPM} \subseteq U_x^{CNPM}$ and

$$U_i^{CNPM} = \{ u_i^{CNPM} : u_i^{CNPM} = \textbf{\textit{predecessor}}(f_i^{CNPM}) \}.$$

Applying these functions to our described examples of CNPM models we can derive the following characteristics:

- $predecessor(f_1^{CNPM\_M2})$ = {'Role 1', 'Object 2'}
- $predecessor(f_1^{CNPM\_M3})$ = {'Role 1', 'Object 1'}
- $successor(f_2^{CNPM\_M3})$ = {'Object 3'}

D. Causal Network Model Exploration

The M^{CNPM} can be evaluated in the following manner considering the typical causal relationships between software process artefacts. The CNPM model could be characterized as empirical evaluation that requires the identification of the empirical aspects explicitly. Such empirical characteristic for objects could be process artefact level, artefact quality or process artefact performance. From this point of view, the CNPM model evaluation could be performed as following:

- *causal coverage analysis* of the fulfilled requirements from a special software process point of view,
- *causal trace analysis* of the successful consideration of process flow based requirements,
- *causal achievement analysis* of the derived results and outputs in different parts on the CNPM model.

In order to explain some of these kinds of analysis we will consider the CPNM model M_x^{CNPM} describing the empirical-based process aspects mainly and the CPNM model M_y^{CNPM} describing the causal basics in general. On that we characterize a simple causal coverage analysis as

$$\textbf{\textit{coverage}}_{My}^{CNPM} = \sum(|F_y^{CNPM}| + |U_y^{CNPM}| + |V_y^{CNPM}|) / \sum(|F_x^{CNPM}| + |U_x^{CNPM}| + |V_x^{CNPM}|)$$

where $F_x^{CNPM} \subseteq F_y^{CNPM}$, $U_x^{CNPM} \subseteq U_y^{CNPM}$, $V_x^{CNPM} \subseteq V_y^{CNPM}$.

---

causal heuristics addressed to the considered/presented function in the set of the software process artefacts.

[2] The full formal description you can find in [3] and [11].

Furthermore, in the case of coverage lower 1 we have the situation of any missing objects. That could be characterized in the following manner.

$$F^{CNPM}_{missing\_function} = \{ F^{CNPM}_{x} \setminus F^{CNPM}_{y} : F^{CNPM}_{y} \subseteq F^{CNPM}_{x} \}$$

$$F^{CNPM}_{missing\_input} = \{ U^{CNPM}_{x} \setminus U^{CNPM}_{y} : U^{CNPM}_{y} \subseteq U^{CNPM}_{x} \}$$

$$F^{CNPM}_{missing\_output} = \{ V^{CNPM}_{x} \setminus V^{CNPM}_{y} : V^{CNPM}_{y} \subseteq V^{CNPM}_{x} \}$$

For the causal trace analysis and achievement analysis the existing graph algorithm and methods of evaluation can be used that would not be considered here.

III. CNPM APPROACH APPLICATION FOR CMMI ANALYSIS

One of the possible uses for the CNPM model is the mapping of process standards. This shall be described by example of the key process area „Organizational Training“ (**OT**) of the CMMI (see [8] and [12]). Also it will be considered that specific practices of this model give a hint for the implementation of a CMMI conformant process environment. The specific practice (**SP**) 1.1 will be used as an example for the implementation of a first part of an CNPM network.

A. CNPM-Based Analysis of SP 1.1

The CMMI practice SP 1.1 as “Establish the Strategic Training Needs” contains the following sub practices:

- *Analyze the organization’s strategic business objectives and process improvement plan to identify potential future training needs.*
- *Document the strategic training needs of the organization.*
- *Determine the roles and skills needed to perform the organization’s set of standard processes.*
- *Document the training needed to perform the roles in the organization’s set of standard processes.*
- *Document the training needed to maintain the safe, secure and continued operation of the business.*
- *Revise the organization’s strategic needs and required training as necessary.*

To create a network it is necessary to split the text into tasks, objects and roles. This decomposition leads to the following elements:

Objects: *Strategic business objectives, Process improvement plan, Set of standard processes, Training needs for roles and skills, Training needs for business, Needed roles, Needed skills*

Functions: *Analyse, Document strategic training needs, Determine roles and skills, Document training needs to perform standard processes, Document training needs for safe, secure, continued business, Revise if necessary*

Roles: *The text of the CMMI contains no detailed information about the role executing the task. But it gives the general definition, that the management is responsible for all quality activities. So for the following networks the management will be used as executing instance of this task.*

The resulting network is shown in the following figure 1.

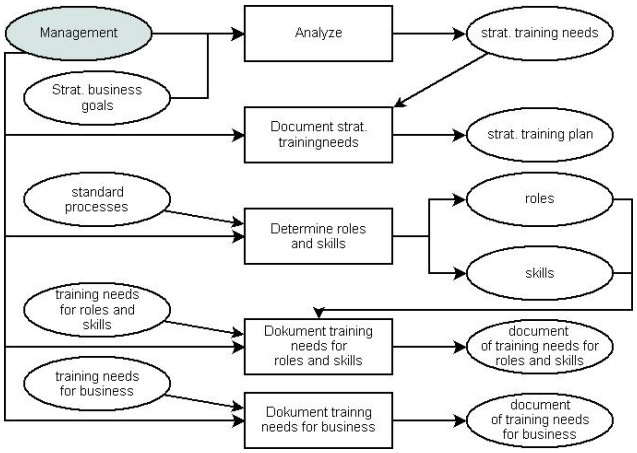


Figure 1: Organizational Training-SP 1.1 – first approach

A deeper analysis of the objects contained in this network shows, that there is no task, creating the objects „training needs for roles and skills“ and „training needs for business“. This shows the incompleteness of the CMMI in some detailed views. The inserted processes are the following:

- *Determine training needs for roles and skills*
- *Determine training needs for business*

Furthermore, it can be seen, that the network contains two functions for documenting two different types of training needs. Giving credit to the fact that the documentation of training needs doesn’t depend on the type of the training need that is to be documented, both functions can be combined to a single one.

- *Document training needs*

The network constructed by these changes is shown in the figure 2. Using the methods described above, the derived networks about the other SP 1 components (as SP 1.2, 1.3, 1.4, 2.1 and 2.2) can be combined to show the complete picture of the tasks fulfilling the requirements of key process area “organizational training”.

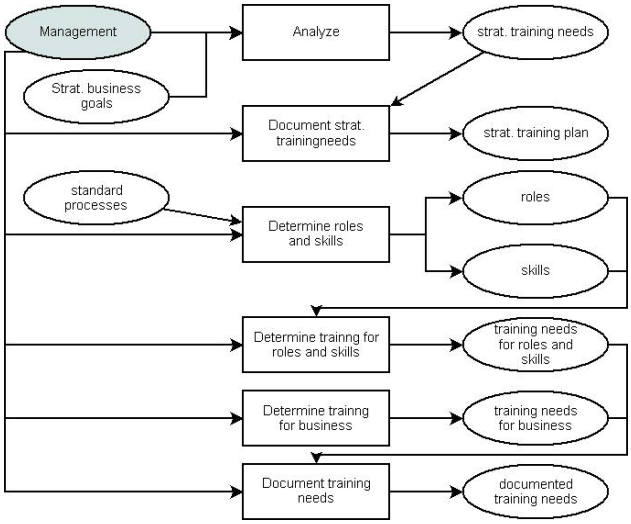


Figure 2: Organizational Training - restructured

B. CNPM-Based Analysis of CMMI Process Descriptions

Further results of CMMI key process (KP) analysis can be characterized as the following chosen situations

- KP "Causal analysis and resolution": created but not used practice 2.1 the "Products",
- KP "Configuration management": created but not used practice 1.2 the "Change request data base"; double (not unique) definition of practice 3.1 as "Configuration documentation"
- KP "Decision analysis and resolution": created but not used practice 1.1 the "Organization standard processes"
- KP "Organizational training": missing management component (see above)
- KP "Project monitoring and control": created but not used practice 1.3 the "Risks documentation" and practice 1.6 the "Changes documentation"
- KP "Project planning": created but not used practice 2.5 the "Knowledge management"
- KP "Process and product quality assurance": created but not used practice 1.2 the "Evaluation criteria" and practice 1.1 "Quality requirements"
- KP "Requirements development": created but not used practice 1.23 the "Requirements history" and practice 1.5 the "Review results"

Note, that the analysis of the KP's "Integrated project management", "Measurement and analysis", "Organizational innovation and deployment", "Organizational process focus" and "Organizational process performance" could be identified with correct (causal-based) semantics.

IV. CONCLUSIONS

The presented CNPM-based approach was applied in practice in order to transform the textual CMMI standard in a causal network based form. This implies the chance of *explicit description* of the CMMI process evaluation from an *implicit* one. Furthermore it allows to consider other causalities and empirical relationships in the software process area depending on concrete industrial situations and methodologies.

In our further research we interpret any improvements in order to keep causal-based correctness in the CMMI. These investigation led to any improvement documented in our next papers.

V. REFERENCES

[1] F. P. Deek, J. A. M. McHugh, , and O. M. Eljabiri: "Strategic Software Engineering – An Interdisciplinary Approach". Auerbach Publications, Boca Raton London New York, 2005

[2] R. R. Dumke, M. Blazey, H. Hegewald, D. Reitz, and K. Richter: "Causalities in Software Process Measurement and Improvement". Proc. of the MENSURA 2006, Nov. 6-8, 2006, Cardiz, Spain, pp.42-52

[3] R. R. Dumke, K. Richter, E. Asfoura, and K. Georgieva: "Process Improvements Using Causal Networks". Proc. of the SERP 2009, July 13-16, Las Vegas, pp. 451-457

[4] W. Emmerich, M. Aoyama, J. Sventek: "The Impact of Research on Middleware Technology". Software Engineering Notes, January 2007, pp. 21-46

[5] N. Fenton, P. Krause, and M. Neil: "Probabilistic Modelling for Software Quality Control". Proc. of the European Conference on Symbolic and Quantitative Approaches to Reasoning and Uncertainty, Toulouse 2001

[6] J. Ferguson, and S. Sheard: "Leveraging Your CMM Efforts for IEEE/EIA 12207". IEEE Software, September/ October 1998, pp. 23-28

[7] W. A. Florac and A. D. Carleton: "Measuring the Software Process – Statistical Process Control for Software Process Improvement". Pearson Education, 1999

[8] M. K. Kulpa and K. A. Johnson,: "Interpreting the CMMI – A Process Improvement Approach". CRC Press Company, 2003

[9] J. Pearl: "Causality – Models, Reasoning, and Inference". Cambridge University Press, 2000

[10] L. H. Putnam and W. Myers: "Five Core Metrics – The Intelligence Behind Successful Software Management". Dorset House Publishing, New York, 2003

[11] K. Richter: "Causal-Based Networks Supported Process Improvement". Software Engineering Notes, 34(2009)9, p. 32

[12] SEI: Capability Maturity Model Integration (CMMI^{SM}), Version 1.1, Software Engineering Institute, Pittsburgh, March 2002, CMMI-SE/SW/IPPD/SS, V1.1

[13] E. Weisstein: "Causal Networks. Script in Computer Science", http://mathworld.wolfram.com/ CausalNetwork.html (February 1, 2011)

Measuring Levels of Abstraction in Software Development

Frank Tsui, Abdolrashid Gharaat, Sheryl Duggins, Edward Jung
School of Computing and Software Engineering
Southern Polytechnic State University
Marietta, Georgia, USA

Abstract – In software engineering and development, we are expected to utilize the technique of abstraction. Yet, it is one of the most confounding topics. In this paper we explore the concept of abstraction as applied to software engineering, define and discuss a conceptual metric called levels-of-abstraction, LOA, and show some attributes of LOA.

Keyword: software, abstraction, measurement

I. Introduction

In developing software from requirements we are often faced with a "first step" syndrome of where should one start. Many high level architectural styles and patterns [1] have helped overcome this initial hurdle. However, we are still faced with further analysis to group similar requirements together along functional line or along some data usage line into sub-components. The question that faces many developers during this early stage of analysis is the decision of what should be the appropriate level of abstraction for specification, design and implementation. In this paper we first explore the general notion of abstraction and enhance the concept to include levels-of-abstraction as it applies to software engineering. Next we propose a "conceptual" metric for levels-of-abstraction, LOA, which helps us gauge the amount of abstraction. Finally, we show some interesting attributes of LOA . This is a report on the current status of our research. This research is showing some promise in the area of explaining and formulating guidelines for the amount of abstraction and the depth of abstraction required in performing different software engineering activities.

II. General Concepts Related to Abstractions in Software Engineering

One of the fundamental reasons for engaging in the task of abstraction in software analysis, design and development is to reduce the complexity to a certain level so that the "relevant" aspects of the requirements, design and development may be easily articulated and understood. This starts with the requirements definition through design to actual code implementation. The general relationships of the individual world domain, the abstractions of those domain entities, and the artifacts specifying those abstractions are shown in Figure 1. The bold, vertical arrows represent the intra-transformations occurring within each individual world domain of requirements, design, and implementation. The horizontal arrows represent the inter-transformations occurring across those domains.

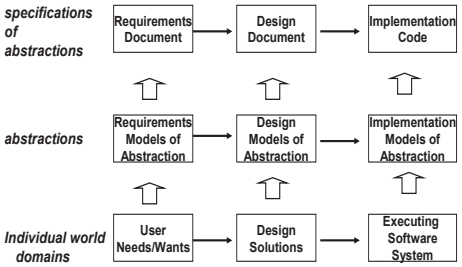


Figure 1: Relationships Among Abstractions, Software Artifacts, and Individual World Domains

The term abstraction used in the form of a verb, as represented with bold vertical arrows in Figure 1 from individual world domain to abstractions, would include the notion of simplification. Simplification represents the concept of categorizing and grouping domain entities into components and relating those components. We simplify: (a) by reduction and (b) by generalization. By reduction, we mean the elimination of the details. Generalization, on the other hand, is the identification and specification of common and important characteristics. Through these two specific subtasks of reduction and generalization we carry out the task of abstraction. Rugaber[6] states that design abstraction is a "unit of design vocabulary that subsumes more detailed information." This notion of abstraction via simplification is also similar to the notion explicated by Kramer [3,4], Perry [5],

Tsui, Gharaat, Duggins and Jung [7] and Wagner and Deissenboeck[8]. Thus via the process of abstraction, we aim to simplify or decrease the complexity of the domain of software design solution.

At the early stage of software development, requirements represent the needs and wants of the users and customers. Different degrees of abstraction may be employed depending on the amount of details that need to be portrayed in the requirements. The intra-transformation is represented by the vertical arrow from User Needs/Wants domain to Requirements Models of Abstraction in Figure 1.

As we move from requirements towards the development of the solution for the user requirements, a new form of abstraction takes place. The new form of abstraction is necessitated due to the fact that the solution world includes the computing machines and other logical constructs that may not exist in the original user requirements. One of the first artifacts from the solution side is the software architecture and high level design of the software system. The inter-transformation of the requirements models of abstraction to the design models of abstraction is shown as the horizontal arrow in Figure 1. Note that the box labeled "Design Models of Abstraction" is the result of two types of transformations:

a) inter-transformation from the requirements domain and
b) intra-transformation within the design solution domain.

The "Implementation Models of Abstraction" box in Figure 1 represents the packaging/loading models of the processes, information and control of the execution of the solution in the form of source code to satisfy the functionalities and attributes described in the requirements and design documents. Thus it is a result of the inter-transformations from requirements models of abstraction through the design models of abstractions and the intra-transformations from the actual software execution domain. The "Implementation Code" box is the specification of this abstraction. The execution of the Implementation Code and the interactions with the users form the "Executing Software System" box in Figure 1. The employment of various abstractions and the transformations of these abstract entities are crucial to software engineering.

III. Measuring Abstraction

Abstraction, both as a verb and as a noun, is a crucial element in software engineering. As a verb, we have defined it as the activity of simplification, composed of reduction of details and the generalization of crucial and common attributes. Now, it is relevant to ask how much abstraction would be appropriate so that we can arrive at the "Implementation Code" and the "Executing Software System" boxes in Figure 1.

Jackson [2] admonishes us that we need to be careful with abstraction and the degree of abstraction because so many seemingly good designs fall apart at implementation time. His warning is well founded in that many design abstractions, the noun, are often missing some vital information for the detail coding activities. In the past, we have utilized the technique of decomposition to move from abstraction to details. However, if our abstraction is generalizing too much to not include the vital information, then Jackson's warning will turn into reality. The levels of abstraction should be different for various software artifacts and be dictated by the purpose of abstraction. Wang [9] has expressed a similar concern and defined a Hierarchical Abstraction Model for software engineering; his hierarchical model of abstraction describes the necessary levels of preciseness in representing abstractions of different objects. In terms of our Figure 1, Wang addressed the issue of rigor of specifications of abstraction in the requirement and design documents, not how much should be included in the abstraction.

The how much, or the amount, of abstraction is a reflection of the result of the simplification activity. Measuring the amount of abstraction is gauging the extent of reduction and generalization that took place. For example, this may be possible in the requirements domain. We may consider the set of the original requirements statements of needs and wants as X in the "User Needs/Wants" box in Figure 1. Then, |X|, the cardinality of X is a count of the raw requirement statements collected through some solicitation process. These are the pre-analysis requirements statements. We then designate Y as the statements in the "Requirements Models of Abstraction" box. The cardinality of Y, |Y|, is a count of the statements that resulted from requirements analysis, which include activities such as organizing, grouping, prioritizing, etc. In other words, the post-analysis of the solicited requirements is a form of abstraction of the raw user needs and wants requirement statements. Then the "difference" between |X| and |Y| is:

$$(\text{Level-of-Abstraction})_{REQ.} = |X| - |Y|.$$

$(\text{Level-of-Abstraction})_{REQ}$ represents the "difference" between pre-analysis and post analysis of requirements, and it may be considered a "conceptual" metric of abstraction for requirements.

Since simplification is a vital characteristic of abstraction, we expect |Y| to be less than |X|. Thus we will need to further refine this definition with the constraint that if |Y| is not less than |X|, then no abstraction activity really took place. We will also take the subscript, REQ, off the terminology for the general case. In general, let X be the statements in the domain world, and let Y be the set of statements in the abstraction, the noun, then

Level-of-Abstraction (LOA) = |X| - |Y|, if |X|>|Y|
else
= 0

Note that when the abstraction activity is carried to its extreme, |Y| should just be 1. For example, all the raw requirement statements of needs and wants are abstracted into one abstract statement. Thus Level-of-Abstraction is bounded by (|X| - 1) and 0.

IV. Requirements Abstraction Example:

In this section we will further explore the Level-of-Abstraction measurement concept, using requirements analysis as an example. Note that it is very likely that X and Y are not expressed with the same language. English sentences and some diagrams may be the main ingredients of the wants and needs expressed by the users and customers. The result of requirements prioritization, categorization and analysis is some form of abstraction, Y, which may be expressed with a Use Case Diagram. The amount of requirements abstraction defined as the "difference" between pre and post analysis of requirements is shown through an example of functionally partitioning the requirements.

Suppose there are X = {x1, x2, ---, xz} raw requirement statements. The set X may contain a variety of statements, referring to functionality, data and other attributes. A common type of abstraction that may be employed is to categorize and group X by functionality. Thus only a subset of X, X', is addressed. X' is the subset of requirement statements that addresses functionality needs. Let X' = {xx1, xx2, ---, xxn}, where |X'| ≤ |X|. The subset X' is analyzed and then partitioned into some set of categories of functionalities. This partitioned set will be called set Y. Y may look as follows.

Y = {(xx1, xx2); (xx3, xx5, xx10); ----}

Every functionality xxi ϵ X' is in one of the partitions of Y and in only one of the partitions. Renaming the partitioned set Y as follows would yield:

Y = {y1, y2, ----, yk} where
y1 = (xx1, xx2)
y2 = (xx3, xx5, xx10)
.
.

The set Y may be represented by a Use Case diagram where y1, y2, ---, yk are the named interaction represented as "bubbles" in the Use Case diagram. Clearly there is more than one way to partition X'; thus there may be different Y's. A use case diagram with one "bubble" would be an extreme case as well as a use case diagram with a bubble for each xxi. The extreme points of |Y| = 1 or |Y| = |X'| would be very rare.

Now consider a specific case where the domain set X has 4 functional requirements x1, x2, x3, x4. Then there are the following partitioning sets, P's for different functional abstractions, Y's.

P0 has Y01 = {(x1); (x2); (x3); (x4)} = X

P1 has Y11 = {(x1); (x2); (x3,x4)},
Y12 = {(x1); (x3); (x2,x4)},
Y13 = {(x1); (x4); (x2,x3)},
Y14 = {(x2); (x3); (x1,x4)},
Y15 = {(x2); (x4); (x1,x3)} and
Y16 = {(x3); (x4); (x1,x2)}

P2 has Y21 = {(x1); (x2,x3,x4)},
Y22 = {(x2); (x1,x3,x4)},
Y23 = {(x3); (x1,x2,x4)}and
Y24 = {(x4); (x1,x2,x3)}

P3 has Y31= {(x1,x2); (x3,x4)},
Y32= {(x1,x3); (x2,x4)}, and
Y33= {(x1,x4); (x2,x3)}

P4 has Y41 = {(x1,x2,x3,x4)}

At P0, there is only one abstraction, Y01, which is the same as the original requirement set X. So Level-of-Abstraction is |X| - |Y| = 0. There is no abstraction at P0. At P1, any of the Y1x has a cardinality of 3. So at P1, |X| - |Y| = 4 – 3 = 1. The Level of Abstraction is 1. At P2, the Y2x's are grouped differently and each has a cardinality of 2. Thus, at P2, |X|-|Y| = 4-2 = 2. P3 partitioning has the functionalities grouped differently, but each Y3x has a cardinality of 2, just

like those in P2. At P3, |X| - |Y| = 4 -2 = 2. Thus all partitions in P2 and in P3 are at the same Level-of-Abstraction, 2. Finally at P4, there is again only one abstraction, Y41, which combined all four functionalities into 1 category. Thus at P4, |X| - |Y| = 4 – 1 = 3. The Level-of- Abstraction is the highest here.

From this example, one can easily see that abstraction of functionalities into groups for a relatively small set of four functional requirements has many choices. In this case there are 15 choices and there are 4 different Level-of-Abstraction, namely 0 through 3. Coming up with the one "best" abstraction is not an easy task even with this small example.

V. Some General Theorems

We have shown that for a |Y| = k, there may be several different abstractions with that same cardinality. That is, given an abstraction level j, there may be more than one solution.

Theorem 1: For Level-of-Abstraction = |X| - |Y| = j, there exists more than one abstraction or partitioned set, Y, at that Level-of-Abstraction, unless j = 0 or j = |X| -1.

Proof: Given |X| - |Y| = j, if j=0 then |X| - |Y| = 0 and there is no abstraction. If |X| - 1 = j, then |Y| has only 1 category with all functionalities included, so |X| - |Y| = |X| - 1, which is the highest level of abstraction. From combinatorics, we know for any set with n>0 elements and r an integer such that $0 \leq r \leq n$, then the number of subsets that contain r elements of S is $\frac{n!}{r!(n-r)!}$. Hence for all the values in between, there will be partitioned sets.

Note that the different Level-of-Abstractions as shown in our simple example of P0 through P4 form a partially ordered set. Next we introduce an Up and a Down operator on Level-of-Abstraction. Given a Level-of-Abstraction, Px, and a Level-of-Abstraction, Py, then UP and Down operators are defined as follows:

UP (Px,Py) = Px if Py ≤ Px and
= Py otherwise.
Down(Px,Py) = Py if Py ≤ Px and
= Px otherwise.

Theorem 2: Set of Level-of-Abstractions, with the operators of Up and Down form a lattice structure.

Proof: The set of Level-of-Abstractions is a partially ordered set, or a Poset. A lattice is a Poset in which any two elements have a lowest upper bound (lub) and a greatest lower bound (glb). The Up operator provides us the glb and the Down operator provides us the lub. Thus the set of Level-of Abstractions forms a lattice.

VI. SUMMARY AND RESULTS

In this paper we explored the general notion of abstraction as applied to software engineering. We further proposed a "conceptual" metric for levels-of-abstraction, LOA, which allows us to gauge the amount of abstraction. Lastly, we showed some characteristics of LOA. Our current research direction is to analyze LOA further to help us pick the "right" level of LOA for a specific domain.

References

1. D. Garlan and M. Shaw, "An Introduction to Software Architecture," CMU-CS-94-166, Carnegie Mellon University, Pittsburgh, PA, 1994.
2. D. Jackson, *Software Abstractions Logic, Language, and Analysis,* MIT Press, 2006.
3. J. Kramer, "Is Abstraction the Key to Computing," Communications of the ACM, Vol. 50, No 4, April 2007, pp 37-42.
4. J. Kramer and O. Hazzan, "Introduction to The Role of Abstraction in Software Engineering," International Workshop on Role of Abstraction in Software Engineering," Shanghai, China, May, 2006.
5. D. E. Perry, "Large Abstractions for Software Engineering," 2nd International Workshop on Role of Abstraction in Software Engineering, Leipzig, Germany, May, 2008.
6. S. Rugaber, "Cataloging Design Abstractions," International Workshop on Role of Abstraction in Software Engineering," Shanghai, China, May, 2006.
7. F. Tsui, A. Gharaat, S. Duggins and E. Jung, "Software Architecture: Functional Composition and Decomposition Complexities," Internal Research Report, Software Engineering, Southern Polytechnic State University, July 2010.
8. S. Wagner and F. Deissenboeck, "Abstractness, Specificity, and Complexity in Software Design," 2nd International Workshop on Role of Abstraction in Software Engineering, Leipzig, Germany, May, 2008.
9. Y. Wang, "A Hierarchical Abstraction Model for Software Engineering," 2nd International Workshop on Role of Abstraction in Software Engineering, Leipzig, Germany, May, 2008.

Reusing Functional Testing in order to Decrease Performance and Stress Testing Costs

Ismayle de Sousa Santos
MDCP/UFC, Fortaleza, CE, Brazil
ismaylesantos@great.ufc.br

Alcemir Rodrigues Santos
DCC/UFMG, Belo Horizonte, MG
alcemir@dcc.ufmg.br

Pedro de Alcântara dos S. Neto
DIE/UFPI, Teresina, PI, Brazil
pasn@ufpi.edu.br

***Abstract* - This work presents an experimental study of an idea related to the automatic generation of performance and stress testing by reusing functional testing. The idea was implemented in a tool named FERRARE GT. This tool is able to generate both test scripts as well as the data required for their execution. In this study we verified that the use of the method can generate benefits related to cost reduction, from the reduction of test effort and, at the same time, benefits related to test quality, from the improvement of the test relevance for the software development.**

Keywords - software testing; data generation; non-functional requirements; experimental study.

I. Introduction

Testing is a critical element in the software quality control and represents the final review of the analysis, design and implementation. However, testing is usually not performed as it should. A fundamental factor that contributes to this situation is the activity cost, which can get as high as 50% of the total project cost [2].

A test category performed by many organizations is functional testing. It aims at verifying the software behavior [6]. There are various other test objectives, as, for instance, performance and stress testing. They are much less often executed that the functional test and have different purposes. The performance test aims at validating the performance requirements, as, for instance, the response time in a specific context, like the access of 100 users in a local network environment. Stress testing is similar to performance testing, however, the execution context is elevated to levels above the average, verifying the system operation in such cases and certifying that no unusual behavior occurs.

In general many organizations that develop Web systems perform functional tests. However, few execute performance and stress tests before launching the system, even though they are just as important for the Web environment. It is fundamental the development of mechanisms which motivates the execution of such tests so needed by Web systems.

From this scenario, it was noticed that the creation of a mechanism, which aids the development of performance and stress testing, from any common artifact to the software development, could result in the cost reduction. Because they are so well known and have a big portion of the necessary information, the functional test was chosen for the input of such automation. From that it was developed a tool, named FERRARE [8], with such an objective. The initial prototype tool enabled the generation of performance and stress testing scripts from functional testing scripts. However, limitations in the tool, which prevented its use in an industrial environment, were discovered.

This paper describes the extensions performed in FERRARE to allow its use in an industrial environment related to software development, as well as an experimental study performed in order to evaluate the benefits related to incorporate the tool in a software development environment.

There are related works that propose the generation of performance tests based on models describing the system under test [4,5,9]. By the other hand, Bertolini et al [1] propose four black box test techniques in order to crash the system, by using a special kind of functional tests. It is important to emphasize that FERRARE does not use models. It generates performance and stress tests scripts by reusing functional tests scripts. Besides, FERRARE can infer the data related to execute a functional test and generate suitable data to execute several tests instances, isolating each one from the others.

This paper is organized as follows: Section II presents FERRARE GT; Section III presents the experimental study; Section IV presents a discussion about the idea and the study results; Section V concludes the paper and presents directions for future works.

II. FERRARE GT

FERRARE is a tool developed for the generation of performance and stress testing scripts from functional testing scripts. It is divided into two modules: Extractor and Generator.

FERRARE was conceived to work with any functional testing tool and any performance and stress testing tool, as long as the extractor and the generator for the desired tools are created, as it is going to be discussed later on. Nowadays, FERRARE works with the functional testing tools Selenium IDE and Canoo Web Test, besides performance and stress testing tools Apache JMeter and WebLoad. This means that it generates performance and stress testing to be executed on JMeter or the WebLoad, from functional testing created with Selenium IDE or Canoo WebTest (input tools) [8]. A sketch of the functioning of FERRARE can be seen in Figure 1.

The Extractor module is responsible for the extraction of the information inside the functional test script. This included the identification of the actions related to the test (test procedure) and of the input data, expected outputs and other

test conditions (test case). The extraction generates an abstract representation of the functional test, independent of technology.

The Generator module is responsible for the generation of performance testing based on the information supplied by the Extractor. This generation involves the specification of different parameters such as the quantity of concurrent users, time limits and number of machines used for test execution.

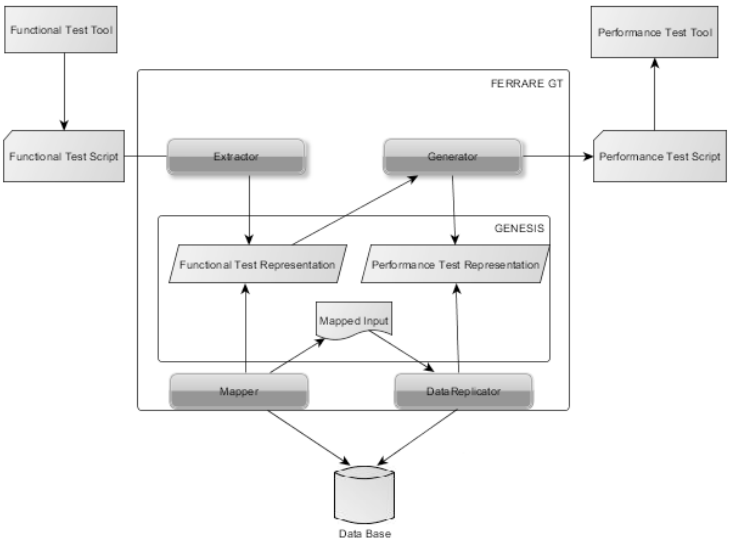


Figure 1. FERRARE GT overview.

FERRARE generates performance and stress testing from the creation of "copies" from functional testing, taking in consideration the restrictions associated to the inputs used in this test. If a functional test that performs a book register in an application is used, FERRARE can generate 100 "copies" of this test, in the format required by performance and stress testing, respecting the characteristics of the fields, as an obligatoriness, sizes and formats. It is important to emphasize that the tool does not perform a simple "copy", since several other actions are also executed, in order to allow its concurrent execution [8].

The initial version of FERRARE did not generate the necessary data for the execution of performance and stress testing. Because of that, its automation level was very limited. This originated another project, with the goal of generating data for performance and stress testing, from an analysis of the data used for the execution of functional testing. Based on that, another module was created for FERRARE, named GENESIS. Its incorporation to the tool originated FERRARE GT [3].

GENESIS aims at generating data for performance and stress testing, by reusing the data from the functional test that serves as a basis for the generation of performance and stress testing. The behavior of the tool is based on the replication of the data derived from the functional test. The central idea implemented is that the replication of data from functional testing can work as a basis for the execution of various concurrent functional testing, resulting in a performance or stress testing.

It is important to emphasize that this approach eliminates the need of having knowledge of all the contraints related to the data model and the application business rules. Because the data are replicated from an instance of the database able to execute a functional test, the replicas should also keep the same feature. This approach is innovative, because it reduces the complexity for the data generation. The innovation is precisely in using a functional test, and the state of the database before its execution, for such replication.

III. Experimental study

A. Goal Definition

An experimental study with the goal of evaluating the use of FERRARE GT in a scenario of software development was performed. The purpose of the study was to evaluate, regarding effort and quality, from the point of view of the researcher, in the context of students of Computer Science, the feasibility of FERRARE GT in order to automate the data generation and performance and stress tests development for a Web system.

The experiment was executed in a controlled environment (in vitro) with the participation of students from Software Engineering Class from the Computer Science Course from UFPI (Federal University of Piauí).

The subjects had to create performance tests for a Web system on book loaning, named in this work BibSystem. The goal of this system is to allow students to perform book loans available in the library. The system users can authenticate themselves in the system, perform a loan, search for a book and return it. The Login (authentication) function was explored in the observation sections.

B. Planning

The experimental study was planned to be executed by students that attended the classes of Software Engineering I and II offered in the first semester of 2010. The Software Engineering I class is offered in the fifth semester of the course and the Software Engineering II class in the sixth, from a total of eight semesters.

The subjects did not have any experience at all in the use of functional testing tools neither in the use of performance and stress testing tools. A verification form was applied to certify this. Therefore, no planning at all was done regarding the type of grouping based on the profile of the subjects [10].

For the execution of the experiment it was planned the use of the tools Selenium (functional testing) and JMeter (performance and stress testing). Both tools were selected because of their big acceptance within the software development industry.

The goal of the study was to evaluate if the reuse of functional testing, for the generation of performance and stress testing, from the support of FERRARE GT is more effective than the development of the same tests in a direct way on JMeter tool. However, generating performance testing without the support of FERRARE GT does not imply only in creating

the test script for the selected tool (JMeter), as well as generating the data required for its execution. This generation is usually done using programs that execute data insertion commands in databases. There are other alternatives, but this was the one used for the study. It is important to emphasize that FERRARE GT does not only support the creation of test scripts, but also generates the data required for its execution.

As mentioned before, the use cases used in the study were Login and Loan. Performance testing for the Login function consisted of executing 100 concurrent authentications, making use of different users and verifying if this happened in up to 5s. Performance testing for Loan consisted of executing 100 different book loans, by different users and verifying if this happened in up to 8s.

Because of that, the main question related with the study was: does the use of FERRARE GT generate a reduction in the required effort for the creation of the performance testing, including the generation of the data necessary for the execution of the tests, when compared with the creation of the same tests using only JMeter? The null hypothesis, related to this question is: there is no difference in terms of effort, measured in minutes, to create tests and to generate data with or without the support of the tool, that is, H0: TestEffort(FERRARE_GT) = TestEffort(JMeter). The alternative hypothesis is that the effort applied in the test, with the support of the method is smaller than the effort applied without the use of the tool, that is, H1:TestEffort(FERRARE_GT) < TestEffort(JMeter).

The experimental sketch used was planned taking in consideration the possible threats to its validity. The Figure 2 summarizes the experimental sketch used. The highlighted parts identify the activities whose time spent by the subjects was registered. The other ones represent the training activities, which respect the times showed in the picture. As it can be visualized in the picture, all the participants had contact with both tools, but in different moments and performing exactly the same tasks. This allowed one group to act as a control of the other. The experiment was divided in two phases. In the first phase all the volunteers generated the data required to execute the 100 book loans simultaneously in the BibSystem. This included the generation of users, books and copies.

In the second phase, the subjects from the Group 1 (G1) should create tests using JMeter and only later create the same tests using FERRARE GT. The subjects from the Group 2 (G2) should begin creating the tests with FERRARE GT and later create tests with JMeter. The attribution to the groups was planned to be random. The selection of the participants was planned based on convenience; that is why the study is considered a quasi-experiment [10].

It was planned that all the tests created by the subjects would be verified by the paper authors, to certify their quality. This means that the difference between them would be only the way used to create them: either using FERRARE GT or JMeter itself. Each submission of the test generated a verification to certify its quality. If the test was not suitable, the registered errors would be highlighted and the subject should proceed with its correction.

The experimental study was planned to reduce threats related to its Internal Validity and External Validity, which are the most important ones related to the studies in the area of Software Engineering [10].

The internal validity defines if the relationship observed between the treatment and the result is causational and not an influence of other factors which are not controlled. The experimental design used reduces the risk of having a bias, since it was planned that every subject would use both treatments, but in different moments. It was also planned the use of the treatments in an alternated order, to evaluate if the execution order could influence the results.

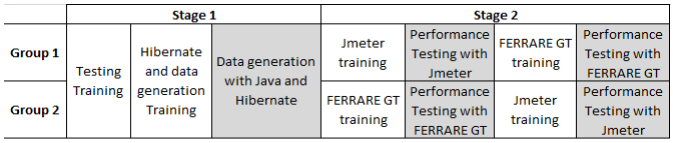


Figure 2. Experimental study scheduling.

The chosen experimental design, where all the subjects used both tools allowing one group to act as the control of the other, validates the conclusion obtained, in the same time that it reduces any threat related to the competitive behavior and to the compensatory behavior [10].

The external validity defines the conditions which limit the ability to generalize the results of an experiment for the industrial practice. The subjects, students from the 5th and 6th semesters of the course, find themselves in the final phase of under graduation, having similar skills of a professional with little experience. The performed trainings, along with the fixation exercise contributed to a good formation of the subjects in the used tools. The BibSystem, although small, had the characteristics commonly existents in Web information systems. Thus, it is believed that the conclusions obtained in the study can be extended to other systems, with the usual size and used by professionals with little experience, without losses in the observed results.

C. *Operation*

Before the beginning of the study activities it was performed a brief presentation related to the activities that would be executed, but the subjects did not have knowledge of the hypotheses that were being tested. It also guaranteed the anonymity of the students explaining even how the data collected would be used.

As mentioned before, the experiment was divided in two phases. In the first phase, all the subjects had training in software testing and in the use of Hibernate[1] and Java to generate data in MySQL database. In the second phase, the subjects had training in the tools that would be used and were asked to create performance tests for the BibSystem.

During the first phase a general vision about software testing was presented to the subjects. The training about tests lasted around 2h and focused on presenting the relevance of the tests, as well as the basic concepts related, the main techniques and existing objectives.

Afterwards, everybody participated in a training about the use of the Java language combined with the Hibernate framework for the generation of data in a MySQL database. The Java language was chosen because it is the one better dominated by the subjects. The Hibernate framework was selected because of the easiness it offers when working with database in the Java language. The MySQL database was used because of its broad use in the academic field and especially in the field where the study took place.

After the training, all the subjects created programs in Java that generated and stored in a MySQL database the required data for the execution of 100 concurrent loans in the BibSystem. It is emphasized that the generated data had to follow the database structure and constraints. The individual time spent by each subject in this activity was registered, since it was executed automatically with the support of FERRARE GT.

During the second phase, the study subjects had to create and execute performance tests for the Loan function. Firstly a training section was performed, lasting about 2 hours, about the tool that would be used. After that the tests were created and executed by the subjects. The Group 1 subjects began using only JMeter in the creation of the tests and used the data generated during the phase one to allow its execution. Afterwards, they used FERRARE GT to create tests for the same function. The Group 2 subjects executed the activities in a reverse path, beginning with FERRARE GT and only later doing the same activity using JMeter.

Twenty-one volunteers participated in the study, from whom 15 concluded all the planned activities. Thus, only the data from these 15 subjects were considered during the analysis and interpretation. This happened because the tests generated by some of them did not reach the quality limit specified. Because of that, their results were not considered concluded.

D. Analysis and Interpretation

Figure 3 presents the data collected from the study. Figure 4 presents the data in the form of a bar chart. Analyzing such data it is perceived that the subjects that used FERRARE GT dedicated an effort considerably smaller for creating performance tests.

The results certify that the idea proposed in FERRARE GT, that reusing functional testing for automation of performance testing is a good alternative. The effort for creating performance testing, as well as the preparation of the environment for its execution, mainly related to the generation of data, can be considerably reduced. This was noted analyzing all the subjects (that is, independent of group) as well as the groups individually. In both cases the gain was expressive and confirmed through student t test.

Group 1 subjects began the study using JMeter directly, while Group 2 began through the use of FERRARE GT. It is possible to notice that the time to generate a test with the use of FERRARE GT by Group 2 was expressively bigger than the time registered by Group 1 for the same task. This indicates that the learning obtained by utilizing JMeter beforehand favoured the use of FERRARE GT. This was expected, once the tools are similar and have similar goals. The knowledge gained with the use of one can influence the use of the other.

Group	Subject	FERRARE GT	Without Using FERRARE GT		
			Data Generation	Jmeter	Total
1	1	7	26	19	45
	2	5	39	19	58
	3	34	31	78	109
	4	6	28	57	85
	5	8	23	54	77
	6	8	45	80	125
	7	7	32	15	47
	8	7	22	58	80
	9	14	33	88	121
	10	9	17	45	62
	11	13	72	85	157
	Average	11	33	54	88
2	12	44	18	54	72
	13	39	26	102	128
	14	23	30	49	79
	15	40	23	74	97
	Average	37	24	70	94

Figure 3. Study data summary.

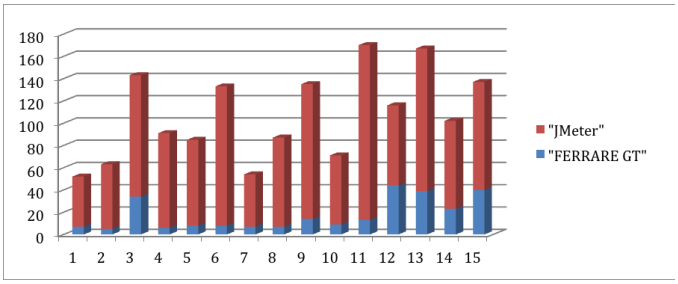


Figure 4. Bar chart showing the study results.

Analyzing the threats to the study, it can be noticed that it does not seem to have been any underlying factor that have interfered in the study. The results reflect the use of treatments and not uncontrolled factors.

Since the subjects had to automate tests for the same part of the system, using both FERRARE GT and JMeter, the problem of instrumentation was not noticed, since there is no difference in the problem used in the study.

Nothing related to the study indicated that there were any threats related to history, that is, it was not identified any external effects that could influence directly in the results differently from what was mentioned above. Relating to maturing, it is believed that the subjects improved with the experience, since they are students and are submitted to new contents, but nothing that could influence directly the result of the developed study.

IV. DISCUSSION

In this article, an experimental study to evaluate the impact of the reuse of functional testing for the generation of performance testing was presented. This was enabled by the tool FERRARE GT, which implements such an idea. It was

possible to verify that this approach is very promising. The effort to generate performance testing with support of the tool FERRARE GT is smaller than the effort to generate the same test without the support of the aforementioned tool.

Performance testing requires data for its execution. The required effort for such preparation can be an inhibiting factor in its use in the industrial environment. The example used in this work illustrates this well: to test the performance of the Loan function of the BibSystem it is necessary to have 100 users and 100 book copies available. Note that this is a fairly simple example and even so it demands a considerable effort. Any tool that helps reducing such effort is contributing to a systematization of performance testing, which is still seldom used by organizations.

The idea explored in this work was based on the reuse of functional testing. Since functional testing indicates the expected inputs and outputs to evaluate a behavior, we can execute such instance various times concurrently to evaluate the performance of this function. However, it is necessary to create copies of the required data, so that each test uses its own data and does not interfere in the execution of the other. The study certified the viability of the proposal, having obtained surprising results to the continuity of the work.

Reusing functional testing to generate performance and stress testing can still bring indirect gains not measured in the study: bigger diffusion of the use of functional testing, because now they have a more important role in the development process, and a bigger quality of the generated tests, since the saved effort can be used in nobler activities.

V. Conclusion and future works

In this work an experiment performed in order to validate the applicability of a tool for the generation of data and performance and stress testing by reusing functional testing, named FERRARE GT, was described. The basic idea is to reuse the existing information from functional tests, in such a way that both the performance and stress testing and the required data for its execution are automatically generated.

Based on the study performed, it was noticed that the use of FERRARE GT in an organization that already performs functional testing can bring a reduction in the effort required to create performance tests and to generate the required data for its execution. This facilitates the systematic use of performance and stress testing by organizations, since the required effort for its use is reduced. Also, the relevance of the functional test increases, what is benefic to the organization, which will be able to give more importance to something that is already considered fundamental by most industries.

As a future work it is intended to increase the experimental study. Also, it is intended to use the tool developed in an industrial environment. FERRARE GT also has many possibilities of improvement, which will be able to allow its use in a simpler and more effective way, raising the associated gains.

VI. acknowledgements

This work was supported by grants from UFPI and CNPq (560128/2010-0). Infoway Technology provided some software products for testing and IT professionals to discuss the directions.

References

[1] Bertolini, C., Peres, G., d'Amorim, M., Mota, A. An Empirical Evaluation of Automated Black Box Testing Techniques for Crashing GUIs. *In Proceedings of the 2nd International Conference on Software Testing Verification and Validation,* p. 21-30, Los Alamitos, CA, USA, 2009.

[2] Binder, R. Testing Object-Oriented Systems: Models, Patterns, and Tools. Addison-Wesley, 2000.

[3] Fé, I. S., Santos, I. S., Santos, A. R., Santos Neto, P. Geração de Dados para Testes de Desempenho e Estresse a Partir de Testes Funcionais. In: *Anais do IX Simpósio Brasileiro de Qualidade de Software,* p. 89-101*,* Belém, PA, 2010.

[4] Garousi, V., Briand, L., Labiche, Y. Traffic-aware stress testing of distributed systems based on UML models. *In Proceedings of the 28th International Conference on Software Engineering (ICSE)*, pages 391-400, Shangai, China, 2006.

[5] Hartman, A., Nagin, K. The AGEDIS tools for model based testing. In Proceedings of the International Symposium on Software Testing and Analysis (ISSTA 2004), Boston, Massachusetts, USA, July 2004.

[6] Myers, G. *The Art of Software Testing*. John Wiley & Sons, 2004, 2nd edtion.

[7] Santos, I. S., Santos Neto, P., Moura, R. S., Soares, A. C. B. Documentação Dirigida por Testes. In: IX Simpósio Brasileiro de Qualidade de Software, Belém, PA. Anais do IX Simpósio Brasileiro de Qualidade de Software, 2010. p. 25-40.

[8] Santos, I. S., Santos, A. R., Santos Neto, P. FERRARE GT: Automação de Testes de Desempenho e Estresse via Testes Funcionais. In: *Congresso Brasileiro de Software: Teoria e Prática* (CBSoft), 2010, Salvador, BA. XVII Sessão de Ferramentas, 2010. v. 4. p. 49-55.

[9] Shams, M., Krishnamurthy, D., Far, B. A Model-Based Approach for Testing the Performance of Web Applications. In: *Proceedings of the 3rd International Workshop on Software Quality Assurance,* p. 54–61, Portland, Oregon, 2006.

[10] Wohlin, C., Runeson, P., Host, M., Ohlsson, M., Regnell, B., Wesslen, A. *Experimentation in Software Engineering: An Introduction*. Kluwer Academic Publishers, 2000.

Empirical Analysis for Investigating the Effect of Control Flow Dependencies on Testability of Classes

Mourad Badri and Fadel Toure
Software Engineering Research Laboratory
Department of Mathematics and Computer Science
University of Quebec at Trois-Rivières, Trois-Rivières, Quebec, Canada
{Mourad.Badri, Fadel.Toure}@uqtr.ca

Abstract — **We present, in this paper, a new metric capturing in an integrated way different attributes of object-oriented systems. The metric uses basically control flow paths and probabilities. It captures the interactions between classes and related control. The study presented in this paper aims at exploring empirically the relationship between the proposed metric and testability of classes. We investigate testability from the perspective of unit testing. We designed and conducted an empirical study using data collected from two open source Java software systems for which JUnit test cases exist. To capture testability of classes, we used different metrics to quantify the corresponding JUnit test cases. In order to evaluate the capability of the new metric to predict testability of classes, we used statistical tests using correlation. The achieved results provide evidence that there exist a significant relationship between the proposed metric and testability of classes.**

Keywords: Software Testability, Testing Effort, Metrics, Control Flow, Control Dependencies, Probabilities and Empirical Analysis.

I. INTRODUCTION

Software testing has an important effect on the quality of the final product. Software testing is probably the most complex task in the software development cycle. It's also a time and resources consuming process. The overall effort spent on testing depends, in fact, on many different factors including: human factors, process issues, testing techniques, tools used, and characteristics of the software development artifacts [3, 4, 9, 35, 36]. Testability is an important quality characteristic of software. Software testability is related to testing effort reduction and software quality [14]. It impacts test costs and provides a means of making design decisions [31]. Zhao [36] argues that testability expresses the affect of software structural and semantic on the effectiveness of testing following certain criterion, which decides the quality of released software. Several software development and testing experts pointed out, in fact, the importance of testability and design for testability. Moreover, Baudry et *al.* [3] argue that testability becomes crucial in the case of object-oriented systems (OOS) where control flows are generally not hierarchical, but diffuse and distributed over whole architecture.

Metrics can be used to predict testability and better manage the testing effort. Having quantitative data on the testability of a software can, in fact, help software managers, developpers and testers to [8, 15]: plan and monitor testing activities, determine the critical parts of the code on which they have to focus to ensure software quality, and in some cases use this data to review the code. A large number of object-oriented metrics (OOM) have been proposed in literature [17]. Some of these metrics (such as coupling, complexity and size) have already been used to measure testability of OOS [8]. However, as stated by Gupta et *al.* [15], none of the OOM is alone sufficient to give an overall reflexion of software testability. Software testability is, in fact, affected by many different factors.

We presented in a previous work [2] a new metric, called *Quality Assurance Indicator* (Qi), capturing in an integrated way different attributes of OOS such as complexity and coupling (interactions between classes). The metric uses control flow paths (capturing the distribution of the control flow in a system) and probabilities. The Qi of a class C_i includes different intrinsic characteristics of the class itself, as well as the Qi of collaborating classes. The metric has, however, no ambition to capture the overall quality of OOS. Moreover, the objective is not to evaluate a design by giving absolute values, but more relative values that may be used, for example, for identifying the critical classes that will require a high testing effort to ensure software quality. The metric has been implemented for Java programs. We compared, in [2], the Qi metric using the Principal Components Analysis (PCA) method to some well-known OOM. The evaluated metrics were grouped in five categories: coupling, cohesion, inheritance, complexity and size. The objective was to find in which proportions the Qi metric captures the information provided by the selected OOM. The obtained results provided evidence that the Qi metric captures, overall, more than 75 % of the information provided by most of the evaluated metrics. The purpose of the present paper is to explore empirically the relationship between the Qi metric and testability of classes in OOS in terms of the effort needed for testing. We investigate testability from the perspective of unit testing, where units consist of the classes of an OOS. We designed and conducted an empirical study using data collected from two open source Java software systems for which JUnit test cases exist. To capture testability of classes, we used different metrics to measure some characteristics of the corresponding JUnit test cases. In order to evaluate the capability of the new metric to predict testability of classes, we used statistical tests using correlation.

The remainder of the article is organized as follows: Section 2 gives a survey on related work on software testability. The proposed metric is presented in Section 3. In Section 4 we define the used metrics to quantify JUnit test classes, describe the experimental design and discuss the statistical technique we used. Section 5 presents the used systems. We also present and discuss in this section the obtained results. Finally, Section 6 summarizes the contributions of this work and outlines directions for further research.

II. SOFTWARE TESTABILITY

IEEE [18] defines testability as the degree to which a system or component facilitates the establishment of test criteria and the performance of tests to determine whether those criteria have been met. ISO [19] defines testability (characteristic of maintainability) as attributes of software that bear on the effort needed to validate the software product. Fenton et *al.* [11] define testability as an external attribute. Indeed, testability is not an intrinsic property of a software artifact and cannot be measured simply such as size, complexity or coupling. Testability measurement is, in fact, influenced by various parameters as stated by Baudry et *al.* [3, 4]. Yeh et *al.* [35] argue also that diverse factors such as control flow, data flow, complexity and size contribute to testability. According to Zhao [36], testability is an elusive concept, and it is difficult to get a clear view on all the potential factors that can affect it. Freedman introduces testability measures for software components based on two factors: observability and controllability [12]. Voas defines testability as the probability that a test case will fail if the program has a fault [32]. Voas and Miller [33] propose a testability metric based on the inputs and outputs domains of a software component, and the PIE (Propagation, Infection and Execution) technique to analyze software testability [34]. Binder [7] discusses software testability based on six factors: representation, implementation, built-in text, test suite, test support environment and software process capability. Khoshgoftaar et *al.* address the relationship between static software product measures and testability [23, 24]. McGregor et *al.* [28] investigate testability of OOS and introduce the visibility component measure (VC). Bertolino et *al.* [6] investigate testability and its use in dependability assessment. Le Traon et *al.* [25, 26, 27] propose testability measures for data flow designs. Petrenko et *al.* [30] and Karoui et *al.* [21] address testability in the context of communication software. Sheppard et *al.* [31] focus on formal foundation of testability metrics. Jungmayr [20] investigates testability measurement based on static dependencies within OOS. Gao et *al.* [13] consider testability from the perspective of component-based software construction, and address component testability issues by introducing a model for component testability analysis [14]. Nguyen et *al.* [29] focus on testability analysis based on data flow designs in the context of embedded software. Baudry et *al.* address testability measurement (and improvement) of OO designs [3, 4, 5]. Metrics can, in fact, be used to locate parts of a program which contribute to a lack of testability. Bruntink et *al.* [8, 9] investigate factors of OOS testability and evaluate a set of well-known OOM with respect to their capabilities to predict testability of classes of Java software systems. Bruntink et *al.* investigate testability from the perspective of unit testing. Khan et *al.* [22] focus also on class level testability using OOM. Chowdhary [10] focuses on why it is so difficult to practice testability in the real world.

III. QUALITY ASSURANCE INDICATOR

In this section, we give a summary of the definition of the *Quality Assurance Indicator (Qi)* metric. The Qi metric is based on *control call graphs*, which are a reduced form of traditional *control flow graphs*. A *control call graph* is a *control flow graph* from which the nodes representing instructions, or basic blocs of sequential instructions, not containing a call to a method are removed. The Qi metric is normalized and gives values in the interval [0, 1]. A low value of the Qi of a class reflects that the class (is a high-risk class and) needs more testing effort to ensure its quality, while a high value indicates that (the class is a low-risk class knowing that) the testing effort invested effectively on the class is high (proportional to its complexity). The Qi of a class depends on the Qi of its collaborating classes (invoked classes).

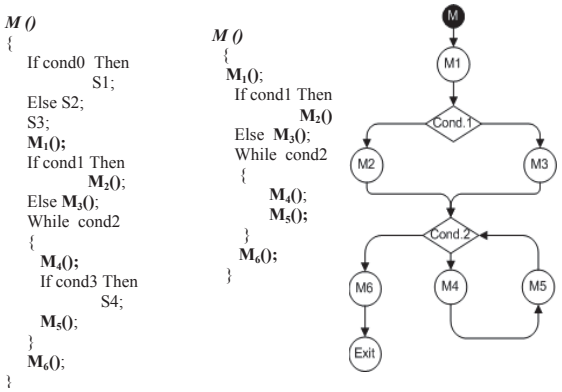


Figure 1 A method and its corresponding control call graph.

A. Control Call Graphs

Let us consider the example of method M given in Figure 1.1. The S_i represent blocs of instructions that do not contain a call to a method. The code of method M reduced to *control call flow* is given in Figure 1.2. The instructions (blocs of instructions) not containing a call to a method are removed from the original code of method M. Figure 1.3 gives the corresponding *control call graph*. Unlike traditional *call graphs*, *control call graphs* are much more precise models. They capture the structure of calls and related control.

B. Quality Assurance Indicator

We define the Qi of a method M_i as a kind of estimation of the probability that the control flow will go through the method without any *failure*. It may be considered as an indicator of the risk associated to a method (and a class at a high level). The Qi of a method M_i depends, in fact, on various intrinsic characteristics of the method itself, such as its unit

testing coverage (testing effort actually invested on the method) and its cyclomatic complexity, as well as on the Qi of all the methods invoked by the method M_i. We assume, in fact, that the quality of a method, particularly in terms of reliability, depends also on the quality of the methods it collaborates with to perform its task. In OOS, objects collaborate to achieve their respective responsibilities. A method of poor quality (lowly tested) can have (directly or indirectly) a negative impact on the methods that use it. There is here a kind of propagation, depending on the distribution of the control flow in a system, that needs to be captured. It is not obvious, particularly in the case of large and complex OOS, to identify intuitively this type of interferences between classes. The Qi of a method M_i is given by:

$$Qi_{M_i} = Qi^*_{M_i} * \sum_{j=1}^{n_i} \left(P\left(C_j^i\right) * \prod_{k \in \sigma} Qi_{M_k} \right) \quad (1)$$

with : Qi_{Mi}: quality assurance indicator of M_i, Qi^*_{Mi}: intrinsic quality assurance indicator of M_i, $P\left(C_j^i\right)$: probability of execution of path C_j^i of M_i, Qi_{Mk}: quality assurance indicator of the methods included in the path C_j^i, n_i: number of linear paths of the control call graph of M_i, and $Card(\sigma)=m_j$: number of the methods included in the path C_j^i. By applying the previous formula (1) to each method we obtain a system of N equations (N is the number of methods in the program). The obtained system is not linear and is composed of several multivariate polynomials. We use an iterative method (method of successive approximations) to solve it. The system is, in fact, reduced to a fixed point problem. Furthermore, we define the Qi of a class as the product of the Qi of its public methods. The calculation of the Qi metric is entirely automated by an Eclipse plug-in that we developed for Java software systems.

C. Assigning Probabilities

The *control call graph* of a method can be seen as a set of paths that the control flow can pass through. Passing through a particular path depends, in fact, on the states of the conditions in the control structures. To capture this probabilistic characteristic of the control flow, we assign a probability to each path C_k of a *control call graph* as follows:

$$P(C_k) = \prod_{i=0}^{n_k} P(A_i) \quad (2)$$

where A_i are the directed arcs composing the path C_k. By supposing, to simplify analysis and calculations, that the conditions in the control structures are independent, the $P(A_i)$ becomes the probability of an arc of being taken when exiting a control structure. The $P(C_k)$ are then reduced to a product of the probabilities of the states of the conditions in the control structures. To facilitate our experiments, we assigned probabilities to the different control structures of a Java program according to the rules given in TABLE I. These values are assigned automatically during the static analysis of the source code of a program when generating the Qi models. As an alternative way, the probabilities values would also be obtained by dynamic analysis, or assigned by programmers (knowing the code). Dynamic analysis will be considered in a future work.

TABLE I ASSIGNMENT RULES OF THE PROBABILITIES.

Nodes	Probability Assignment Rule
(if, else)	0.5 for the exiting arc « condition = true » 0.5 for the exiting arc « condition=false »
while	0.75 for the exiting arc « condition = true » 0.25 for the exiting arc « condition = false »
(do, while)	1 for the arc: (the internal instructions are executed at least once)
(switch,case)	1/n for each arc of the n cases.
(?, :)	0.5 for the exiting arc « condition = true » 0.5 for the exiting arc « condition = false »
for	1 for the arc
(try, catch)	0.75 for the arc of the « try » bloc 0.25 for the arc of the « catch » bloc
Polymorphism	1/n for each of the eventual n calls.

D. Intrinsic Quality Assurance Indicator

The *Intrinsic Quality Assurance Indicator* of a method M_i, noted Qi^*_{Mi}, depends on its cyclomatic complexity as well as on its unit testing coverage (as an indicator of the testing effort). It is given by:

$$Qi^*_{M_i} = \left(1 - \frac{F_i}{F_{\max}}\right) \quad (3)$$

with: $F_i = cc_i * (1 - tc_i)$, where : cc_i : cyclomatic complexity of the method M_i, and tc_i : unit testing coverage of the method M_i, $tc_i \in [0, 1]$.

$$F_{\max} = \max_{1 \le i \le N} \left(F_i\right)$$

Cyclomatic complexity can help software engineers determining the inherent risk of a program. Several studies provided empirical evidence that there is a significant relationship between cyclomatic complexity and fault proneness [1, 37]. Cyclomatic complexity is also recognized as a good indicator of testability [8, 9]. The more the cyclomatic complexity of a program is high, the more likely its testing effort would be high. Testing activities will reduce the risk of a complex program and achieve its quality. Moreover, testing coverage provide objective measures on the effectiveness of a testing process.

IV. EXPERIMENTAL DESIGN

The goal of this study is to explore empirically the relationship between the Qi metric and testability of classes in OOS. We evaluate the Qi metric at the class level, and limit the testing effort to the unit testing of classes. For our experiments, we selected from each of the used systems only the classes for which JUnit test cases exist. In this section, we present the metrics we used to quantify the testing effort required for a class by evaluating the corresponding JUnit test class, and describe the experimental design.

A. Metrics Related to Testability

To indicate the testing effort required for a software class (noted C_s), we used various metrics to quantify the corresponding JUnit test class (noted C_t). JUnit[1] is a simple framework for writing and running automated unit tests for

1 www.junit.org

Java Classes. Test cases in JUnit are written by testers in Java. A typical usage of JUnit is to test each class C_s of the program by means of a dedicated test class C_t. We used in our experiments each pair $<C_s, C_t>$ for classes for which test cases exist. The objective is to use these pairs to evaluate the relationship between the Qi metric and the measured characteristics of the test classes C_t. To capture testability of classes, we decided to measure for each test class C_t, corresponding to a software class C_s, various characteristics. We used the following suite of *test case metrics*:

- *TLOC*: This metric gives the number of lines of code of the test class C_t. This metric is used to indicate the size of the test suite corresponding to a class C_s.
- *TAssert*: This metric gives the number of invocations of JUnit *assert* methods that occur in the code of a test class C_t. The set of JUnit *assert* methods are, in fact, used by the testers to compare the expected behaviour of the class under test to its current behaviour. This metric is used to indicate another perspective of the size of a test suite. It is directly related to the construction of the test cases.
- *THEff*: This measure is one of the Halstead Software Science metrics [16]. This metric gives the effort necessary to implement or understand a test class C_t. It is proportional to the volume and to the difficulty level of the test class. We assume that this will reflect also the difficulty of the class under test and the effort required to construct the corresponding test class.

The approach used in this paper is based on the work of Bruntik et *al.* [8]. Two of the used test case metrics (*TLOC* and *TAssert*) have, in fact, been introduced by Bruntink et *al.* in [8, 9] to indicate the size of a test suite. Bruntink et *al.* based the definition of these metrics on the work of Binder [7]. We assume, in this paper, that these metrics are indicators of the testability of classes. These metrics reflect different source code factors as stated by Bruntink et *al.* in [8, 9]: factors that influence the *number of test cases required* to test the classes of a system, and factors that influence the *effort required to develop each individual test case*. These two categories have been referred as *test case generation factors* and *test case construction factors*. However, by analyzing the source code of the JUnit test classes of the systems we selected for our experiments, we feel that some characteristics of test classes are not captured by these two metrics (like the set of local variables or invoked methods). Since our work is exploratory in nature, we decided to extend the two metrics *TLOC* and *TAssert* by using the *THEff* metric to quantify the global effort necessary to implement a test class. We used this suite of metrics for characterizing the testing effort of classes. We assume that the effort necessary to write a test class C_t corresponding to a software class C_s is proportional to the characteristics measured by the used suite of test case metrics.

B. Data Collection

We calculated the values of the Qi (and Qi^*) metric for all classes for which JUnit test cases exist. We used the Eclipse plug-in we developed. For our experiments, we fixed the unit testing coverage to 75% for each of the methods of the analyzed systems. Furthermore, we evaluated other values of testing coverage. The obtained results were substantially the same (in a relative way). We also used the suite of test case metrics to quantify, for each of the subject systems, the JUnit test classes C_t that have been developed by the programmers using the JUnit framework. The test case metrics have been computed using the Borland Together tool.

C. Goal, Hypotheses and Statistical Analysis

We present, in this section, the methodology of the empirical study we conducted in order to assess the relationship between the Qi (and Qi*) metric and testability of classes. We performed statistical tests using correlation. The null and alternative hypotheses that our experiments have tested were:

- H_0: There is no significant correlation between the Qi metric and testability.
- H_1: There is a significant correlation between the Qi metric and testability.

In this experiment, rejecting the null hypothesis indicates that there is a statistically significant relationship between the Qi metric and test case metrics (the chosen significance level is α=0.05). For the analysis of the collected data, we preferred a non-parametric measure of correlation in order to test the correlation between the Qi (and Qi*) metric and the suite of test case metrics. We used the Spearman's correlation coefficient. This technique, based on ranks of the observations, is widely used for measuring the degree of linear relationship between two variables (two sets of ranked data). It measures how tightly the ranked data clusters around a straight line. Spearman's correlation coefficient will take a value between -1 and +1. A positive correlation is one in which the ranks of both variables increase together. A negative correlation is one in which the ranks of one variable increase as the ranks of the other variable decrease. A correlation of +1 or -1 will arise if the relationship between the ranks is exactly linear. A correlation close to zero means that there is no linear relationship between the ranks. We used the XLSTAT software to perform the statistical analysis.

V. EMPIRICAL STUDY

A. Selected Systems

The selected systems are : ANT (www.apache.org): a Java-based build tool, with functionalities similar to the unix "make" utility, and JFREECHART (http://www.jfree.org/jfreechart): a free chart library for Java platform.

TABLE II summarizes some characteristics of the analyzed systems : number of software classes, number of attributes, number of methods, number of lines of code, average value of lines of code, average value of cyclomatic complexity, percentage of tested classes (software classes for which JUnit test cases have been developed), number of JUnit test classes, and for software classes for which JUnit test classes have been developed : total number of lines of code, average value of lines of code and average value of cyclomatic complexity.

TABLE II SOME CHARACTERISTICS OF THE USED SYSTEMS.

	SOFTWARE CLASSES							TESTED SOFTWARE CLASSES			
SYSTEMS	#Classes	#Attributes	#Methods	LOC	MLOC	MWMPC	%TestedClasses	#TClasses	LOC	MLOC	MWMPC
ANT	713	2491	5365	64062	89.85	17.1	16.1%	115	17655	153.52	30.37
JFC	496	1550	5763	68312	137.73	28.09	46.37%	230	53131	231	46.08

The first observations that we can already make are: only a subset of software classes have been tested using JUnit, the pourcentage of tested classes varies from one system to another, and the software classes for which JUnit test cases have been developped are in general large and complex classes.

B. Results

For each pair <C_s, C_t> we analyzed the collected data set by calculating the Spearman's correlation coefficient r_s for each pair of metrics. TABLE III summarizes the results of the correlation analysis. It shows, for each of the subject systems and between each distinct pair of metrics the obtained values for the Spearman's correlation coefficient. We also calculated the Spearman's correlation coefficient r_s for each pair of test case metrics (TABLE IV). The obtained Spearman's correlation coefficients that are significant (at α=0.05) are set in boldface in the two tables. This means that for the corresponding pairs of metrics there exist a correlation at the 95 % confidence level.

TABLE III CORRELATION VALUES BETWEEN THE Q_i AND Q_i* METRICS AND TEST CASE METRICS.

ANT	TLOC	TAssert	THEff
Qi	**-0.560**	**-0.326**	**-0.448**
Qi*	**-0.586**	**-0.393**	**-0.523**

JFC	TLOC	TAssert	THEff
Qi	**-0.425**	**-0.365**	**-0.353**
Qi*	**-0.447**	**-0.458**	**-0.439**

TABLE IV CORRELATION VALUES BETWEEN TEST CASE METRICS.

ANT	TLOC	TAssert	THEff
TLOC	**1**	**0.679**	**0.900**
TAssert		**1**	**0.781**
THEff			**1**

JFC	TLOC	TAssert	THEff
TLOC	**1**	**0.848**	**0.922**
TAssert		**1**	**0.896**
THEff			**1**

The first global observation that we can make is that the obtained results confirm that there is a significant relationship between the Qi and Qi* metrics and the used test case metrics. The obtained Spearman's correlation coefficients between the Qi and Qi* metrics and the test case metrics are all significant (at α=0.05) for the two selected systems (for all the pairs of metrics). We can reject the hypothesis H_0 and accept the hypothesis H_1. Moreover, the measures have negative correlation. As mentioned previously, a negative correlation indicates that the ranks of one variable (Qi and Qi* values in our case) decrease as the ranks of the other variable (test case metric) increase. These results are plausible knowing that the more classes (and methods) are complex, the more they are difficult to test and their Qi (and Qi*) values decrease. The second global observation that we can make is that the Qi* metric is, overall, better correlated to the test case metrics than the Qi metric. This may be explained by the fact that the metric Qi* takes into account only the inherent characteristics of methods (and classes) compared to the metric Qi which take into account the dependencies between methods (and classes). The other global observation that we can make is that the test case metrics are also correlated between themselves (TABLE IV).

C. Limitations

The study performed in this paper should be replicated using many other systems in order to draw more general conclusions about the relationship between the metrics Qi and Qi* and testability. In fact, there are a number of limitations that may affect the results of the study or limit their interpretation and generalization. The obtained results are based on the data set we collected from the analyzed systems. To collect data we only used a subset of classes (and corresponding JUnit test cases) from each of the subject systems. From TABLE II we can see that for system ANT we used only 115 software classes and corresponding JUnit test cases, and for system JFREECHART we used 230 software classes and corresponding JUnit test cases. In total, we analyzed 345 software classes and corresponding JUnit test classes. Even if we believe that the analyzed set of data is enough large to allow obtaining significant results, the study should be, however, replicated on a large number of OOS to increase the generality of the results. Moreover, we can also observe from TABLE II that the classes for which JUnit test cases have been developed are relatively large and complex classes. This is true for the two subject systems. This may affect the results of our study in the sense that depending on the methodology followed by the developers while developing test classes and the criteria they used while selecting the software classes for which they developed test classes (randomly or depending on their size or complexity for example, or on other criteria) the results may be different. It would be interesting to replicate this study using systems for which JUnit test cases have been developed for a maximum number of classes. By analyzing the source code of the JUnit test classes, we observed that, in many cases, they do not cover all the methods of the corresponding software classes. This may also affect the results of the study. It is also possible that facts such as the development style used by the developers for writing test cases might affect the results or produce different results for specific applications.

VI. CONCLUSIONS AND FUTURE WORK

We presented, in this paper, a metric capturing in an integrated way different attributes of OOS. The metric, called *Quality Assurance Indicator*, uses control flow paths and probabilities, and captures the collaboration between classes. The paper investigated empirically the relationship between the proposed metric and testability of classes. Testability has

been investigated from the perspective of unit testing. As a first attempt, we designed and performed an empirical study on two open source Java software systems, for which JUnit test cases exist. We used three metrics for characterizing the JUnit test classes. We performed statistical tests using correlation. The achieved results support the idea that there is a statistically and practically significant relationship between the Qi and Qi* metrics and the used test case metrics.

The performed study should, however, be replicated using many other systems in order to draw more general conclusions. The findings in this paper should be viewed as exploratory and indicative rather than conclusive. Moreover, knowing that software testability is affected by many different factors, it would be interesting to extend the used suite of test case metrics to better reflect the testing effort. We hope, however, this study will contribute to a better understanding and characterizing of what contributes to testability of classes in OOS. As future work, we plan: to extend the used test case metrics to better reflect the testing effort, to extend the study by using some well-known OOM, and to replicate the study on other projects to be able to give generalized results.

ACKNOWLEDGEMENTS

This project was financially supported by NSERC (National Sciences and Engineering Research Council of Canada).

REFERENCES

[1] Aggarwal, K.K., Yogesh, S., Arvinder, K., and Ruchika, M., "Empirical analysis for investigating the effect of object-oriented metrics on fault proneness": A replicated case study, Software Process: Improvement and Practice, 16 (1), 2009.

[2] Badri, M., Badri, L., Toure, F., "Empirical Analysis of Object-Oriented Design Metrics : Towards a new metric using control flow paths and probabilities", JOT, vol. 8(6), 2009.

[3] Baudry, B., Le Traon, B., Sunyé, G., "Testability analysis of a UML class diagram", 9th International Software Metrics Symposium *(METRICS'03)*, IEEE Computer Society, 2003.

[4] B. Baudry, B., Le Traon, Y., Sunyé, G., Jézéquel, J.M., "Measuring and Improving Design Patterns Testability", Proceedings of the 9th International Software Metrics Symposium (METRICS), IEEE Computer Society, 2003.

[5] Baudry, B., Le Traon, Y., Sunyé, G., "Improving the Testability of UML Class Diagrams", Proceedings of *IWoTA* (International Workshop on Testability Analysis), Rennes, France, 2004.

[6] Bertolino, A., Strigini, L., "On the Use of Testability Measures for Dependability Assessment", IEEE Transactions on Software Engineering, Vol. 22, NO. 2, February 1996.

[7] Binder, R.V., "Design for Testability in Object-Oriented Systems", Communications of the ACM, Vol. 37, 1994.

[8] Bruntink, M., Deursen, A.V., "Predicting Class Testability using Object-Oriented Metrics", 4th Int. Workshop on Source Code Analysis and Manipulation (SCAM), IEEE, 2004.

[9] Bruntink, M., Deursen, A.V., "An empirical study into class testability", Journal of Systems and Software, 2006.

[10] Chowdhary, V., "Practicing Testability in the Real World", International Conference on Software Testing, Verification and Validation, IEEE Computer Society Press, 2009.

[11] Fenton, N., Pfleeger, S.L., "Software Metrics: A Rigorous and Practical Approach", PWS Publishing Company, 1997.

[12] Freedman, R.S., "Testability of Software Components", IEEE Transactions on Software Engineering, Vol. 17(6), June 1991.

[13] Gao, J., Tsao, J., Wu, Y., "Testing and Quality Assurance for Component-Based Software", Artech House Publishers, 2003.

[14] Gao, J., Shih, M.C., "A Component Testability Model for Verification and Measurement", COMPSAC, IEEE, 2005.

[15] Gupta, V., Aggarwal, K.K., Singh, Y., "A Fuzzy Approach for Integrated Measure of Object-Oriented Software Testability", Journal of Computer Science, Science Publications, 2005.

[16] Halstead, M. H., "Elements of Software Science", Elsevier/North-Holland, NY, 1977.

[17] Henderson-Sellers, B., "Object-Oriented Metrics Measures of Complexity", Prentice-Hall, 1996.

[18] IEEE, 1990. IEEE Standard Glossary of Software Engineering Terminology, IEEE Computer Society Press, NY, 1990.

[19] ISO/IEC 9126: Software Engineering Product Quality, ISO Press, 1991.

[20] Jungmayr, S., "Testability Measurement and Software Dependencies", Proceedings of the 12th International. Workshop on *Software Measurement*, October 2002.

[21] Karoui, K., Dssouli, R., "Specification transformations and design for testability", Proc. of the IEEE Global telecommunications Conference (GLOBECOM'96), 1996.

[22] Khan, R.A., Mustafa, K., "Metric Based Testability Model for Object-Oriented Design (MTMOOD)", ACM SIGSOFT Software Engineering Notes, vol. 34, no. 2, March 2009.

[23] Khoshgoftaar, T.M., Szabo, R.M., "Detecting Program Modules with Low Testability", 11th ICSM, 1995.

[24] Khoshgoftaar, T.M., Allen, E.B., Xu, Z., "Predicting Testability of Program Modules Using a Neural Network", 3rd IEEE Symp. on Application-Specific Systems and SE Technology, 2000.

[25] Le Traon, Y. and Robach, C., "Testability analysis of co-designed systems", Proc. of the 4th Asian Test Symposium, ATS. IEEE Computer Society, Washington, DC, 1995.

[26] Le Traon, Y., Robach, C., "Testability Measurements for Data Flow Design", Proceedings of the Fourth International Software Metrics Symposium, New Mexico, November 1997.

[27] Le Traon, Y., Ouabdessalam, F., Robach, C., "Analyzing testability on data flow designs", ISSRE'00, San Jose, 2000.

[28] McGregor, J., Srinivas, S., "A measure of testing effort, Proc. of the Conference on Object-Oriented Technologies", pages 129-142. USENIX Association, June1996.

[29] Nguyen, T.B., Delaunay, M., Robach, C., "Testability Analysis Applied to Embedded Data-Flow Software", Proc. of the 3rd International Conference on Quality Software (QSIC'03), 2003.

[30] Petrenko, A., Dssouli, R., and Koenig, H., "On Evaluation of Testability of Protocol Structures", IFIP, Pau, France, 1993.

[31] Sheppard, J.W., Kaufman, M., "Formal Specification of Testability Metrics" in IEEE P1522, *IEEE AUTOTESTCON,* Pennsylvania, August 2001.

[32] Voas, J.M., PIE: "A dynamic failure-based technique", IEEE TSE, 18(8), August 1992.

[33] Voas, J., Miller, K.W., "Semantic metrics for software testability", Journal of Systems and Software, Vol. 20, 1993.

[34] Voas, J.M., Miller, K.W., "Software Testability: The New Verification", IEEE Software, 12(3), 1995.

[35] Yeh, P.L., Lin, J.C., "Software Testability Measurement Derived From Data Flow Analysis", 2nd Euromicro Conference on Software Maintenance and Reengineering, Italy, 1998.

[36] Zhao, L., 2006. "A New Approach for Software Testability Analysis", 28th ICSE, May 2006.

[37] Zhou, Y., Leung, H., "Empirical analysis of object-oriented design metrics for predicting high and low severity faults", IEEE Trans. on software engineering, vol. 32, no. 10, 2006.

EMPIRICAL STUDY UPON SOFTWARE TESTING LEARNING WITH SUPPORT FROM EDUCATIONAL GAME

Marcello Thiry, Alessandra Zoucas and Antônio C. da Silva
Master's degree course in Applied Computing
Universidade do Vale do Itajaí - UNIVALI
Florianópolis, Brazil
{thiry, azoucas, antonio.carlos}@univali.br

Abstract— **Although software testing is seen as one of the main quality measurements of software, test practices and techniques are still seldom applied by software development companies. One possible reason to this scenario is the lack of skilled and available professionals to implements such techniques and practices. The teaching of software testing is typically approached as a topic of Software Engineering lectures. In this context, one of the main challenges is to allow the student to apply the concepts seen in classroom. The developed educational game aims to allow the students to practice through experimenting concepts, techniques and practices of software testing in a simulated environment. To evaluate the game contribution, we present the results of two experiments performed with undergraduate students. In the first experiment the learning effectiveness was compared to students who did not play the game. In the second experiment, the learning effectiveness was compared to a traditional pencil-and-paper exercise.**

Keywords: Software testing; educational game; learning; Experimental Software Engineering.

I. INTRODUCTION

The quality control and assurance in software projects consist in activities and techniques that, when systematically applied, permit to assess the chances of success of a software project concerning the fulfillment of the expectations of internal and external clients [1]. Models as CMMI [2] include concepts and practices to increase capacity and maturity of an enterprise's processes [3]. Lined up with the models and standards of process, Software Engineering (SE) gathers activities and techniques that excel by anticipating, in a systematic, organized and controlled way, the identification of nonconformities in software. These activities and techniques are called Verification and Validation (V&V).

Although software testing is present in the developing process, researches show that its techniques and practices are not totally employed by the software development organizations [4] [5]. In the research performed in [4] it was evidenced that, even though the organizations admit they apply software tests, almost half (48,5%) of software testing practices are regarded as not applied and not important. Among the practices regarded as not applied and not important are all those related to measurement and analysis, including test coverage [6]. Considering these results, we can conclude there are a great amount of practices regarded as not important and not applied, regardless how big the companies are [4]. The reasons for that include the lack of knowledge about the practices, lack of skilled and available human resources to its implement, lack of support of high level managers, lack of approaches, among others.

The results obtained in [4] confirm the results of another study performed in 2004 that assessed software testing in software development enterprises [6]. It was observed a great distance between what is produced in the academic environment and what is put into practice by software enterprises. The results allow verifying how immature the assessed enterprises are, concerning software testing.

One of the challenges of teaching SE is the necessity to assure that the student acquires enough experience in applying the concepts through laboratory practices [7]. As an alternative to the practical approach in Software Engineering, an experiment performed with Computer Science undergraduate students explored the possibility of having the students working on a real software project [8]. Before the experiment application, it was verified that the SE students from the third semester had little capacity to remember concepts already approached and the possible cause was the little practice the students had. The experiment main goal was the application of concepts and techniques of Software Engineering [8]. The result identified greater motivation and interest from the participants in the SE subject.

Other approach to the practice in SE teaching refers to the application of educational games [9][10][11][12]. However, it is possible to verify a small number of experiments performed to verify the games learning effectiveness [9][10][11][12]. In this context, this paper intends to promote learning software testing techniques and practices through an educational.

The next section presents researches related to this work. Section III presents the developed game "U-TEST". The details of the game learning effectiveness assessment are presented in section IV. Section V presents a discussion on the results and the conclusions of this paper.

II. Related Work

The analysis of similar solutions, besides the definition of the assessment criteria was based in a search using different information sources: Google Scholar, CiteSeer, IEEExplorer and ACM Digital Library. At first, specific games to support Software Engineering teaching were sought, using a protocol containing terms such as: *software testing game learning, software testing game based learning, software testing teaching approach*, among others. As a result, it was observed the absence of games in software testing. Therefore, games to support Software Engineering teaching in general were chosen. In this new search, seven games were found: (1) **SimSE** [9], (2) **SE•RPG** [10], (3) **TIM - The Incredible Machine** [13], (4) **Planager** [14], (5) **SESAM** [11], (6) **X-MED** [12] e (7) **SimulES** [15].

The games were analyzed under seven criteria: (1) It has the definition of educational goals; (2) Game genre: action, adventure, puzzle, simulation, strategy, etc.; (3) It gives a feedback to the students about their performance; (4) The game is available for free use; (5) There is game learning effectiveness assessment; (6) Game platform: non digital, web, desktop, etc. ; and (7) Field of knowledge on SE.

Characterization of the works assessed uses the following description: (T) Totally present; (P) Partially present and (N) Not present. Table I presents criteria and assessment results.

TABLE I. Assessed Games Result

	SimSE	SE•RPG	TIM	Planager	SESAM	X-MED	SimulES
1	T	P	P	P	T	T	P
2	Simulation	Simul. RPG	Simulation	Simulation	Simul Advent	Simulation	Simulation
3	T	T	P	T	T	T	T
4	T	T	T	T	T	T	T
5	T	T	T	N	T	T	P
6	desk top	web	desk top	desk top	desk top	desk top	not digital
7	Project Mgmt	Project Mgmt	Project Mgmt	Project Mgmt	Project Mgmt	Softw. measu rem.	Softw. pro cess

There is a concern in most studies according to the importance of experiments to the game learning effectiveness assessment, although they was not applied to all assessed games. Based on the results, little can be concluded about the game learning effectiveness and it continues to be a topic to be explored in future studies [10]. It was also observed that most of the games focus on project management or general Software Engineering. Thus, specific games in software testing may increase the didactic content and offer a complement to traditional capacitating.

III. The Educational Game "U-TEST"

This section presents the educational game "U-TEST" developed to assist software testing teaching. The game has an instructional design based on ADDIE model [16] and it was designed to support Software Engineering students. It is expected the students have basic concepts of programming, Software Engineering and software testing. The educational goals were based on the Bloom's revised taxonomy [17] and comprehend the Remember, Understand and Apply levels. The game goals are: (1) Recognize and Understand the main concepts of software testing in a general way and (2) Understand and apply the techniques of data entry selection, equivalence class partitioning and boundary-value analysis.

"U-TEST" is a simulation game to support software testing with focus on unit tests and black box techniques, approaching theoretical and practical questions. The game is based on a case where the player is seen as a candidate to a position in a software company. After an interview the player must solve challenges to prepare unit test cases. The game presents brief comments about the company and the project the player will take part. Next, the player must build the test cases to the presented functions. As a result of their performance, the player is informed about his position on the players' ranking. The challenges proposed by the game focus mainly on the data entry selection, using a combination of equivalence classes partition and limit value analysis. It allows the player to identify the necessity of application of test techniques and practices. Figure 2 presents the screen of the first challenge of the game.

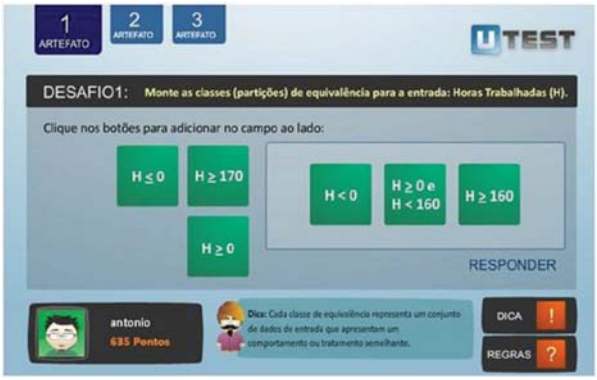


Figure 2- Screen of the "U-TEST" game.

During the game, the player will undergo ten stages, and in six of them there are the following challenges: 1) Presentation of the artifact; 2) Setup the equivalence classes; 3) Define limit values for the identified classes; 4) Select the correspondent value to the identified value; 5) Setup a cause-effect graph or decision tree; and 6) Project final feedback. The feedback is provided by a graphic indicator placed on the interface and at the end of each challenge. The player is informed about their performance and its indicator is updated. At the end of the game, the player is informed about their general performance and position on the players' ranking.

IV. Learning Assessment

A. The Empirical Study

During this study, two research questions were defined: 1) Is the learning effect on the remembering, understanding and applying level in the group of students that played the

game higher than in the group that didn't play?; 2) Is the educational game considered appropriate in terms of content relevancy, correctness, sufficiency and degree of difficulty, sequence, teaching method and duration in the context for which it is intended? Is the game considered engaging?

Based on the first research question, it was established the following hypotheses:

- **H_0**: There is no significant difference in relative learning effectiveness between group A (experimental group) and group B (control group).
- **H_1**: There is significant difference in relative learning effectiveness between group A and group B.

These hypotheses were assessed from a statistical test whereas the second research question was assessed from the qualitative assessment based on the questionnaire answered by students.

Two experiments were performed, the first one involving Computer Science undergraduate students and the second one involving Information Technology undergraduate students. Each experiment was based on the proposal defined by Kochanski [18]: 1) the assessment was designed considering all the contents fulfilled in the game; 2) the students signed a term showing interest in taking part on the experiment; 3) the students filled a questionnaire about their experience and previous knowledge to establish each student and group backgrounds; 4) all students attended theoretical lectures about software testing before playing the game; 5) all students performed a *pretest* measuring their knowledge after the lectures; 6) the students were randomly partitioned into two groups in a balanced manner, one experimental group (A) and other control group (B); 7) in the first experiment, the group A played the game and the group B played another game with no relation to Software Testing (this game was considered a placebo). In the second experiment, the group B took part in a traditional pencil-and-paper exercise with problems to solve on the same content explored in the game; 8) all students performed a *posttest* measuring again their knowledge after the previous step; 9) using the data collected from pre and posttest, hypothesis tests were conducted to verify any improvement on student's performance after playing the game; 10) all students in the group A answered questions related to their perception of the game (this questionnaire served as a base to a qualitative assessment of the game).

The pre and posttest questions (steps 5 and 8) were developed based on the Bloom's revised taxonomy [17] and comprehend the Remember, Understand and Apply levels.

B. Hypothesis Testing and Data Analysis

In this series of experiments, our principal concern for accuracy and to overcome problems with statistical power is due to the very small sample size. For such small data sets, it is basically impossible to tell, if the data come from a variable that is normally distributed [19], as with small sample sizes ($n < 20$), tests of normality may be misleading. Unfortunately, with small samples, parametric tests lack statistical power and it may be almost impossible to generate a p-value of < 0.05, whatever the differences between the groups of sample data. But, on the other side, nonparametric tests are not robust. However, inspecting the data distribution we could not assume a normal distribution of the variables. Therefore, we used non-parametric test (one-tailed Mann-Whitney U) as it is considered the most powerful nonparametric alternative to the t-test for independent samples. Due to the small samples size ($n<20$), we also did not use a z-value to approximate the significance level for the test, but compared the minimum U to tabellized U values [19].

The first experiment was performed with ten Computer Science undergraduate students. They are independently and randomly sorted into two groups, the first of size $n_A = 5$ (experimental group) and the second of size $n_B = 5$ (control group). The students of the group A played the game "U-TEST", while those of the group B played another game with no relation to Software Testing (this game was considered a placebo). Both groups performed a pretest (before the treatment) and a posttest (after the treatment). Each test has 20 questions distributed according to the Remember, Understand and Apply levels (Bloom's revised taxonomy).

The first step was to assembly the measures (each measure represents the difference between the student pretest and posttest scores) from groups A and B into a single set of size $N = n_A + n_B = 10$. These measures were then rank-ordered from lowest (#1) to highest (#N). When measure entries are tied for ranks, each measure receives the average of those ranks. After they have been sorted out, the rankings are then returned to the group, A or B, which they belong and substituted for the original measures that gave rise to them.

The next step was to calculate w_A (the sum of the n_A ranks in group A) e w_B (the sum of the n_B ranks in group B). In this experiment, $w_A = 40$ e $w_B = 15$. Using the Mann-Whitney U formula, we calculated $U_A = w_A - n_A (n_A+1)/2 = 40 - 5*(5+1)/2 = 25$ e $U_B = w_B - n_B (n_B+1)/2 = 15 - 5*(5+1)/2 = 0$. Considering the smaller value of U, we obtained $U_{obt}=0$. Then, we consulted the critical value $U_{crit} = 2$ for the Mann-Whitney Test (U) considering $n_A = 5$ and $n_B = 5$ with the level of significance 0.05. According to the Mann-Whitney Test, once $U_{obt} <= U_{crit}$ ($0 <= 2$) is true, **we could reject H_0**.

The second experiment was performed with ten Information Technology undergraduate students. They are also independently and randomly sorted into two groups, the first of size $n_A = 6$ (experimental group) and the second of size $n_B = 7$ (control group). The students of the group A played the game "U-TEST", while those of the group B took part in a traditional pencil-and-paper exercise with problems to solve on the same content explored in the game. Both pretest a posttest were the same of the first experiment.

In this case, we have $N = n_A + n_B = 13$. After following the same procedure described for the first experiment, we calculate $w_A = 44$ e $w_B = 47$. We also obtained $U_A = 23$ e $U_B = 19$. Considering the same level of significance 0.05, we have $U_{crit}=7$. Once $U_{obt} <= U_{crit}$ ($19 <= 7$) is false, **we could not reject H_0**.

To assess the second research question, we applied a perception questionnaire with students of the experimental groups. In both experiments, students answered they liked the game and felt motivated, besides they thought the game contributes to learning. Based on these results, we verify that the game, besides promoting a significant improvement at the students' scores in the first experiment, was positively evaluated by them. In the second experiment, although it was not possible to confirm higher learning effectiveness in relation to a traditional pencil-and-paper exercise, the game was also considered more motivating that the exercise. It may be an indication that, even when educational games are not superior to the traditional exercises, the students' motivation may be a decisive factor to long term learning. A question to be assessed in the future is: if the students who played the game had a greater capacity to keep the knowledge than the ones who only did the exercise.

V. CONCLUSION

The experiments were conducted in a systematic and documented way, establishing a formalized assessment that can be repeated countless times. This repetition allows the comparison with previous assessments, offering better conditions to analyze historical results and identify positive/ negative learning tendencies. Although the number of experiments is still reduced, as well as the number of participants, the positive results (effects with statistical validity) are a stimulus to the continuation of this research. The results also helped to identify strong and weak points of the game and it will guide its development in the future. Based on the feedback obtained, the game is already been improved, mainly to increase its difficulty levels and variability of directions. Moreover, we intend to increase the number of software testing topics considered by the game.

An initial concern was the comparison of the game with a traditional pencil-and-paper exercise, once the game is still very direct and based on problems solution. However, according to the feedback received from the qualitative questionnaires, we verify that the game is a motivational differential, increasing interest and raising greater curiosity on knowledge. As a continuation of this study, researchers have been developing and experiment educational games in other areas of Software Engineering. Currently, games on Project Management, Requirements Engineering and Software Improvement Process are being studied.

The results obtained so far, even under the statistic rigor, cannot be generalized. Among the main threatens to the assessment, it must be considered that the experiments were performed inside two institutions only, where teachers are part of the research group. This way, it is necessary to widen the set of experiments and number of students so that it will be possible to assess learning tendencies with more accuracy. Other threaten was the inequality on the students knowledge and experience in Software Testing students. However, it was treated with a previous evaluation of the profile questionnaire answered by the students before the experiments. In this case, no student who could have been a threat to the obtained results was found.

REFERENCES

[1] J. Tian, "Software Quality Engineering: Testing, Quality Assurance, and Quantifiable Improvement," Hoboken: John Wiley & Sons. Los Alamitos: IEEE Computer Society Press, 2005.

[2] SEI – Software Engineering Institute, "CMMI for Development, Version 1.3," Pittsburgh: SEI, Carnegie Mellon University, 2010.

[3] B. Mutafelija, and H. Stromberg, "Process Improvement with CMMI V1.2 and ISO Standards," CRC Press, 2008.

[4] A. R. C. da Rocha, A. C. Dias-neto, A. C. C. Natali, and G. H. Travassos, "Caracterização do estado da prática das atividades de teste em um cenário de desenvolvimento de software brasileiro," In: simpósio Brasileiro de Qualidade de Software, 5., 2006, Vila Velha. Vila Velha: SBC, 2006. p. 27-41.

[5] A. Bertolino, "The (Im)maturity level of software testing," ACM SIGSOFT Software Engineering Notes. New York, v. 29, n. 5, p. 1-4, set. 2004.

[6] H. Zhu, P. Hall, and J. May, "Software unit test coverage and adequacy," ACM Computing Surveys. New York, v. 29, n. 4, p. 366-427, dez. 1997.

[7] CEEInf-MEC, "Diretrizes curriculares de cursos da área de computação e informática," accessed at 2008 august: <http://www.mec.gov.br>.

[8] M. Gnatz, L. Kof, F. Prilmeier, and T. Seifert, "A Practical Approach of Teaching Software Engineering," In: Conference on Software Engineering Education and Training - CSEE&T, 16., 2003, Madrid. Proceedings… Madrid: IEEE, 2003. p. 120-128.

[9] E. Navarro, "SimSE: a Software Engineering simulation environment for software process education," 321 p. Dissertation (Doctor of Philosophy in Information Computer Science) – University of California, Irvine, 2006.

[10] F. B. V. Benitti, and J. S. Molléri, "Utilização de um RPG no ensino de gerenciamento e processo de desenvolvimento de software," In: Workshop sobre educação em computação - WEI, 16., 2008, Pará SBC/UFPA, 2008. p. 258-267.

[11] J. Ludewig, "Models in Software Engineering – an introduction," In Software and Systems Modeling 5-14. Springer Berlin / Heidelberg. Volume 2, Number 1 / March, 2003.

[12] C. G. von Wangenheim, M. Thiry, and D. Kochanski, "Empirical evaluation of an educational game on software measurement," Empirical Software Engineering, v. 14, n. 4, p. 418-452, ago. 2009.

[13] A. Dantas, M. Barros, and C. Werner, "Treinamento experimental com jogos de simulação para gerentes de projeto de software," In: Simpósio Brasileiro de Engenharia de Software – SBES, 18, Brasília: SBC/UNB, 2004. p. 23-38.

[14] R. Rosa, and E. Kieling "Planager - Um Jogo para Apoio ao Ensino de Gerência de Projetos de Software," TCC Bacharelado em Sistemas de Informação PUC RS, 2006.

[15] E. Figueiredo, C. Lobato, K. Dias, J. Leite, and C. Lucena, "Um jogo para o ensino de engenharia de software centrado na perspectiva de evolução," XV Workshop sobre Educação em Computação (WEI), Rio de janeiro, 2007, pp. 37-46.

[16] M. Molenda, "The ADDIE model," In Kovalchick, A. Dawson, K. (Eds.), Educational Technology: An Encyclopedia. 201-215. Santa Barbara: ABC- CLIO, 2003.

[17] L. W. Anderson, and D. R. Krathwohl, (Eds.). "A taxonomy for learning, teaching, and assessing: a revision of bloom's taxonomy of educational objectives," New York: Longman, 2001.

[18] D. Kochanski, "Um framework para apoiar a construção de experimentos na avaliação empírica de jogos educacionais," Dissertação apresentada à Universidade do Vale do Itajaí como requisito para a obtenção do título de Mestre em computação. São José, Brasil. 2009.

[19] J. Levin, and J. Fox, "Elementary Statistics in Social Research," Allyn & Bacon, 2006.

A Study on Performance Inconsistency between Estimation by Analogy and Linear Regression

Sousuke Amasaki
Okayama Prefectural University
111 Kuboki Soja, Okayama, Japan 719–1197
Email: amasaki@cse.oka-pu.ac.jp

Abstract

Background: *Many comparative studies have been performed on effort estimation models. Linear regression (LR) and Estimation by Analogy (EbA) were often compared. The past research revealed that those comparative studies reported inconsistent results among performance measures. However, those studies seemed not to reflect actual or desirable study procedure.* ***Aim:*** *We aimed to examine performance inconsistency in comparative study on LR and EbA under more desirable procedure.* ***Method:*** *We carefully determined datasets and experiment procedure. LR and EbA were then compared under appropriate condition.* ***Results:*** *Performance measures showed statistically consistent results in almost all datasets.* ***Conclusion:*** *Comparative study could show consistent results with suitable experiment procedure.*

1. Introduction

Software effort estimation is popular and important research area. Model-based effort estimation has been very studied in this area. Many software estimation models have been proposed showing its superior predictive performance to that of old or popular models.

Linear regression (LR) and Estimation by Analogy (EbA) are the most popular methods among them [6] and repeatedly compared. Comparative studies on them often reported inconsistent results regarding predictive performance. Myrtveit et al. [10] revealed using simulation that performance measures favored different models in a comparative study on LR and EbA. Mair et al. [7] revealed with systematic review that predictive performance was inconsistent between and within the past comparative studies on LR and EbA even if the same dataset and performance measures were used.

Revealed interesting knowledge, these studies seemed not to reflect actual or desirable comparative study. Myrtveit et al. compared 7 performance measures with 1000 simulated datasets. However, those datasets were generated using a single dataset. Furthermore, they selected "best results" for EbA after performing evaluations on different parameter settings. Thus, their result was not generalized for usual comparative study. The past comparative studies reviewed in [7] actually performed with a few dataset or a few performance measures. It has been criticized to insist validity of new proposed models by an experiment with a few datasets. Performance evaluation with only MMRE and PRED(25), for instance, was not enough because MRE-based measures were biased and tend to penalize overestimation more seriously. Furthermore, in contrast to [10], they did not control potential sources of inconsistency such as dataset, experiment procedure, and options like variable selection methods. In addition, clear dataset selection criteria were rarely found. We believe that controlled and recommended conditions were important for comparative study to discuss sources of inconsistent results.

This paper thus conducted comparative study on LR and EbA with not a few datasets and performance measures. We carefully selected datasets suitable for experiment procedure and performance measures. As a result, we found that performance measures could be consistent in almost all datasets. This result can contribute for discussing and investigating sources of inconsistent results.

2. Experiment Settings

2.1. Performance Measures

This study used the same performance measures as [10]: Mean MAE (MMAE), MMRE, Median MRE (MdMRE), PRED(25), MMER, RSD, and LSD. MAE, MRE, and MER-based measures are very popular ones. RSD and LSD

are defined as follows:

$$\mathrm{RSD} = \sqrt{\frac{\sum(\frac{\mathrm{Act}_i - \mathrm{Est}_i}{\mathrm{Size}_i})^2}{n-1}},$$

$$\mathrm{LSD} = \sqrt{\frac{\sum(e_i - (-\frac{s^2}{2}))^2}{n-1}}.$$

Here, s^2 is an estimator of the variance of the residual e_i, where e_i is given by $e_i = \ln \mathrm{Act}_i - \ln \mathrm{Est}_i$.

2.2. Experiment Procedure

Cross-validation (CV) has been popular experiment procedure in comparative study. Two types of CV have been often used: leave-one-out CV and K-fold CV. Leave-one-out CV allows us use smaller datasets than K-fold CV. It was also recommended because of its deterministic property.

However, we adopted K-fold CV for two reasons related to evaluation reliability. First, leave-one-out CV may lead unreliable estimates because an estimate from it has high variance [3]. Second, leave-one-out CV produces identical values for average-based and median-based measures such as MMRE and MdMRE. This is because a fold of leave-one-out CV has only one project and performance measures for a test subset are calculated from single estimate. In case of K-fold CV, we can avoid this situation if a test subset has more than 2 test cases.

Eventually we adopted 10 $\times$ 10-fold CV followed by paired t-test with 10 degree-of-freedom in order to avoid inflated Type I error [2]. Comparative studies often adopted nonparametric test for the reason that distributions of a performance measure were skewed. In fact, skewness was not a problem because for most of the distributions one encounters in practice, the significant level of the t-test is almost exact for sample sizes greater than 12 [4].

2.3. Outlier Elimination

We removed influential data points (outliers) by Cook's distance [9] in advance so that all projects had Cook's distance lower than $3\times4/N$, where N is the number of projects in a dataset. In contrast to [9], this study left projects with distance higher than $4/N$ but smaller than $3 \times 4/N$. This is because decision of inclusion or exclusion for these projects depends on model-based judgment and it was difficult when two different models were compared.

3. Effort Estimation Models

3.1. Linear Regression

In many situations, a fitted regression model is likely to be reliable when the number of predictors p is less than $N/10$ or $N/20$. Following this rule, sample size N must be $N > \frac{100p}{9}$ in case of 10$\times$10-fold CV. This is because $p < 1/10 \cdot 9/10 \cdot N$ must hold.

Linear regression can utilize multiple predictors in order to improve predictive performance. However, simple models including only size-related metric as a predictor are also popular because it is easy to construct and to be able to avoid variations in an experiment. Myrtveit et al. revealed inconsistent results in [10] even if simple models were used. The size of dataset must be equal or more than 12 for simple models because of $p = 1$. In order to include more datasets, we determined to adopt simple models. This study adopted the following popular formulation:

$$\log(\mathrm{Effort}) = \beta_0 + \beta_1 \log(\mathrm{Size}).$$

3.2. Estimation by Analogy

In EbA estimation process, similarity between a target project and historical projects is calculated from recorded metrics and then some projects similar to that target project are selected. EbA estimates effort of that target project from the similar projects.

EbA has several options for fitting to a specific dataset. This study adopted the following options:

- Size-related metric for only predictor
- Euclidean-distance as similarity function
- Geometric mean as projection method

This EbA tried to estimate p50 Effort [5]. Geometric mean is identical to median under log-normal distribution. It also mitigates effect of extremely large effort under a distribution skewed right.

This study determined the number of similar projects k before an experiment by 10-fold CV which minimizes an average of MMREs. This was because EbA showed better result than LR only with k minimizing MMRE [10] and thus we expected to find inconsistency.

4. Datasets

4.1. Source

We performed comparative study with some datasets listed in [8] and served on PROMISE repository [1] at January 2011. We used only public datasets because they are freely available and replication can be open.

Table 1. Selected datasets descriptions

Name	Sample Size	Removed
COCOMO81	63	–
CSC	145	4
Desharnais	81	1
Heiat-Heiat	35	–
MERMAID2	30	–
Miyazaki94	48	1
Moser-etal	37	–
cocomonasa_v1	60	–
Maxwell	62	–
NASA93	93	1

PROMISE repository site serves datasets as readable text file which can include comments. We selected datasets from this repository which contain a comment regarding information of projects or a citation they were used.

Dataset listed in [8] were from three journals: Transactions on Software Engineering, Information & Software Technology, and Journal of Systems & Software. We selected datasets freely available from original papers they referenced. We also limited datasets to which were collected in software development organizations.

Having removed duplicates between the two sources, we identified 28 datasets as an initial set.

4.2. Dataset Selection

We selected datasets with the following conditions:

- effort or size-related metrics must be contained.
- at least 30 projects must be contained.

Effort estimation models usually estimate target effort at least from size-related metrics such as KSLOC and Function Points (FP). We thus imposed the first condition.

Second condition was related to the limitation in subsection 2.2. This limitation imposed that the size of test subsets must be more than 2. Supposing that test subsets of a dataset have 3 or more projects, $1/10 \cdot N \geq 3$, where N is size of that dataset, must hold. Consequently a dataset must have 30 or more projects. Although performance measure based on 3 results was still less precise and less fine-grained, it was a reasonable criterion for dataset selection.

As a result, we specified 10 datasets. Table 1 shows their descriptions. "Removed" indicates results of outlier elimination. The number of removed was small for all datasets and thus it did not influence dataset selection results.

5. Results

Table 2 shows averages of performance evaluation results. At the next most right column, we counted performance measures showing higher performance. Performance measures showing the same values were ignored. We found consistent results on Moser-etal, cocomonasa_v1, and Heiat-Heiat. However, we found inconsistent results on the other datasets.

Table 3 shows p-values for performance measure comparisons. The most right column shows the number of performance measures with statistical significance at $\alpha = 0.05$. We having used k based on MMREs for EbA, statistical tests on MRE-based measures turned to be significant less times than those on the others.

The most right column of Table 2 shows the number of statistically higher performance measures. Considered statistical significance, we could found inconsistency results only on MERMAID2. EbA was consistently preferable on Maxwell and Heiat-Heiat. LR was consistently preferable on the other datasets.

MRE-based and MER-based measures were inconsistent on MERMAID2. Following definitions of MRE and MER, LR might tend to overestimate and EbA might tend to underestimate on MERMAID2. In contrast, we did not find statistical inconsistency among performance measures based on MRE. Thus, inconsistency could be explainable.

We think that our experiment procedure contributed to consistency. This study adopted more reliable evaluation procedure than that of [10] and performed under identical conditions in contrast to [7]. Carefully selected datasets suitable for experiment procedure was also effective. We thus concluded that a large part of inconsistency in the past study might be due to experiment procedure.

6. Conclusions

This study revealed that performance measures could be consistent with appropriate experiment procedure. Out experiment procedure reflect actual or desirable comparative study style and thus it was useful and practical to adopt our experiment procedure style for future comparative studies.

These results can be useful as a starting point for investigating and discussing sources of inconsistency. Investigating multiple feature models, some datasets having no useful feature will have to be dropped, though.

This study does not deny using datasets we dropped. However, using the selected datasets would be more suitable and reliable in comparative study, at least with our experiment procedure.

References

[1] G. Boetticher, T. Menzies, and T. Ostrand. PROMISE repository of empirical software engineering data. `http://promisedata.org/repository`, West Virginia University, Department of Computer Science, 2007.

Table 2. Performance Measure Results

Dataset	Models	MMAE	MMRE	MdMRE	PRED	MMER	RSD	LSD	#val.	#sig.
COCOMO81	LR	478	1.04	0.74	0.18	1.15	8.16	49.54	6	5
	EbA	577	0.98	0.77	0.12	2.09	8.77	53.47	1	0
CSC	LR	1117	0.63	0.40	0.29	0.54	4.90	16.24	4	2
	EbA	1215	0.62	0.41	0.31	0.66	4.95	15.16	3	0
Desharnais	LR	2017	0.58	0.35	0.38	0.45	8.72	43.48	5	5
	EbA	2205	0.56	0.35	0.34	0.51	8.98	46.41	1	0
Heiat-Heiat	LR	28	0.13	0.11	0.88	0.13	0.30	0.40	0	0
	EbA	27	0.12	0.10	0.90	0.12	0.25	0.32	7	4
MERMAID2	LR	5399	1.07	0.79	0.16	0.98	28.36	670.95	3	1
	EbA	5129	0.94	0.72	0.19	1.12	29.12	706.46	4	1
Miyazaki94	LR	21	0.40	0.32	0.39	0.38	0.41	0.49	4	1
	EbA	23	0.40	0.32	0.44	0.46	0.42	0.51	1	0
Moser-etal	LR	62	0.08	0.06	0.97	0.08	0.13	0.16	7	7
	EbA	132	0.20	0.14	0.76	0.19	0.38	0.45	0	0
cocomonasa_v1	LR	132	0.33	0.23	0.52	0.31	1.84	3.07	7	7
	EbA	166	0.41	0.32	0.39	0.44	2.39	4.73	0	0
Maxwell	LR	3800	0.55	0.48	0.21	0.58	8.56	46.15	2	0
	EbA	3900	0.55	0.44	0.29	0.60	8.38	44.65	4	2
NASA93	LR	346	0.64	0.41	0.35	0.79	6.83	35.90	5	4
	EbA	388	0.64	0.42	0.37	0.91	7.37	40.59	1	0

Table 3. Statistical Testing Results (p-value)

Dataset	MMAE	MMRE	MdMRE	PRED	MMER	RSD	LSD	stat.sig ($\alpha = 0.05$)
COCOMO81	0.00	0.13	0.53	0.00	0.00	0.00	0.00	5
CSC	0.00	0.40	0.08	0.17	0.00	0.82	0.47	2
Desharnais	0.00	0.11	1.00	0.01	0.00	0.03	0.01	5
Heiat-Heiat	0.31	0.02	0.03	0.24	0.08	0.01	0.01	4
MERMAID2	0.29	0.03	0.25	0.07	0.00	0.06	0.07	2
Miyazaki94	0.06	0.79	0.62	0.10	0.00	0.14	0.15	1
Moser-etal	0.00	0.00	0.00	0.00	0.00	0.00	0.00	7
cocomonasa_v1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	7
Maxwell	0.32	0.77	0.03	0.00	0.63	0.39	0.47	2
NASA93	0.01	0.83	0.36	0.18	0.00	0.00	0.00	4

[2] R. R. Bouckaert. Choosing between two learning algorithms based on calibrated tests. In *Proc. of 20th International Conference on Machine Learning*, pages 51–58, 2003.

[3] B. Efron. Estimating the error rate of a prediction rule: Improvement of cross-validation. *Journal of the American Statistical Association*, 78:316–330, 1983.

[4] P. I. Good and J. W. Hardin. *Common Errors in Statistics (and How to Avoid Them)*. Wiley Publications, 2006.

[5] M. Jørgensen. Practical guidelines for expert-judgment-based software effort estimation. *IEEE Softw.*, 22(3):57–63, 2005.

[6] M. Jørgensen and M. Shepperd. A systematic review of software development cost estimation studies. *IEEE Trans. Softw. Eng.*, 33(1):33–53, 2007.

[7] C. Mair and M. Shepperd. The consistency of empirical comparisons of regression and analogy-based software project cost prediction. In *Proc. of ISESE 2005*, 2005.

[8] C. Mair, M. Shepperd, and M. Jørgensen. An analysis of data sets used to train and validate cost. In *Proc. of PROMISE'05*, 2005.

[9] K. D. Maxwell. *Applied Statistics for Software Managers*. Prentice Hall, Inc., 2002.

[10] I. Myrtveit, E. Stensrud, and M. Shepperd. Reliability and validity in comparative studies of software prediction models. *IEEE Trans. Softw. Eng.*, 31(5):380–391, 2005.

Recommending Component by Citation: A Semi-supervised Approach for Determination

Sibo Cai, Yanzhen Zou†, Lijie Wang, Bing Xie, Weizhong Shao
Software Institute, School of EECS, Peking University, Beijing, P.R. China
Key Laboratory of High Confidence Software Technologies,
Ministry of Education, Beijing, P.R. China
{caisb06, zouyz, wanglj07, xiebing}@sei.pku.edu.cn, wzshao@pku.edu.cn

***Abstract*—Reusing existing components can help developers improve the development productivity as well as reduce the cost. Reuse repositories in this scenario act as a fundamental facility for acquiring needed components. While retrieving components in reuse repositories, developers often face the problem of choosing components from candidates that provide similar functionalities. To address the problem, this paper proposes a semi-supervised method to recommend developers components in reuse repositories. Different from existing rating based recommendation approaches that often suffer from the lack of user ratings, our approach calculates the recommendation probabilities of components based on their citations on the Internet. The citations are acquired through the websites (called *host* in this paper) that are associated with the components. Using a random walk algorithm, the associations between components and hosts are explored with recommendable components identified. We implemented our approach in a prototyping system based on which we conducted an experimental study to evaluate our approach. The experimental results demonstrate that our approach can accurately recommend components and thus has the potential to assist developers in reuse.**

Keywords-software reuse;component recommendation;reuse repository

I. INTRODUCTION

It is widely believed that reusing existing components can help developers create applications with less effect and improved quality [1]. To find proper components, developers need to retrieve reuse repositories according to their reuse plan [2]. Generally, the component retrieval process can be divided into two steps: Firstly, developers search reuse repositories to find component candidates that provide needed functionalities. Secondly, developers select components from candidates and integrate them together to build the target system.

Selecting components from candidates is very crucial for system integration. Before the adaption and integration of the components, developers often have to prototype a testing environment to validate the selected components to make sure that the selected components could satisfy the detailed requirements [3]. A casual selection of components may put developers into the risk of wasting a lot of time and effort. The situation becomes even worse if the components are not cost-free.

Currently, most widely used mechanism to provide hints for developers to make informed component selection decision in real world reuse repositories, such as SourceForge [5] and ComponentSource [6], is the user rating/review system. In such systems, developers can share their experience of certain component by simply giving a rating or a paragraph of remarks. Then all the ratings are aggregated for providing decision information to assist developers with components selection. User rating/review systems can provide information to assist developers in making selection decision; however, this kind of mechanisms is always blamed for their limited ratings due to the user motivation problem [7]. In most cases, developers have to spend extra effort collecting related information or try the retrieved candidates one by one to decide which component to use. The retrieval process becomes inefficient.

To resolve the problem, we propose a novel component recommendation method based on the citations of components appearing on the Internet. The citations of components on the Internet can be components for download (such as the components in Download.com [21]), components at runtime (like Java Applet [9]) or component-centric description or discussion (such as the appearance in Ohloh [22]). No matter what cases, components appearing on the Internet would relate to certain hosts. In general, the more times a component appears on the Internet, the more probable it should be recommended to developers [8]. Furthermore, if a host is known to involve amounts of recommended components, components cited by this host are more likely to be recommendable ones. Therefore, a component's citation on the Internet can be utilized for recommendation instead of user ratings.

In this paper, we propose a semi-supervised approach for component recommendation in reuse repositories based on their citations on the Internet and implement the approach in a prototyping system. In the approach, we obtain the hosts that involve the components from the Internet and build the associations between the components and the hosts. Through exploring the associations with a random walk algorithm, we work out the recommendation probability for each component. We evaluate our approach through an experimental study based on our implemented system and real world data. The results show that our approach can accurately identify recommendable components.

†Corresponding Author

The rest of this paper is organized as follows: Section 2 describes our approach for recommending components and an implemented prototyping system. Section 3 presents our experimental study on the proposed approach with the results analyzed. Section 4 presents some discussions about the approach and the future work. Section 5 describes the related works. Then in the last section, we conclude this paper.

II. OUR APPROACH

In this section, we will introduce our approach for recommending components as well as an implementation of the approach based on which we conducted the experimental study. The approach consists of following steps.

- Obtaining hosts associated with components. This step aims to find hosts related to the components from the Internet and build associations between them.
- Setting up the association weight. The relevance degree of components with their hosts is considered in our approach. For each association, a weight denoting the relevance degree is assigned for further computation.
- Computing component recommendation probability. This step computes the recommendation probability for each component through exploring the associations using a random walk algorithm and finally identifies the recommendable components.

Details of these steps are described below.

A. Obtaining Hosts associated with Components

To obtain the hosts related to the components from the Internet, we use the web search engine Google[1] to accomplish the hosts crawling task. We use Google to search the components with component name as the query and extract URLs from the returned result list. Since the most relevant results are usually distributed at the top position, we only keep the top 30 URLs for each component (returned list less than 30 URLs will all be kept). Then the hosts related to the component are identified (we use the domain part of a URL to represent a host. URLs with the same domain part will be merged).

However, the returned results by Google do not always refer to the exact component if we directly use the component name as the query. In order to obtain the hosts and build the associations more accurately, some strategies are adopted to refine the associations between the components and the hosts. Firstly, Google in general will split the input keywords into word segments, search each segment separately and at last merge the results of each word segment to produce the final results. Such manner, however, is intended to fetch web pages that contains word segments of the query but in fact are irrelevant to the components in our scenario. To solve the problem, we validate the returned results by using full match of the component name in both the title and the snippet. This can be fulfilled by adding quotation mark to the component name when searching components. Secondly, the Google web search engine aims to obtain as diverse results as possible, but in our scenario, the returned results by Google should be restricted in the domain of "Software" or "Software Development". For this problem, we append the keyword "Software" to the query to refine the returned results.

Formally, the associations between components and their hosts can be described as a bipartite graph, $G=<COMP, HOST, EDGE>$. The component set *COMP* and the host set *HOST* constitute the partitions of the graph. Every association is described as an edge $(comp, host) \in EDGE$ where $comp \in COMP$ and $host \in HOST$. Fig. 2(a) gives an example of a bipartite graph.

B. Setting up the Association Weight

In our approach, we consider the degree of components associated with their hosts and a weight is assigned to each association. The association weight between a component and a host denotes the relevance degree of the component to the host. In general, the more times a component appears on one host, the more degree it is relevant to the host.

The association weight of a component related to a host is firstly set as the number of URLs related to the component that are merged into the host. Then, the weights of all associations are normalized using (1) where $N_{comp\text{-}host}$ is the number of URLs related to the component *comp* and merged into *host*. Since the association is non-directional, the weight of a component related to a host equals to the one in reverse order.

$$w_{comp-host} = w_{host-comp} = \frac{N_{comp-host}}{\sum_{c,h:(c,h)\in EDGE} N_{c-h}} \quad (1)$$

C. Computing Component Recommendation Probability

The idea behind our component recommendation approach is that the more hosts by which a component is cited, the more probable a component is to be recommended. More importantly, if a host is known to involve amounts of recommended components, components appearing on the host are more likely to be recommendable ones. To implement the idea, we assign each component a recommendation probability (called "component recommendation probability" in this paper), which means the degree of the component to be recommended. We also set a weight (called "host recommendation probability" in this paper) to every host. The host recommendation probability indicates the degree of a host involving recommended components.

Let *P(host)* denotes the host recommendation probability of *host* and *P(comp)* denotes the component recommendation probability of *comp*. Clearly, a host that involves more components with high component recommendation probability will gain higher host recommendation probability. We use (2) to calculate the host recommendation probability of *host* where *C(host)* denotes the components cited by *host*.

$$P(host) = \sum_{comp:comp\in C(host)} w_{host-comp} P(comp) \quad (2)$$

[1]http://www.google.com/

On the other hand, a component that is involved by more hosts with high host recommendation probability can also gain higher component recommendation probability. Hence, we use (3) to update the component recommendation probability of a component where *H(comp)* denotes the hosts that involve *comp*.

$$P(comp) = \sum_{host:host \in H(comp)} w_{comp-host} P(host) \qquad (3)$$

The starting point of the computation is a set of components that are prejudged to be recommended ones and assigned high recommendation probability. We call this group of components "seeds". To implement the computation, we propose a propagation algorithm by adopting a random walk algorithm with absorbing states [14]. The algorithm is presented in Fig. 1.

Input: the seed set S, COMP-HOST bipartite graph G, the vanishing threshold β, the transition probability α to ω
Output: *P(comp)*, for every component except for the seeds

1: **for each** *comp* **in** S **do** *P(comp)*=1
2: **repeat**
3: **for each** *host* **in** HOST **do**
4: $P(host) = (1-\alpha)\sum_{comp:comp \in C(host)} w_{host-comp} P(comp)$
5: **if**($P(host) < \beta$) **then** *P(host)*=0
6: **end for**
7: **for each** *comp* **in** COMP \ S **do**
8: $P(comp) = (1-\alpha)\sum_{host:host \in H(comp)} w_{comp-host} P(host)$
9: **if**($P(comp) < \beta$) **then** *P(comp)*=0
10: **end for**
11: **until** convergence

Figure 1. The propagation algorithm

The input of the algorithm contains the seed set S, COMP-HOST bipartite graph G and two parameters α and β. The transition probability α denotes that both the components and the hosts have the probability to transfer to the absorbing state ω [14]. This means that if a component or a host does not keep acquiring the recommendation probability, its probability will gradually decrease to 0 and thus be absorbed by ω. The vanishing threshold β is used as the threshold to distinguish the recommendable components. The output of the algorithm is the component recommendation probability *P(comp)* for each component except the seeds.

Initially, the component recommendation probabilities of the seeds are set to 1 (Line 1). Then the algorithm (Line 3 to Line 10) iteratively calculates the probability for both the hosts and the components. In each iteration the host recommendation probability for each host is calculated using (2) and weakened by $(1-\alpha)$ (Line 4). If the calculated probability is less than β, the probability is set to 0. The calculation of the component recommendation probability is similar to the one of host recommendation probability. Note that besides the components, we also use β as a threshold to discard the probabilities of hosts in Line 5. This is mainly for efficiency purpose [14]. The iteration will continue until the recommendation probabilities of both the components and the hosts are convergent. Components with the recommendation probability greater than 0 then will be regarded as recommended ones.

To explain the propagation algorithm in a more concrete way, an illustrated example is given in Fig. 2. In the example, there are 5 components (indicated as squares) and 4 hosts (indicated as circles) that have associations with the components as shown in (a). Components *C2* and *C3* are seeds. Suppose that all the associations have the same weight. Initially, the component recommendation probabilities of *C2* and *C3* are set to 1. In the first iteration, hosts that have associations with *C2* and *C3* will gain host recommendation probability. Thus, the host recommendation probabilities of *H1*, *H3* and *H4* (indicated as solid circles) are updated as shown in (b). Note that the host recommendation probability of *H3* will be larger than those of *H1* and *H4* since *H3* have associations with both *C2* and *C3*. The hosts will also backwards affect the recommendation probabilities of the components except the seeds. Therefore, the component recommendation probabilities of *C4* and *C5* (indicated as solid squares) will be updated as shown in (c). Similarly, the host recommendation probabilities of all the 4 hosts (shown in (d)) and backwards the component recommendation probabilities of *C1*, *C4* and *C5* will be updated (shown in (e)) in the following iteration. The iteration continues until the recommendation probabilities of both the components and the hosts converge.

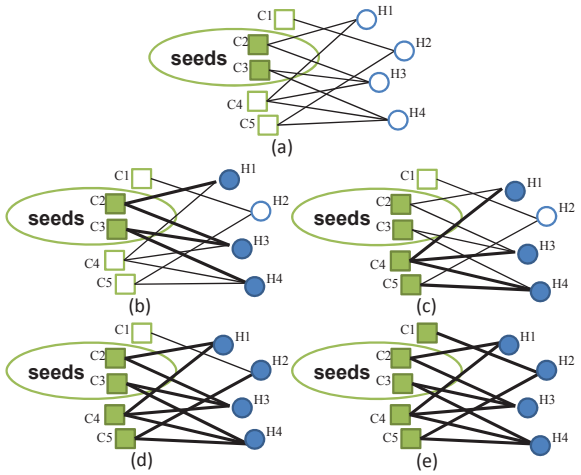


Figure 2. An illustrated example of the propagation algorithm

D. Implementation with a Prototyping Reuse Repository

We implement our approach with a prototyping reuse repository that simply provides free-text based component retrieval [10]. Note that although we use free-text approach to acquire relevant components in the prototyping system, our method for recommending components is not limited by the component retrieval mechanisms. The architecture of the reuse repository with our recommendation approach is presented in Fig. 3. Components are stored in the ***reuse repository***. The ***retriever*** accepts queries of developers and retrieves relevant

component candidates from the reuse repository for further recommendation.

The implementation of our approach is denoted in the dash line rectangle in Fig. 3. The ***crawler*** obtains the hosts that have association with the components in the reuse repository from the Internet and stores the associations in the ***association database***. Based on the work done by the crawler, the ***analyzer*** computes the recommendation probability for each component utilizing the associations and produces the recommendable components based on the retrieved candidates to developers.

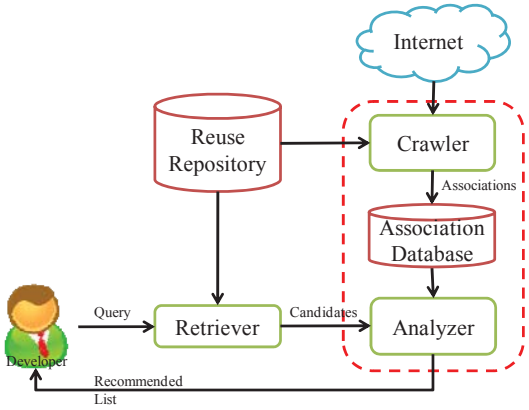


Figure 3. The architecture of the reuse repository with our remmendaiton approach

III. EXPERIMETNAL STUDY

A. Experimental Organization

To evaluate our approach, we applied it to the data collected from a real world reuse repository, i.e. SourceForge, which not only provides software systems, but also many reusable libraries that fuel the further development of new applications. Particularly, we selected the category "Software Development" as our evaluation base. Category "Software Development" totally contains 35,602 software projects at the time we carried out the evaluation. We acquired the project information including project name, description, user positive rating and negative rating etc. by implementing a web page crawler. The 35,602 software projects constituted the reuse repository in our approach. In our experiment, each software project was viewed as a component.

Based on the built reuse repository, we used Google to search for the hosts that involve the components in the reuse repository and extracted associations between the components and the hosts. In this study, we removed the SourceForge from the host set to eliminate the impact of SourceForge since SourceForge acts as the evaluation base of our experimental study. Associations including SourceForge were also excluded. Thus, we totally fetched 251,873 hosts as well as 937,016 associations between the component set and host set.

We sorted the 35,602 projects according to the user ratings provided in our crawled pages from SourceForge. After a review of the sorted list, 100 projects were selected as seeds (less than 0.3% of the component set). The 100 projects were either famous software projects, or with high user positive rating that are supposed to recommend to developers.

In the execution of the propagation algorithm, we also need to set the parameters, i.e. α and β. In our experiment, α was set to an empirically small value 0.01 [14] while the selection of β somewhat depends on the application scenarios. To solve the selection of β, we used the seed set to tune it. We divided the seed set into 2 parts. Four fifth of the seed set were still used as seeds in the propagation algorithm while the remaining one fifth were regarded as the test set to tune β. The tuning algorithm is shown in Fig. 4. After conducting the tuning algorithm, we finally set β to 5E-5.

Step 1: Randomly select 80 seeds from the seed set.
Step 2: Conduct the propagation algorithm with the parameter α set to 0.01 and β set to 0.
Step 3: Store the minimum value of the calculated probabilities of the remaining 20 seeds.
Step 4: Repeat Step1, Step 2 and Step 3 in 5 runs and set β as the average of the stored minimum values.

Figure 4. The tuning algorithm for β

B. Experimental Results

We evaluated the effectiveness of our approach by using developer queries. In order to decide which queries to use, we interviewed 7 graduate students in Peking University who have more than 2 years of software development experience. Finally, 11 queries were identified. We then submitted the 11 queries to our prototyping system to retrieve relevant components and more importantly the recommended ones by our approach. To validate the recommended components, we adopted the user ratings provided by SourceForge. Components were regarded as recommended ones if their number of positive ratings in SourceForge is larger than the one of negative ratings.

TABLE I. RECOMMENDED COMPONENTS BY SOURCEFORGE COMPARED TO THE ONES BY OUR APPROACH

Query	Recommended	Retrieved
XML Parser	3(5)	14
Data Encryption	1(1)	7
Logging	9(11)	12
Math	5(5)	12
Statistics	5(5)	7
Data Compression	1(1)	10
Email	1(1)	6
File Upload	2(4)	14
Configuration File	3(3)	11
Network Utility	1(1)	11
IO Utility	2(2)	14

To evaluate our approach, we identified the recommended components by SourceForge from the retrieved ones for each query and then found out whether our approach could produce similar recommended list. The results are presented in TABLE I. The queries are listed in the "Query" column. Column "Retrieved" indicates the number of retrieved components according to the query while the "Recommended" column

represents the number of recommended components by our approach contained in the recommended list of SourceForge. The number of components recommended by SourceForge is also indicated (in the bracket). Take query "XML Parser" as an example, the number of retrieved components related to the query is 14 and in the 5 recommended components of SourceForge, our approach suggests 3 of the 5.

Through the comparison with recommended components by SourceForge that are based on user ratings, we preliminarily drew to the conclusion that our approach possesses the ability to identify the recommended components suggested by SourceForge. More queries should be performed to further validate our approach; however, the selected queries to some extent cover amply portion of fundamental software development needs based on our interview with software developers. In order to show the effectiveness of our approach in a better fashion, seeds (if retrieved) were excluded in the retrieval results.

We also evaluated the precision of the recommended components by our approach. Precision here means the ratio of actual recommended components compared to the ones suggested by our approach. We compared the number of recommended components by our approach to the ones suggested by SourceForge. The 11 queries were still used. TABLE II shows the comparison results. In the column "Recommended", the number of recommended components of our approach is indicated in the bracket. In the recommended list of our approach, the number of components that are also suggested by SourceForge is indicated outside the bracket. In half cases, our approach performs well and produces recommended components similar to the ones by SourceForge, such as "Logging", "Statistics". While in the other half, our approach recommended more components than SourceForge does, especially in the case "Network Utility".

TABLE II. OUR RECOMMENDED COMPONENTS COMPARED TO THE ONES BY SOURCEFORGE

Query	Recommended	Retrieved
XML Parser	3(7)	14
Data Encryption	1(3)	7
Logging	9(9)	12
Math	5(8)	12
Statistics	5(6)	7
Data Compression	1(3)	10
Email	1(5)	6
File Upload	2(7)	14
Configuration File	3(7)	11
Network Utility	1(8)	11
IO Utility	2(6)	14

To explore the reason of this phenomenon, we investigated the recommended components by our approach while not suggested by SourceForge. We found that almost all these components received 0 positive rating and 0 negative rating in SourceForge. According to the criteria of recommended components, such components will be judged as not recommended by SourceForge. "GSA Simple XML Parser[2]" is one of the examples. However, through our investigation of "GSA Simple XML Parser" by hand, we finally judged that this component should also be recommended.

[2] http://sourceforge.net/projects/gsa-simple-xml/

IV. DISCUSSION AND FUTURE WORK

A. Issues about the Association Refinement

In our approach, we refine the association between components and hosts using the strategies described in section 2, but there are still some problems in building the associations. The most extrusive case is that the name of a component is too general. For example, an xml parser named "xml parser". To search such keywords in Google, the returned results are often irrelevant to the component. Such cases will reduce the effectiveness of our approach and provide problematic recommended list to the developers. Nevertheless, we find that such examples only occupy a very small fraction of the components in real world reuse repositories since people who develop components are mostly intended to pick up a more meaningful name for their components while further consideration should be taken to deal with such cases.

B. How to Obtain the Seed Set

In our approach, one of the inputs to calculate the recommendation probabilities of the components is the seed set. The effectiveness of our approach will be greatly reduced if the seed set is hard to obtain. However, the seed set seems not so difficult to identify in real world reuse repositories. Firstly, just like our experimental study, components that have already received high user positive ratings can be considered. Secondly, famous software components are another option. Thirdly, components that developed by famous companies or organizations can also be taken into consideration.

Another issue about the seed set is how to construct a better seed set. The selection of the seeds may influence the performance of our approach since it is the starting point. Selecting seeds as divergent as possible may be one of the possible strategies that can be used to enhance the performance of our approach. For instance, seeds can be selected considering different application domains. Further study will be carried out in our future work.

V. RELATED WORKS

Helping developers obtain appropriate components is crucial for successful software reuse. In the literature, many research works have been studied to facilitate the component retrieval, such as 1) free text approaches [10]; 2) facet based approaches [11]; 3) signature based approaches [12] and 4) behavior based approaches [13]. Most of these research works concentrate on obtaining relevant components while pay little attention to providing information for developers choosing components from candidates that provide similar functionalities. As the size of reuse repositories becomes large, helping developers make informed decision among functionally similar components can accelerate the process of selection and this is what our approach is trying to accomplish.

To recommend software components among candidates, Inoue et al. proposed the component rank model based on which they developed SPARS-J, a java class retrieval system

[4]. The component rank model analyzes the usage relation of components so as to identify recommendable components that are used more frequently. The work is based on the premise that the component source code is acquirable but in many cases (e.g. COTS) this is unrealistic. Ichii et al. proposed a software component recommendation approach based on collaborative filtering (CF) utilizing user browsing history [15]. However, CF inborn suffers from the "Cold Start" problem. In real world reuse repositories, such as SourceForge and ComponentSource, user rating/review systems are usually used to recommend software components while these systems are often blamed for their shortage of ratings/reviews [7]. In our approach, we make use of the associations between components and hosts obtained from the Internet to avoid the above problems as well as provide hints to help developers make selection decision.

There are also several pieces of work concerning recommendation systems in the research area of software engineering. CodeBroker [16] provides recommendation of APIs for developers that may implement the needed functionalities the moment developers write down comments or the signature of methods. Strathcona [17] developed by Holmes et al. recommends example code to developers by monitoring the code under development. The recommendation is made according to six structure-based heuristics. Li et al. proposed an approach to recommend typical usage example of APIs by adopting code clustering [18]. Kim et al. presented an approach that searches code bases on the Internet and extract code examples to insert into API documents to recommend developers suitable examples while using certain APIs [19]. ParseWeb [20] is another useful tool to suggest call sequences for developers when developers are using certain APIs. Different from these approaches that recommend implemented APIs or example code, our approach aims to help developers make informed choice facing components that provide similar functionalities.

VI. CONCLUSION

In this paper, we proposed a semi-supervised approach to produce recommended components to the developers to assist their selection of components in reuse repositories. The approach utilizes the associations between the components and the involved hosts. Through a selected group of components that are supposed to be recommended and a propagation algorithm, the recommendation probability for each component is calculated. We also implemented a prototyping system and conducted an experimental study on our approach using real world data. The results show that our approach can accurately recommend components to the developers comparing to the data from SourceForge.

ACKNOWLEDGMENT

We would like to thank Jing Jin for the experimental data collection. This research was sponsored by the National Natural Science Foundation of China under Grant No. 60821003, the National Basic Research Program of China (973) under Grant No. 2009CB320703 and the National High-Tech Research and Development Plan of China under Grant No.2007AA010301-01.

REFERENCES

[1] V. Basili, L. Briand, and W. Melo, "How reuse influences productivity in object-oriented systems," Communications of the ACM 39(10), pp.104-116, 1996.

[2] H. Mili, A. Mili, S. Yacoub, and E. Addy, Reuse based software engineering: techniques, organizations, and measurement. Wiley-Interscience Press, Chichester, 2001.

[3] R. Land, A. Alvaro, and I. Crnkovic, "Towards efficient software component evaluation: an examination of component selection and certification," In: 34th Euromicro Conference Software Engineering and Advanced Applications, pp. 274-281, 2008.

[4] K. Inoue, R. Yokomori, T. Yamamoto, M. Matsushita, and S. Kusumoto, "Ranking significance of software components based on use relations," IEEE Transactions on Software Engineering, 31(3), pp. 213-225, 2005.

[5] SourceForge, 2010, http://sourceforge.net/

[6] ComponentSource, 2010, http://www.componentsource.com/

[7] A. Jøsang, R. Ismail, and C. Boyd, "A survey of trust and reputation systems for online service provision," Decision Support Systems 43(2), pp. 618-644, 2007.

[8] D. Weiss, "Measuring Success of Open Source Projects Using Web Search engines," In: Proceedings of the First International Conference on Open Source Systems, pp. 93-99, 2005.

[9] R.C. Seacord, S.A.Hissam, and K.C. Wallnau, "AGORA: a search engine for software components," IEEE Internet Computing 2(6), pp. 62-70, 1998.

[10] Y.S. Maarek, D.M. Berry, and G.E. Kaiser, "An information retrieval approach for automatically constructing software libraries," IEEE Transactions on Software Engineering, 17(8), pp. 800-813, 1991.

[11] R. Prieto-Diaz, and P. Freeman, "Classifying software for reuse," IEEE Software 4(1), pp. 6-16, 1987.

[12] A. Zaremski, and J.M. Wing, "Specification matching of software components," ACM Transactions on Software Engineering and Methodology 6(4), pp. 333-369, 1997.

[13] A. Podgurski, and L. Pierce, "Retrieving reusable software by sampling behavior," ACM Transactions on Software Engineering and Methodology 2(3), pp. 286-303, 1993.

[14] A. Fuxman, P. Tsaparas, K. Achan, and R. Agrawal, "Using the wisdom of the crowds for keyword generation," In: Proceedings of the 17th international conference on World Wide Web, pp. 61-70, 2008.

[15] M. Ichii, Y. Hayase, R. Yokomori, T. Yamamoto, and K. Inoue, "Software component recommendation using collaborative filtering," In: Proceedings of the 2009 ICSE Workshop on Search-Driven Development-Users, Infrastructure, Tools and Evaluation, pp. 17-20, 2009.

[16] Y. Ye, and G. Fischer, "Supporting reuse by delivering task-relevant and personalized information," In: Proceedings of the 24th International Conference on Software Engineering, pp.513-523, 2002.

[17] R. Holmes, R.J. Walker, and G.C. Murphy, "Approximate structural context matching: an approach for recommending relevant examples," IEEE Trans. Softw. Eng., 32(1), pp. 952–970, 2006.

[18] Y. Li, L. Zhang, G. Li, B. Xie and J. Sun, "Recommending typical usage examples for component retrieval in reuse repositories," In: Proceedings of the 10th international conference on Software Reuse: High Confidence Software Reuse in Large Systems, pp. 76-87, 2008.

[19] J. Kim, S. Lee, and S. Hwang, "Towards an intelligent code search engine," In: Proceedings of the AAAI Conference on Artificial Intelligence, 2010.

[20] S. Thummalapenta and T. Xie, "PARSEWeb: A programming assistant for reusing open source code on the web," In: Proceedings of the IEEE/ACM International Conference on Automated Software Engineering, pp. 204–213, 2007.

[21] Download.com, 2010, http://download.cnet.com/

[22] Ohloh, 2010, http://www.ohloh.net/

Testing Configurable Component-Based Software - Configuration Test Modeling and Complexity Analysis

Jerry Gao[①②], Jing Guan, Alex Ma, Chuanqi Tao
①San Jose State University, USA

Xiaoying Bai
②Tsinghua University, China

David C. Kung
University of Texas at Arlington, USA

Abstract: As the advance of software component technology, engineers encountered different issues and challenges in testing and automation of configurable components and component-based programs. One of them is how to validate configurable components and programs to achieve adequate test criteria and support test automation. This paper uses a test model, known as *a semantic tree*, to assist engineers to model and analyze diverse composite components and configurable software in terms of configurable environments, organization structures and functions. Based on this model, well-defined test criteria are presented to address the adequate testing issues. In addition, the paper discusses two test complexity evaluation methods for configurable components and software. Furthermore, some case study results are reported to demonstrate the testing complexity of diverse configurations.

KEYWORDS: test modeling and analysis, configurable software testing, test complexity, test criteria, and configuration testing.

1. Introduction

In the last two decades, software component reuse has been a popular concept and approach in software construction. To assure software product quality, engineers need to assure the quality of reused components and programs before deployment. Hence, testing component-based systems has been a very hot research subject in the past decade.

As the advance of component technology, the complexity of components increased from functional component box to frameworks/middleware and configurable/customizable components. Software users (or clients) are allowed to configure, select, and customize components and software based on their functional requirements, desirable environments, and selected organization structures. In the past two decades, many published papers focus on software design, construction and management issues concerning configurable components and programs in production lines [2][4][5][6][7][8]. Although there are numerous published papers discussing how to construct configurable software and components in different aspects, only a few of them focus on testing and automation of configurable components and software [21-26].

In 2003, the authors have pointed that there is an emergent need in testing configurable and customizable components [1][27]. In the past decade, there are numerous published papers addressing issues on testing components and component-based systems. However, only a few of them addressed testing issues and challenges for configurable components, frameworks, and configurable component-based software [21-26]. From 2008 to 2009, we collaborated with a software QA group in SUN Microsystems to set up several test projects in the software testing class at San Jose State University to validate the selected open-source components and middleware developed by ***SUN Microsystems***. Typical examples are ***SocialSite, SailFin, and Glassfish***, After went through the projects, students not only found numerous bugs, but also discovered some other problems in testing component-based software with configurable features and structures. These issues and challenges are summarized below.

- How to identify and present diverse configurations in terms of component/software composition structures, functions, and environments?
- How to define adequate test criteria and evaluate the test coverage for configurable components and software?
- How to analyze and measure the test complexity of configurable components and software?

Today engineers lack well-defined test process, adequate test models and criteria, as well as test automation solutions for configurable components and systems. According to QA and test engineers in the real world, there are the following needs.

- Effective test models to assist test modeling and analysis of components and systems with configurable structures and features.
- Well-defined test criteria and coverage analysis for components with diverse configurations in environments, functions, and structures.
- Cost-effective solutions to evaluate configuration-oriented test complexity and cost.
- Test automation solution to support auto-testing of configurable components and software.

This paper focuses on the first three needs. The paper uses a model-based approach to address testing issues. A new model, known as a semantic tree, is used to assist engineers to perform test modeling and analysis for configurable component-based systems. This model can be used to present diverse component configurations statically or dynamically, including their configurable environments, organizational structures, and functions. Based on the given model, a set of configuration-based test criteria is defined, and a test complexity evaluation method is provided.

The paper has three primary contributions in software component testing. Firstly, it uses a model-based approach to modeling, presenting and analysis of diverse configurations in components. Unlike the existing work, the proposed model allows engineers to present and analyze both static and dynamic configurable structures and features of component-based systems. Secondly, it presents well-defined test criteria based on the proposed model to address configuration-oriented test adequacy. Thirdly, it provides a systematic method to evaluate test complexity of diverse configurations in components and systems.

The paper is structured as follows. The next section discusses the background and related work. Section 3 discusses the essential issues in testing configurable components. Section 4 presents a test model, known as a semantic tree, and related properties and

spanning trees to model the configurable structures for software components. Section 5 discusses a set of test criteria for configuration features in components and software. Finally, Section 6 reports some case study results, and the conclusion remarks and future work are given in Section 7.

2. Related Work

There are numerous published books and papers addressing different topics and issues on component testing in the past decade.

- *Component testing techniques* - For example, the authors in [13] presents flow-graph based test model and a technique to support API-based component black-box testing by focusing on a component's accessible functional sequences. In addition, the paper in [17] discusses testing of state-based dynamic behaviors for a component.
- *Design for component testability* – Roy Freedman [10] discusses how to measure component function testability. Gao et al. in [11] proposed a component testability model to evaluate software components from five different factors. In addition, there are a number of papers discussing how to design for component testability using different approaches.
  - Built-in-test components [9][20], where component tests are coded inside components.
  - Framework-based testable components [14][15], where test-driven handlers are built inside components based on well-specified component APIs.
  - Systematic wrapping for testable components [19], where test-oriented component wrappers are created to support tests based on well-defined API artifacts.
- *Component test adequacy and coverage* – As pointed out in [1][27], most existing test criteria and test methods can be used in component testing to achieve function-oriented test criteria and state-based test adequacy. However, there are some open questions. One of them has been addressed by [12], which is API-based access sequences and related test criteria. The other open issues are relating to component reuse contexts, component interactions, and configurations.

Based on our recent literature survey, there are only a few of papers addressing testing issues and challenges in configurable components. The existing work can be classified into three groups:

- *Testing configurable system constraints using combinatorial interaction testing* (CIT) [24, 25, 26] – CIT is a method to sample configurations of a software system systematically for testing. Many algorithms have been developed to create CIT samples. A general constraint representation and the related solving technique are presented in [26]. It focuses on this problem by examining two highly configurable software systems to quantify the nature of constraints in real systems. CIT can provide an effective way to sample configurations for testing. Cohen et al. in [25] focused on CIT-based test coverage and fault coverage for crossing system configurable parameters and their constraints. Based on CIT, Qu et al. in [21] introduced an approach to regression testing of configurable software using prioritization, which aims at earlier detection of defects.
- *Regression testing of system with configurable features* – For instance, Robinson et al. proposed a firewall method for regression testing of user-configurable software [22]. This paper focused on user-centered tests for system configuration. They constructed a firewall to identify the impacted area in system based on setting changes and configurable element changes respectively, then created or selected test cases to cover the impacts. Some case studies are reported.
- Component compatibility testing – Yoon et al. in [23] presented some configuration space models, known as CDG (component dependency graph) and ACDM (annotated component dependency model), to present the relationships among components, their versions, inter-component dependency, and constraints. Moreover, they also provided three test strategies based on the proposed models and experimental results.

Unlike the existing research, this paper provides a configuration model to present three-dimension configuration space for components and systems, which includes configurable environments, compositional architectures, and selective functions. The model is known as a semantic tree model, which can be used to present both static and dynamic configurations for component-based software and its parts. Instead of focusing on configurable system combinational constraints based on parameters, we focus on diverse configurations in environments, architectural structures, and selective functional features. We use a model-based approach to address the testing issues in three-dimension configuration space for component-based systems, including test modeling, test adequacy, and test complexity analysis. Some case studies are reported.

3. Testing Configurable Software

What is a *configurable component*? It refers to a compostable component, which not only provides a specified contracted interface, domain-specific service functions, and deployment solution with necessary artifacts, but also is equipped with the configuration capability that allows its users to make configuration selections in its environments, functions, and structures statically or dynamically. Similarly, *configurable software* is a program supported with a capability that allows users to make various configuration decisions statically or dynamically to generate different deployment instances based on their desires. Component-based software usually supports configurations in two different ways: a) static configuration, in which decisions and selections are made statically, and b) dynamic configuration, in which decisions and selections are made during runtime. Based on our recent experience, many modern frameworks and component-based software allow users to make following types of configuration decisions.

- Environment configuration capability – which allows users to configure and select different deployment environments.
- Organization configuration capability – which allows users to select different organization and composition structures based on available components and building parts.
- Function configuration capability – which allows user to select different functional features based on their needs.

Figure 1 presents the configuration test space as a 3-dimension space for configurable *software* and *components*, in which the X-axis presents all possible environment configurations, the Y-axis presents all of configurable structures/organizations, and the Z-axis presents all of configurable functions.

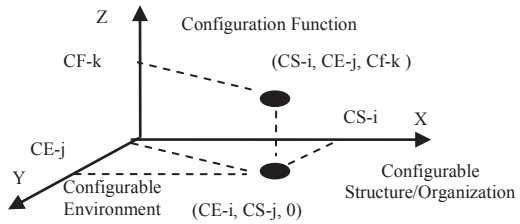


Figure 1 Configuration Test Space for Configurable Software

3.1. A Test Process

Figure 2 shows a test process for a configurable software and its components, which consists of the following steps.

1. ***Configuration environment check***, in which different environments are set-up to check product installation and deployment. The focus of this step is to make sure that the under-test component (framework or software) can be operated properly under each specified configuration environment.
2. ***Configurable structure/organization validation***, in which diverse configurable architectures (or organizations) are the focuses. Its primary purpose is to assure that configurable architectures can be structured successfully to support functions.
3. ***Configuration function check,*** in which configuration functions must be validated to achieve the selected test criteria. The major focus here is to assure the quality of the provided configuration service functions.
4. ***Configurable instance validation***, in which each configured component/software is deployed and validated under a specific environment to assure its quality in a reuse context.

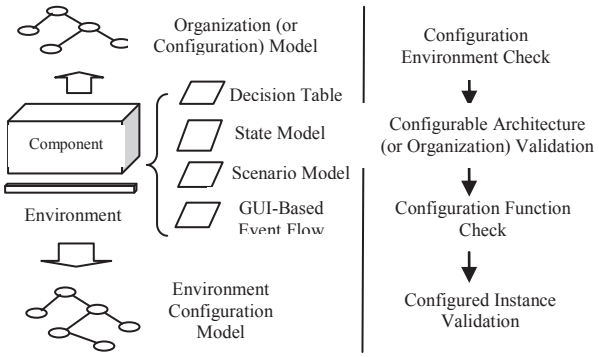


Figure 2 A Test Process for Configurable Software

3.2. Testing Myths

Since 2009, we used two software testing classes (collaborated with Sun Microsystems) to test several open-source configurable frameworks and platform servers (i.e. glassfish-v3) in San Jose State University. Both of them support rich configuration functions and environments. Here, we summarize our findings regarding to problems and myths in testing configurable software (or component) ***P*** below. Assume P' is a configured instance of P.

Myth #1*:*
If P_1 has been tested in a deployment environment P_E, then, there is no need to validate another configured instance P_2 of P on the same deployment environment P_E.
Realty:
We found that P_2 may have problems to perform its quality function services on P_E, even though P_1 functions well on the same deployment environment P_E.

Myth #2:
When Component A is a configurable part of a software P, if it has been tested as a part of one instance of P, then there is no need to test it again in another instance of P when it is included as its part.
Realty:
We found that as a configurable part, Component A may work well in one deployed instance of P, but it may not work properly inside another deployed instance of P, because both instances may include component A with different composition structures (or relations). This leads to two different reused contexts for Component A. Therefore, it should be tested in both reused contexts.

Myth #3*:*
When a function (F) is a configurable part of a configurable component P, if it has be tested in one of its configured instance P', then there is no need to validate the function F again in another configured instance P".
Realty:
We found that a configurable function F in a configurable component P may work well in one configured instance, but may not work properly in another instance of component P although both instances provide function F as one of their service functions. This leads to two different reused contexts for Component A. Therefore, it should be tested in both instances since they are different reused contexts.

4. Test Modeling for Configurable Software

Although there are numerous useful models in software testing, very few are suitable to model and present diverse configurations and their mappings in the 3-dimension configuration space for configurable component-based software. As shown in Figure 1, we need a well-defined test model to assist engineers to analyze and present each configurable structure (or compositional architecture) (say CS-i) under a specified configurable environment (say CE-j) as well as the corresponding configuration function (say CF-k), so that model-based test criteria, test generation methods, test complexity analysis techniques can be developed.

This section first discusses an updated test model to serve this purpose. It is known as a semantic tree model, which has been proposed as a test model to address problems and needs in software installation testing [16]. Here, the semantics tree model is used as a basis to model to address the needs in configuration testing of the configurable component-based software in the 3-dimension space.

4.1. Test Model – A Semantic Tree Model

Intuitively, a semantic tree is a special tree model for configurable component-based software. Each tree can be used to model and present the configuration elements and their relations in one of three dimensions in the configuration space. The tree nodes present configurable parts (or elements), for example, configurable

components. The links present different semantic relations between nodes. A semantic tree model can be formally defined as 3-tuple = (N, E, R), where

- N is a set of tree nodes. There are three types of nodes: a) a single root node, b) intermediate nodes (or parent nodes), and c) leaf nodes.
- E is a set of links between nodes. Each link connects a parent node and one of its child nodes in a tree. Each link show a part of a semantic relation between a parent node and its child nodes.
- R is a set of relations, and each item in R has a semantic label that presents a semantic relation between a parent node and its child nodes. There are four types of semantic relations with labels: EOR, AND, SELECT-1, and SELECT-M. Their detailed semantics are given in Table 1. Figure 3 shows their notations, where P-Node is a parent node, and C-nodes are its child nodes.

To support the model-based analysis, we introduce a concept of semantic spanning trees based on the semantic tree model.

Semantic Spanning Tree:

A semantic spanning tree G_{SPT} is a sub-tree of a given semantic tree G_{ST}. Unlike common spanning trees, a semantic spanning tree G_{SPT} for G_{ST} only can be derived based on the following properties:

- For each parent node (Np) with an AND relation in G_{SPT}, it must include all of its child nodes and its links.
- For Np with an EOR relation in G_{SPT}, it must include only one of its child nodes and the corresponding link.
- For Np with a SELECT-1 relation in G_{SPT}, it must include only one of its child nodes and the corresponding link.
- For Np with a SELECT-M relation in G_{SPT}, it must include only M child nodes and the related links.

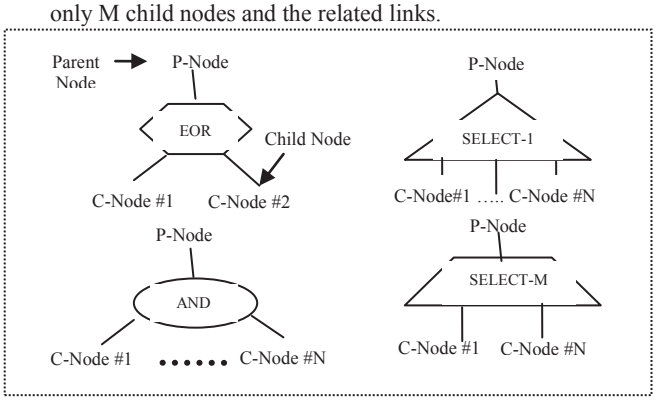


Figure 3 The Notation of Semantic Relations in a Semantic Tree

Table 1 Different Semantic Relations in a Semantic Tree Model

Relations	Relation semantic descriptions (where P is a parent node, and Ci is its child node)
EOR(P, <C1, C2>)	EOR relation indicates that P-Node has two child nodes C1 and C2. They only can be configured exclusively.
AND(P, <C1,…,Cn>)	AND relation indicates that P-Node must be configured with all of its child nodes C1,…,, Cn.
SELECT-1(P, <C1,…,Cn>)	SELECT-1 relation indicates that P-Node must be configured with one of its selective child nodes C1,…, and Cn.
SELECT-M(P, <C1,…,Cn>)	SELECT-M relation indicates that P-Node must be configured with M selective nodes from its child nodes (C1,…, and Cn).

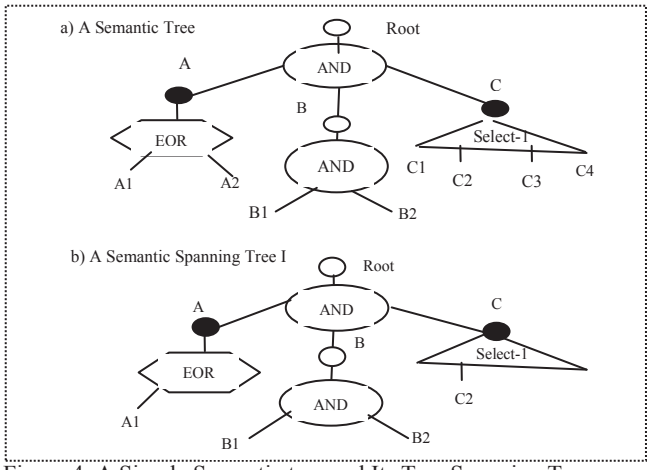


Figure 4. A Simple Semantic tree and Its Two Spanning Trees

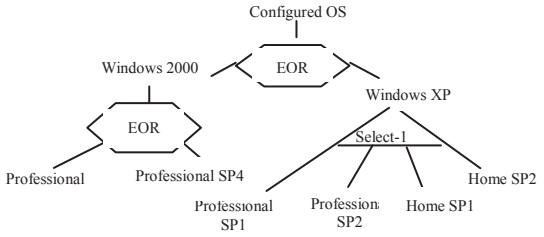


Figure 5 A Sample Model for Component Environment Configuration

As shown in Figure 4(a), a simple semantic tree example is presented with one root node and three parent nodes (A, B, and C). Each has one relation. One corresponding semantic spanning trees are given in Figure 4(b). An algorithm is given in Figure 6 to find out a semantic spanning tree for a given smantic tree model.

```
Semantic-Spanning-Tree(GST-Node, NSPT)
{
if GST -Node is a leaf node, then GST –Node →NSPT; // add this leaf node into NSPT
        return
else add GST -Node into NSPT
        switch (GST -Node"s relation) {
        case "EOR": pick a GST-Node"s child node (say Ci);
                Ci →NSPT // add into NSPT
                Add GST-Node"s link to Ci →ESPT
                Semantic-Spanning-Tree(Ci, NSPT);
                break;
        case "Select-1": pick a GST-Node"s child node (say Ci)
                Ci →NSPT // add into NSPT
                Add GST-Node"s link to Ci →ESPT
                Semantic-Spanning-Tree(Ci, NSPT);
                break;
        case "AND": GST-Node"s child nodes →NSPT
                Add all its links to its child nodes →ESPT
                Loop for each child node (say Ci) and do:
                Semantic-Spanning-Tree(Ci, NSPT);
                break;
        default "NOT": pick a GST-Node"s child node (say Ci)
                Ci →NSPT // add into NSPT
                Add GST-Node"s link to Ci →E SPT
                Semantic-Spanning-Tree(Ci, NSPT);
                break;
        }
}
```

Figure 6. An Algorithm for Finding A Semantic Spanning Tree

4.2. Modeling for Configuration Testing in Configurable Component-Based Software

Since 2009, we used two software testing classes and two master project teams to apply the semantic tree model and spanning tree concept onto one in-house-built component-based system and several open-source configurable frameworks and platform servers in San Jose State University. Here, we provide some examples to demonstrate its effectiveness in modeling various environment configurations, diverse configurable organizations, and configuration functions.

4.2.1 Modeling Diverse Configurable Environments

All commercial software and components must be executed in a certain operation environment Figure 5 shows a simple semantic tree example which presents different configurable operating system environments for software. In the real world, we can use this model to consider all required configurable hardware and software elements (or entities) in a product's operation environment. They include different configuration selections in network protocols, device drivers, diverse operating systems and their versions, multimedia and third-party dependent technologies. As shown in [16], it is very important and necessary to have a model like semantic tree to present diverse configurable environments in software installation testing. Similarly, in a component-based software production line, test engineers also need a semantic tree model to perform test modeling and test complexity of these diverse configuration environments. Since each configurable environment usually requires a set of environment-oriented test scripts to set up so that system function and performance testing can be conducted properly.

4.2.2. Modeling Configurable Structures/Organizations

In a configurable component-based system, the program can be configured by using selective components and parts to form different organizational structures. In this case, software testing must address and cover these diverse structures to make they are adequately covered whenever they are required. In 2009, we have used the semantic tree model to present the different configurable organization structures for a configurable component-based elevator simulation system, which is developed by SJSU students using Java-based component technology. This system provides a set of configurable components, including Floor Panel, Door, User Panel, and Elevator Car components. A user interface is provided to its customer to support a user to select and build diverse elevator system instance based on their need. For example, a user can configure a Door component with two models: a) Single-Door and b) Double Door. Similarly, the user can configure other parts of an elevator system. For example, Figure 7 shows the semantic tree model for the Lift component (known as a ICar) of the elevator system.

4.2.3. Modeling Diverse Configuration Functions

Today most configurable programs (or frameworks) provide rich configuration service functions that allow users to make many configuration decisions to configure different product instances. All of them provide an Administration Console with a graphic user interface to support user's selections in setting diverse configurations. To test the GUI-based configuration service functions, we can apply many existing black-box test methods, such as decision table, category-partition, and boundary value analysis. However, based on our testing experience, students entered the difficulty in dealing with the configuration-oriented test coverage for the provided configuration service functions; because these methods are not designed to easily address the diverse configuration selections and decisions. They found that using the semantic tree model is more effective to address and model the test problems in software configuration functions in configurable software. Due to the space limitation, we report the detailed finding in our future publications.

4.3. Configuration Model Identification and Generation

It is important to have some systematic way to specify diverse configurations to support test modeling, complexity and coverage analysis. Although there are well-established software analysis and design models, such as UML, they are not suitable to present the diverse configurations in configurable software in terms of environments, organizational structures, and configuration functions. The proposed semantic tree model provides an effective modeling tool to support engineers to perform software configuration analysis and specification for configurable component-based system in the 3-dimensional configuration space. The *first* approach is a static specification-based approach, in which engineers use the semantic tree model to specify and model the configurations in three steps:

1. Identifying and modeling configuration environments.
2. Identifying and modeling configurable component-based organizational structures (or architectures) based on the customer needs.
3. Identifying and modeling configuration functions, including their sectional options.

Clearly, when the given software supports complicated configurations, this approach becomes tedious. Therefore, the second approach is more dynamic and systematic one, in which some built-in dynamic configuration discovery and tracking capability will be provided in configurable software. With this capability, dynamic configuration decisions in environments, organization structures, and configuration functions can be tracked and analyzed for the purpose of test modeling, test complexity analysis, and test coverage measurement.

5. Configuration Test Criteria and Complexity

To adequately test configurable component-based software (***P***), engineers must answer the following questions:

- Have we adequately checked ***P*** under each configurable environment?
- Have we adequately tested ***P*** with each of configurable organizational structures?
- Have we adequately tested ***P*** for each of its provided configuration service functions?

To answer these questions, engineers need well-defined test criteria and an effective test coverage analysis solution. This section defines the test criteria based on the semantic tree model given in Section 3. In addition, two configuration test complexity evaluation approaches are presented.

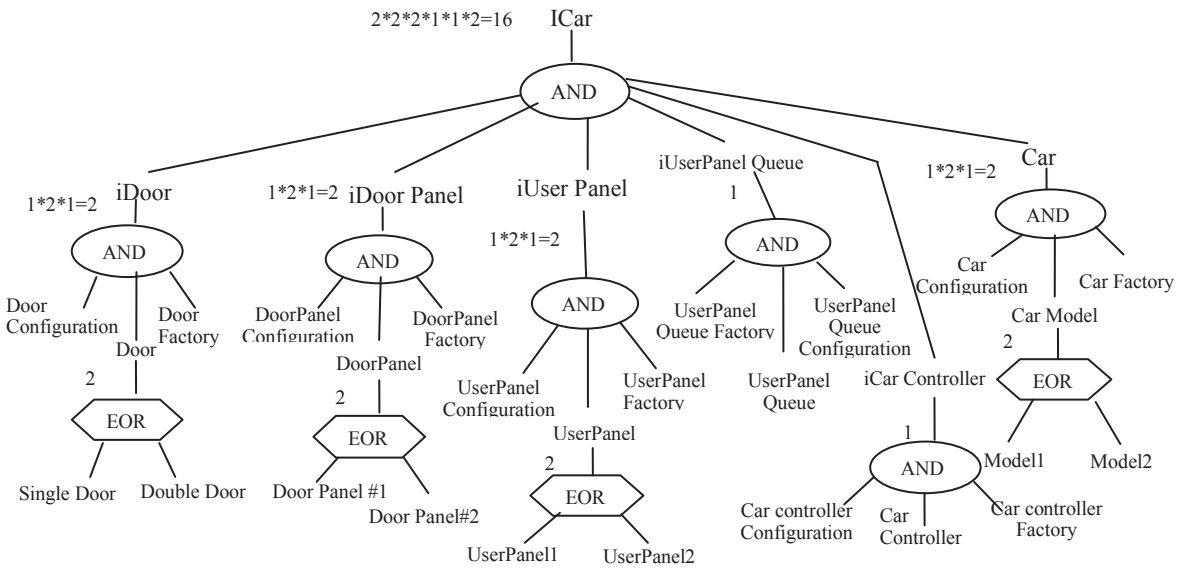


Figure 7. A Semantic Tree Model and Test Complexity for Configurable Organization Structures of the Elevator Car (ICar)

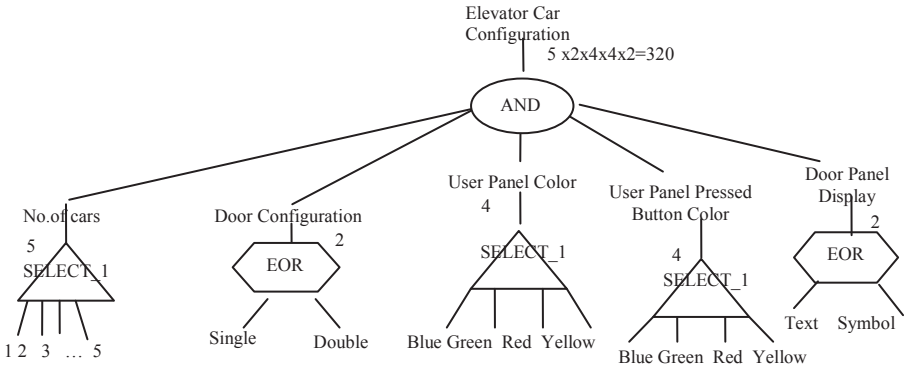


Figure 8. A Semantic Tree Model and Its Test Complexity of One Configuration Functions for the Elevator Car (ICar)

5.1. Test Criteria and Complexity for Configuration Environments

When configurable software (say ***P***) can be operated under different configurable environments, it must be validated under diverse configured environments to achieve certain environment test criteria for vendors and customers. For example, if a customer requires P to be functioned properly in a set of specified operation environments, then *P* must be validated under these environments.

Let's use the semantic tree model (G_E) to present the diverse configurable environments of software *P*. G_{SPE} be the set of semantic spanning trees of G_E. Since each configured operation environment for ***P*** can be modeled as a semantic spanning tree, we can define the test criteria for configuration environments of ***P*** below.

Test criterion for Single-Operation-Environment:

- *For any configured operation environment, this test criterion can be achieved only when software **P** has been tested under this configured environment.*

Test criterion for All-Operation-Environment:

- *This criterion can be achieved only when software P has been validated under each of the configurable environments.*

Clearly, setting up these various configurable environments require engineers efforts to prepare and execute the set-up scripts. Using the semantic tree model G_E for ***P***, engineers can easily prepare the environment set-up scripts based on its semantic spanning trees in G_{SPE}. Hence, the test environment complexity, represented as $T\text{-}complexity_P(G_E)$ of ***P*** can be computed below.

$$T\text{-}complexity_P(G_E) = \text{No. of semantic spanning trees in } G_{SPE} \quad (1)$$

This complexity formula enables to engineers to figure out the required number of pre-test scripts for environment configuration and set-up. This will be useful for test planning in test cost and complexity analysis.

5.2. Test Criteria and Complexity for Configurable Organization Structures

For a configurable component-based software P, each configurable organizational structure directly corresponds to one of its semantic spanning trees of G_S, For a software vendor of P, it is very important to validate its instances to cover its diverse organizational structures.

Since each semantic spanning tree in G_S represents one of its configured organization structures, it is necessary for us to understand and define the adequate test criteria for validating its various organizational (compositional) structures. Here, we

provide test criteria for configurable organization structures of P based on a semantic tree model. Let's use $G_S = (N_S, E_S, R_S)$ as a test model to present its configurable organizations. $\boldsymbol{G_{SPT}} = \{\boldsymbol{G_{SPT\text{-}i}} \mid i = 1,..n\}$, where $\boldsymbol{G_{SPT\text{-}i}}$ is a semantic spanning tree *of* $\boldsymbol{G_S}$. Let *Ts* be the test set for *P*.

Adequate test criterion for Single-Organization-Structure:

- *For a single organization structure modeled as a semantic spanning* $\boldsymbol{G_{SPT\text{-}k}}$ *of* $\boldsymbol{G_S}$*, this test criterion is achieved if and only if* T_S *of P has been exercised onto at least one deployed instance Pi with* $\boldsymbol{G_{SPT\text{-}i}}$ *as its organization structure.*

Adequate test criterion for All-Organization-Structures:

- *This test criterion is achieved if and only if* T_S *of P has been exercised onto the deployed instances of P configured with each spanning tree in* $\boldsymbol{G_S}$.

To support the evaluation of test complexity of diverse configurable organization structures, we provide a hierarchical computation method below. For any parent node N_{Pi} in N_S of G_S, its organizational configuration complexity can be computed based on its semantic relation with child nodes. Let *T-complexity*(N_{Pi}) be the configuration complexity for its organization structures. It can be computed below.

- If the node N_{Pi} has the EOR semantic relation with its child nodes, then its configuration complexity can be computed below.
  $T\text{-}complexity_C(N_{Pi})$
  $= T\text{-}complexity_C(N_{C1}) + T\text{-}complexity_C(N_{C2})$ (2)
  Where N_{C1} and N_{C2} are the two child nodes of N_{Pi}
- If the node N_{Pi} has the AND semantic relation with its child nodes, then its configuration complexity can be computed below.
  $T\text{-}complexity_C(N_{Pi}) = \prod (T\text{-}complexity_C(N_{Cj}))$ (3)
  Where j = 1, …, n, and N_{Cj} is a child node of N_{Pi}.
- If the node N_{Pi} has the SELECT-1 semantic relation with its child nodes, then its configuration complexity can be computed below.
  $T\text{-}complexity_C(N_{Pi}) = \sum (T\text{-}complexity_C(N_{Cj}))$ (4)
  Where j = 1, …, n, n is the number of its child nodes, and
  N_{Cj} is a child node of N_{Pi}.
- If the node N_{Pi} has the SELECT-M semantic relation with its child nodes, then its configuration complexity can be computed below.
  $T\text{-}complexity_C(N_{Pi})$
  $= \sum (T\text{-}complexity_C(N_{Cj}) * n!/(m!(n-m)!))$ (5)
  Where j = 1, …, M, N_{Cj} is one of M selected child nodes of N_{Pi}.from its N child nodes.

Following these formula, engineers can easily implement an automatic solution to compute the test complexity for diverse configurable organization structures in any given configurable component-based software. Figure 7 presents the detailed test complexity of different configurable organization structures for Elevator Controller (ICar), which is a composite component in the elevator simulation system. For example, a Door component with an EOR relation to its two child nodes (Single Door and Double Door) has to be checked for two different settings. Hence, its test complexity is 2. The iDoor component has a AND relation with its three child nodes, including Door component. Its test complexity for different compositions is 2 based on the formula in (3). The test complexity for the Elevator Controller (ICar) is 16.

Similarly, the test complexity computation approach can be used for a semantic tree model to present diverse configuration functions in the elevator simulation system. Figure 8 shows the detailed results of test complexity for testing function configurations for the Elevator Controller (ICar) based on the same approach. As shown in Figure 8, its test complexity is 320.

6. Case Study Results on Configuration Testing

In 2010, we used our students in software quality testing class (CMPE287) to conduct a case study, where 10 student groups are required to conduct a term-project to validate the given component-based elevator systems. One of the tasks is to use the semantic tree model to support configuration-based testing, so they are able to understand the testing complexity and challenges in testing configurable component-based software.

In this case study, we focus on the following items:

- Discover the semantic tree model for the selected configurable software to see the effectiveness of the model in presenting different configurations, including environments, organization structures, and configuration functions.
- Identify, understand, and analyze the test complexity of configuration-based testing in configurable component-based systems.

Table 2. The Semantic Tree for Organization Structures and Its Complexity

Semantic Tree for Organization Structures	# of Spanning Trees	# of Nodes /# of Leaves	# of Links	Max Height	# of EOR	# of AND
Elevator Semantic Tree	64	54/36	53	4	6	12
ICar Semantic Tree	16	33/22	32	3	4	7

Table 3. The Semantic Tree for Configuration Functions and Its Complexity

Semantic Tree for Configuration Functions	# of Spanning Trees	# of Nodes /# of Leaves	# of Links /Max Height	# of EOR	# of AND	# of SELECT-1
Elevator Configuration	640*256	47/37	46/3	4	3	6
ICar Configuration	320	23/17	22/2	2	1	3

As shown in Figure 7, the semantic tree model for ICar has 33 nodes. Among of them, there are 22 leaf nodes, and 10 parent nodes. In addition, there are two different types of semantic relations, including EOR and AND. And its test complexity of various configurable organization structures is 16 since the semantic tree has 16 different spanning trees. Table 2 shows the detailed complexity of the semantic tree model of the Elevator Simulation System. Its total test complexity is 64, which presents the total number of different configured structures for the elevator system. Hence, while validating this software, a vendor's engineers must test its deployed instances to cover its configurable organization structures. In practice, they can achieve the defined adequate test criteria (discussed in Section 5) in an incremental approach. For example, whenever a customer is deployed one instance, its configured system organization structure (or component composition structure) will be recorded. This idea has been implemented in one of our master project. For this system, we

found that we need to develop 64 scripts to set up and cover different organization structures so that the deployed system instance can be tested with certain adequate function test set using the existing test methods. Table 3 shows the related semantic tree model for configuration functions in the system based on the provided system user interface and configuration service functions. Clearly, its complexity is much higher due to more choices are given in the semantic tree. This suggests the configuration testing could be very complicated. More test automation research work for configurable software testing is needed.

7. Conclusions and Future Work

Although there are numerous papers addressing how to construct configurable software and components, only a few papers discussed how to test configuration features and structures of component-based software. This paper uses a model-based approach to discussing the relating issues, challenges, and test process. It applied a semantic tree model as a test model to present and analyze the diverse configuration environments and configurable structures in component-based systems. In addition, a set of adequate test criteria configurations has been defined. Furthermore, some case study results are reported demonstrate its effectiveness and application in test modeling and test complexity analysis. Currently, we are developing a test automation solution to support automatic testing of configurable component-based software. The future extension of this research is to study how to use a model-based approach to addressing regression testing issues and challenges in configurable features and services in SOA applications.

Acknowledgement

This research project was funded by Sun Microsystems in 2009, and Dr. Gao is supported by Tsinghua National Laboratory for Information Science and Technology (TNLIST) in 2010.

References

[1] Jerry Gao, Jacob Tsao, and Ye Wu, Testing and Quality Assurance of Component-Based Software, Artech House Publishers, 2003.

[2] J. M. Nogiec, "A configurable component-based software system for magnetic field measurements. IEEE Transactions on Applied Superconductivity", Volume 16, Issue 2, June 2006.

[3] Jonathan E. Cook, Jeffrey A. Dage, "Highly reliable upgrading of components", ICSE 1999.

[4] Roman Lysecky and Frank Vahid, "A configurable logic architecture for dynamic hardware and software partitioning", Proceedings of the conference on Design, automation and test in Europe - Volume 1, Feb. 2004.

[5] Hokkaido Hakodate, "Configuration and dynamic reconfiguration of component-based applications with Microsoft .NET", Proceedings of The Sixth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, 2003.

[6] D.B. Stewart, R.A. Volpe, P.K. Khosla, "Design of dynamically reconfigurable real-time software using port-based objects. Software Engineering, IEEE Transactions on Volume 23, Issue 12, Dec 1997.

[7] Thirunavukkarasu Sivaharan et al. "GREEN: A Configurable and Re-configurable Publish-Subscribe Middleware for Pervasive Computing". On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE, Springer Berlin/ Heidelberg, 2005.

[8] L. F. Friedrich, et al., "A Survey of Configurable, Component-Based Operating Systems for Embedded Applications", IEEE Micro, Vol. 21, No. 3, May/June 2001.

[9] Jonathan Vincent, et al., "Principles of Built-In-Test for Run-Time-Testability in Component-Based Software Systems", Software Quality Control, Volume 10, Issue 2, 2002.

[10] R. S. Freedman, "Testability of Software Components", IEEE Transactions on Software Engineering, 17(6), June 1991.

[11] Jerry Gao and Ming-Chih Shih, "A Component Testability Model for Verification and Measurement", The Proceedings of COMPSAC2005, Edinburgh, Scotland, July 25-28, 2005.

[12] Jerry Gao, Raquel Espinoza, and Jingsha He, "Testing Coverage Analysis for Software Component Validation", Proceedings of COMPSAC2005, Edinburgh, Scotland, July 2005.

[13] S. H. Edwards, "Black-Box Testing Using Flowgraphs: An Experimental Assessment of Effectiveness and Automation Potential," Journal of Software Testing, Verification, and Reliability, Vol. 10, No.4, December 2000.

[14] Stephen H. Edward, "A Framework for Practical, Automated Black-box Testing of Component-Based Software", Journal of Software Testing, Verification and Reliability, Vol. 11, No.2, June 2001.

[15] Jerry Gao, et al, "On Building Testable Software Components", Proceeding of International Conference on COTS-Based Software Systems, Orlando, Feb. 2002, Springer's Lecture Note in Computer Science (LNCS).

[16] Jerry Gao, Karen Kwok, and Todd Fitch, "Model-Based Test Complexity for Software Installation Testing", The proceedings of SEKE2008, San Francisco, July 2008.

[17] Leonardo Mariani, Mauro Pezze. "A technique for verifying component-based software", Proceedings of TACoS2004, Electronic Notes in Theoretical Computer Science Volume 116, 19, January 2005, Elsevier B.V.

[18] Venkita Subramonian, et al. "The Design and Performance of Configurable Component Middleware for Distributed Real-Time and Embedded Systems", The Proceedings of the 25th IEEE International Real-Time Systems Symposium, 2004.

[19] Jerry Gao, et al, "Building Testable Software Components-Approach and Its Experimental Results", The proceedings of SEKE2008, San Francisco, July 2008.

[20] Yves Le Traon, et al, "Self-testable components: from pragmatic test to design-for-testability methodology", The Proceedings of TOOLS, Nancy, France, 1999.

[21] Xiao Qu, Myra B. Cohen, and Gregg Rothermel, "Configuration-aware regression testing: an empirical study of sampling and prioritization", ISSTA 2008.

[22] B. Robinson and L.White, "Testing of User-Configurable Software Systems Using Firewalls", ISSRE 2008.

[23] Il-Chul Yoon, et al, "Effective and Scalable Software Compatibility Testing", ISSTA 2008.

[24] D. M. Cohen, et al, "The AETG System: An Approach to Testing Based on Combinatorial Design," IEEE Transactions on Software Engineering, 1997.

[25] M. B. Cohen, J. Snyder, and G. Rothermel, "Testing across configurations: implications for combinatorial testing", SIGSOFT Software Engineering Notes, 2006.

[26] M. B. Cohen, et al., "Interaction testing of highly-configurable systems in the presence of constraints", ISSTA 2007.

[27] E. J. Weyuker, "Testing Component-based Software: A Cautionary Tale", IEEE Software, 15(5): 54-59, 1998.

Data Uncertainty Model for Mashup

Xin Gao
School of Electronics Engineering and Computer Science
Peking University
Beijing, China
gaoxin54@gmail.com

Wenhui Hu, Wei Ye, ZHANG Shi-kun
National Engineering Research Center for Software Engineering
Key laboratory of High Confidence Software Technologies (Ministry of Education)
Peking University
Beijing, China
xiaodiahu@gmail.com

***Abstract*—Mashup is a new kind integration application and users can compose related services as components to build new application—the mashups. Now the services on the web have different degrees of data uncertainty, including data error, stale data, and improper data processing and so on. We provide a data uncertainty model for mashup component which is assessed in the space of the homogeneous components that have the same functionality, and then we give the uncertainty computation mechanism of mashups based on the data uncertainty of mashup components and the computation sequence for different composition relationships.**

Keywords-component; mashup; data uncertainty

I. INTRODUCTION

In Web development, a Mashup is a Web application that combines data from more than one source into a single integrated tool. The term Mashup implies easy, fast integration, frequently done by access to open API's and data sources to produce results data owners had no idea could be produced[1]. And mashups are applications developed by integrating content and functionality sourced from the Web. Mashups typically integrate heterogeneous elements available on the Web, such as RSS/Atom feeds, Web services, content scraped from third-party websites, or widgets (such as Google Maps)[2]. Different kinds of mashups reuse user interface (UI) components to build the composite application's UI, leverage and require external computational services, or simply integrate multiple plain data sources. So web environment has been required to provide data services with high quality.

With more and more sources of web service, the number and types of web service increase significantly. Now the services that we can get from the Internet include maps, e-mail, pictures, weather, traffic, search, etc. From the QWS Dataset [3,4,5] and ProgrammableWeb [6], the statistics of web service shows that there are a lot of choices in each type, and the overall amount of web service increases all the time. So we regard these services that share the same functionality as homogeneous services. There are thousands of web services, and also a large number of mashup services for the users, which build a solid base for related applications. From the point of view of the source of web service, it has changed from the original single site to numerous web application's open APIs and service supporting architectures like SaaS, SOA, and Cloud computing. So the users can access to the original stand-alone local application in the form of service by the service framework like Axis, and also can access to the core data and applications through SaaS platforms and the OpenAPI of flickr, Google and yahoo on the web.

So user can use the mashup tools to build the new application, just so-called situational applications—that is, applications where the developer is also the final user and that serve a highly focused purpose. Situational applications typically aim to answer a precise query over a limited but heterogeneous data space. Their quality, therefore, depends strongly on the information that different integrated components can provide. As the promotion of mashup applications, there is one issue that has taken more attentions and is unavoidable: data uncertainty.

The uncertainty of the data source is very common on the web. We can see the web as the largest database of human society. This database is variable and uncertain because of numerous sources, complex structure, frequent change and dynamic transmission. The data uncertainty exists as inconsistent data, fault data, data variance, service instability and the impossible traceability of the change of data. We focus on the variance of web data which can be caused by error data source, unsuitable data process and stale data. So there are always many alternate results for the same data. For example, yahoo search service and Google search service have different results for the same keyword by a compare web site [7]. So the web can also be viewed as the largest uncertain database, and uncertain homogeneous service is an important part. For example, there are many services from travel sites which provide hotel information of scenic spots. But because of different information source and data update mechanism of these web sites, there may be some differences of price and room number among the services, and the uncertain data will mislead the travelers. Now most of service selection methods only focus on QoS strategy without considering the uncertainty aspect, so it can't meet the requirement of providing users with reliable service in web environment. The mashup tools use these uncertain services as mashup components to build mashup applications, and it needs corresponding solutions for the data uncertainty of mashup. So data uncertainty is crucial for both components and composition in mashup. Assessing a mashup's data uncertainty requires understanding both

component data uncertainty and the effect that the composition has on the final mashup's overall data uncertainty.

In this paper, we introduce a data uncertainty model for mashup components, analyze different types of composition in mashup, and then define the calculation of data uncertainty for mashups based on the relationships of mashup components in the mashups. In the following parts of this paper, section 2 describes related work of mashup and data uncertainty. Section 3 describes the data uncertainty of mashup components, including the model to describe uncertain mashup components and the mechanism to access the data uncertainty of components. Section 4 introduces the composition of mashup components. Section 5 describes the data uncertainty of mashups, including the model describing the mashups and the mechanism to access the data uncertainty of mashups. Section 6 makes the conclusions and discusses the future work.

II. RELATED WORK

In traditional database applications, the existence and accuracy of the data is deterministic. In recent years, as technology advances, people deepen the understanding of data acquisition and processing technology. Now uncertain data problem has widely attracted public's attention, and many researchers have focused on uncertainty in data processing. Literature [12] describes the background of uncertainty in the application of the data, and summarizes the challenges of uncertainty data management. Literature [13] gives an overview of algorithms and application of uncertain data management techniques. From the point of view of uncertain systems, Purdue University's Orion project [10] tries to design a general-purpose uncertain database system; the Trio project at Stanford University [11] studies the lineage according to the analysis of uncertainty. In the model aspect, the most commonly used model is possible world model [8, 9], which evolves a lot of database instances (called possible world instances) from uncertain database. There are also some researchers who focus on data integration, and literature [14] designs data-integration system which handles uncertainty at three levels: semantic mapping, uncertain data and uncertain queries. So based on these progresses of data processing on uncertain data, we can carry out the work to handle data uncertainty of web service regarded as the data source.

Currently there are a number of Mashup tools. Damia[15] is a Mashup tool provided by IBM, which allows the users to assemble data feeds from Internet and enterprise data sources. Yahoo Pipes [16] is a web-based tool provided by Yahoo. The users can build mashup applications by aggregating and manipulating data from web feeds, web pages, and other services. A pipe is composed of one or more modules, each one performing a single task like retrieving feeds from a web source, filter, sort or merge feeds. Popfly[17] is a web-based Mashup application by Microsoft. It allows the users to create a Mashup combining data and media sources. The Mashup is built by connecting blocks. Apatar[18] is a Mashup data integration tool that helps users to integrate desktop data with the web. Users install a visual job designer application to create integration schemas called DataMaps. MashMaker[19] is a web-based tool by Intel for editing, querying and manipulating web data. So we can see that many mashup tools focus on the integration of web data sources, and the mashups is the result of data integration of web data from the data aspect.

There are only a few researches on the data uncertainty of mashup application. The UQBE [20] is a mashup tool for non-programmers that supports query-by-example (QBE) over a schema made up by the user without knowing the schema of the original sources. Based on automated schema matching with uncertainty, the UQBE system returns the best confident results. MashRank [21] is a mashup tool that treats ranking as a first-class citizen in mashup construction, and allows for rankjoining Web sources with uncertain information. Both these two tools which consider the uncertainty problem just study the data uncertainty of the ranking data and schema matching, but don't study the composition of uncertain data and the data uncertainty of mashups. So the research on data uncertainty in mashup is in the initial stage, and more and more people will take attention to this topic.

III. DATA UNCERTAINTY OF MASHUP COMPONENTS

On the web, there are many alternates for each kind service as the result shown in ProgrammableWeb, and we refer to them as homogeneous services. And these services can be used as the components of mashup, and data uncertainty of these homogeneous services is just the main data uncertainty of mashup components. We think that the data uncertainty of mashup component is relative to their alternate components, and the data uncertainty is considered in the space of all the mashup components which share the same functionality. We see these mashup components as the tuples in uncertain database, each of which has its information and its confidence possibility that reflects component's uncertainty. We think that the data uncertainty of component is caused by component's quality to provide data service and user's distrust among all the alternate choices. So we can get the information of the confidence degree of the component based on user's decision when the users participate in the selection of homogeneous components, which can give the uncertainty assessment of the component. Based on the assessments, we can build uncertain model for these homogeneous mashup components.

A. Uncertain component model

We consider the data uncertainty of mashup component from two aspects: the ability to provide data service and user's confidence of component's data. Before the data processing of the data uncertainty of mashup components, we first need to propose the syntax description of mashup component. Now services for mashup are mainly described by the basic information, such as service name, operation and corresponding input / output, etc. At the same time, we should also consider the quality of the component and the attributes which can describe QoS (quality of component). Then we can provide the components which meet user's requirements based on QoS and user decision. Therefore we need a unified component description model with additional QoS information, and the service model is extended to following model:

Definition 1: According to the consideration of QoS, a component can be identified as a six-tuple set:

MC={ComponentName, Op, Input, Output, Profile, Qset}

Where ComponentName is the name of the component, Op is the operation to get corresponding data, Input / Output is the input and output of the operation, Profile describes component auxiliary information. And Qset is a set of attributes which represents the qualities of component to provide data service.

$$\text{Qset}=\left\{\begin{matrix}\text{Response Time，Availability，Throughput，}\\ \text{Reliability Success ability Latency}\end{matrix}\right\}$$

QoS attributes help determine which of the available components is the best and meets users' requirements. The meanings of the attributes are as follows：

a) Response Time：Time taken to send a request and receive a response
b) Availability：Number of successful invocations/total invocations
c) Throughput：Total Number of invocations for a given period of time
d) Reliability：Ratio of the number of error messages to total messages
e) Success ability：Number of response / number of request messages
f) Latency：Time taken for the server to process request

Considering the problem of uncertain data management, we apply probabilistic database model to create uncertain service model. We wish to model probabilistic information using a probability space whose possible outcomes are all the conventional instances. The finiteness of D implies that there are only finitely instances $I \in D$. By finite probability space we define a probability space (D, P[]) in which D is finite. We shall use the equivalent formulation of pairs (U, p), where U is the finite subset of D and the probability assignment $p: U \rightarrow [0,1]$ satisfies $\sum_{w\in D} p(w) = 1$ and $\sum_{w\in U} p(w) = P[U]$. Therefore, we use probabilistic model to map data uncertainty by adding the probability attribute to the description of components. This uncertain component model is as follows:

Definition 2: An uncertain component model can be identified as a four-tuple set:

$$\text{MC}=\{\text{ComponentName, N, Qset, P}\}$$

Among them, ComponentName stands for the service; N is a number which stands for the times that this service has been chosen, and it is also the subjective information of user decision in the model; the Qset is the same as definition 2; the additional P represents the assessment of the credibility of each component. So D_S represents the domain of all the homogeneous components, and D_u represents the subset of D_S, then satisfied:

$$\sum_{\text{MC}\in D_u} p(\text{MC}) = P[D_u] \text{ and } \sum_{MC\in D_S} P(MC) = 1$$

B. Uncertainty component Model Building

From the definition of uncertain component model, we can see that how to compute probability attribute is the core work, and it relates to the problem of what is certainty and what is uncertainty. In our view, the origin of certain in the web is credibility assessment by users. Mashup component is a kind of software application in web environment, and is a dynamic changing data carrier for numerous users' ideas and thoughts. So for the component, the user is the root cause of its existence, and the user's knowledge and sense of service determines the credibility of the service. And from the further analysis, user awareness of the component's credibility comes from user's intuitive understanding of component's quality properties which represent the objective uncertainty of component and user's requirements which represent the subjective uncertainty of component. Therefore, the service's credibility can be evaluated based on the combination of these two kinds information of component. In order to support uncertain data assessment of homogeneous components, we need to establish the appropriate process to meet the requirements of component registration and management, component accessing and uncertainty modeling. Our approach sets the probability attributes in uncertain service model as follow: we first calculate the weight of each quality attribute according to statistics on the user's selection through the neural network approach; and then calculated quality of component; finally, according to the quality of components we get the credibility of uncertain component model which is the value of corresponding probability property.

Set up a group of homogeneous components $MC = \{mc_1, mc_2, mc_3 \dots mc_n\}$, and users give the choice based on personal feeling. Then we can use the number of choices of homogeneous components from the user group to calculate the services QoS attribute weights by neural network.

First, we need to collect and organize the user's choice of homogeneous components. Within a certain period of time the user's choice is a collection of components, which can be expressed as $\text{UserChosen} = \{N_{i_1}, N_{i_2}, N_{i_3} \dots N_{i_m}\}$, where N_{i_k} means that the i_k component has been chosen N_{i_k} times. Use matrix to organize QoS attributes of selected components as follow. It is the matrix in which each row represents a single component metric, where each column represents a single QoS attribute and the QoS attributes include Response Time，Availability，Throughout，Successability，Reliability and Latency.

$$\text{QosM}=\begin{bmatrix} RT_1 & A_1 & T_1 & S_1 & R_1 & L_1 \\ RT_2 & A_2 & T_2 & S_2 & R_2 & L_2 \\ \dots & & & \dots & & \dots \\ \dots & & & \dots & & \dots \\ RT_k & A_k & T_k & S_k & R_k & L_k \end{bmatrix}$$

Then set QosM^T, which is a 6*n matrix, as the input of neural network algorithm unit which means that set every QoS attribute as a node in the neural network and the corresponding value of each component is an assignment of the node; set the times of each component being selected in the form of $[N_1, N_2, \dots, N_n], \sum_{i=1}^{n} N_i = m$, which is 1*n matrix, to be the target; at last, construct the training sample. Through the training of neural network, we can get the weight for each QoS attribute. Then each QoS weight divides the max of them, and we can get the set of relative weights：

$$W = \{W_1, W_2, W_3, W_4, W_5, W_6\}$$

which are corresponding to the six attributes of QoS. Regarding the evaluation mechanism of service credibility, we view homogeneous components as an independent set and

compute relative quality of each component in the corresponding homogeneous component set as follow:

$$\overline{RT_i} = 1 - \frac{RT_i}{\sum_{k=1}^{m} RT_k};\ \ \overline{A_i} = \frac{A_i}{\sum_{k=1}^{m} A_k}$$

$$\overline{T_i} = \frac{T_i}{\sum_{k=1}^{m} T_k};\ \ \overline{R_i} = \frac{R_i}{\sum_{k=1}^{m} R_k}$$

$$\overline{S_i} = \frac{S_i}{\sum_{k=1}^{m} S_k};\ \ \overline{\mathrm{L}_i} = 1 - \frac{\mathrm{L}_i}{\sum_{k=1}^{m} L_k}$$

Because response time and latency are smaller, the quality of component's data is better, so they are represented by their complements. Thus quality of component is formed by these relative qualities and the relative weights, just as follow:

$$\overline{QoS_i} = \{\overline{RT_i}, \overline{A_i}, \overline{R_i}, \overline{T_i}, \overline{S_i}, \overline{\mathrm{L}_i}\}, 1 \le \mathrm{i} \le \mathrm{m}$$

$$Q_i = \overline{QoS_i} \cdot W^T = \overline{RT_i} * \mathrm{W}_1 + \overline{A_i} * \mathrm{W}_2 + \overline{R_i} * \mathrm{W}_3 + \overline{T_i} * \mathrm{W}_4 + \overline{S_i} * \mathrm{W}_5 + \overline{\mathrm{L}_i} * \mathrm{W}_6$$

When we get the relative quality $\mathrm{Q_i}$, the degree of credibility which is also the attribute $\mathrm{P_i}$ of component can be calculated by $\mathrm{Q_i}$, in proportion to the overall relative qualities of homogeneous components, just like:

$$\mathrm{P}_i = \frac{Q_i}{\sum_{k=1}^{m} Q_k},\ \ 1 \le \mathrm{i} \le \mathrm{m}$$

IV. COMPOSITION IN MASHUP

Assessing each mashup component's data uncertainty isn't enough: the final mashup application's data uncertainty also depends on how these components are interconnected. We can assess the final applications' overall data uncertainty by aggregating the composing services' data uncertainty. Mashup's data uncertainty isn't simply an aggregation of individual component's data uncertainty. Instead, it depends on how particular components combine into a composite logic, layout, and hence user experience. By analyzing the most popular mashups published on programmableweb.com, the paper [2] identified the following typical roles:

1) Master. Even if a mashup integrates multiple components in a single page, in most cases, one component is more important than the others.

2) Slave. A slave component's behavior depends on another component: its state is mainly modified by events originating in another (master) component.

3) Filter. Filter components let users specify conditions over the content the other components show. They provide (possibly hierarchical) access mechanisms that let users incrementally select which content they want to see.

Based on these three roles, there are three basic patterns that characterize most mashup applications (see Figure 3) and highlight some mutual dependencies among the identified roles that impact mashup's data uncertainty.

1) The slave-slave pattern, in which the mashup integrates several slave components the user can interact with in an isolated fashion, without any propagation of data or events from one component to another. At startup or during runtime, users define filter conditions that steer all the slave components. The effect is that of a rather static application with very simple interaction facilities that lets users "query" the slave components' data set. Regarding the resulting mashup's data uncertainty, we assume that the filter doesn't increase the components' data uncertainty.

2) The master-slave approach, the most widely used pattern among today's mashup applications. It features all three component roles. A filter component lets users restrict the data all the other components simultaneously show. Users employ the master component to perform the main interactions with the application, such as selecting related data items. The slave component is automatically synchronized according to the selections performed on the master component, thereby visualizing the selected elements' details. With the master-slave pattern, the final application's data uncertainty could depend on the application's composition logic. Provided that master and slave are compatible in terms of data to be visualized, their integration might increase the slave's data uncertainty.

3) The master-master pattern. This is the most complete pattern, in which — in addition to suitable filter components — all integrated components are masters. All components provide interaction facilities that let users perform selections or that provide inputs that propagate to all the other components that synchronize accordingly. The master components therefore also act as slaves. From a data uncertainty perspective, the master-master pattern is similar to the master-slave pattern. If the components have different underlying data sets, situations could occur in which one component satisfies the user request, while another component can't, lowering the mashup's overall perceived data uncertainty.

We will give our data uncertainty computation mechanism based on these three roles of mashup components and the three mashup patterns. In this paper, we assume that the mashup composition performs integration at the process and presentation levels correctly. To characterize data uncertainty in the context of mashups, we focus on the data level.

V. DATA UNCERTAINTY OF MASHUPS

A. Date Uncertainty in the data sets of mashups

Data integration in mashups corresponds to a global-as-view (GAV) problem, in which the global schema is expressed in terms of views over the integrated data sources. During mashup development, the designer can inspect the attributes the components expose, as specified in the component APIs, and infer join attributes on which to base data integration.

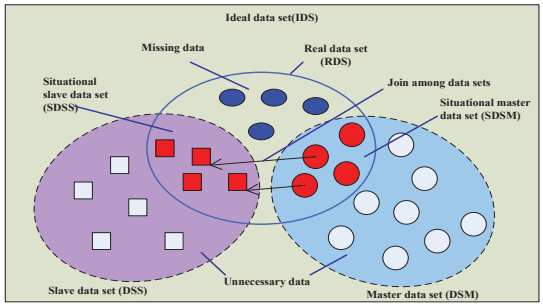


Figure 1. Data sets involved in data integration of mashup.

We can characterize data integration for mashups by categorizing the data relationship between different data sources as follows:

- Mashup applications are developed to let users retrieve and access a set of data that we call the ideal data set (IDS); it is the final data view of mashup components.
- Each component k has its own data set DSk. To fulfill the mashup requirements, a smaller portion SDSk ⊆ DSk could be sufficient. SDSk is the corresponding components' situational data set.
- The integration of all situational data sets SDSk gives the real data set RDS ⊆ IDS that the mashup provides. RDS's data uncertainty thus depends on the uncertainty of the data individual components provide.
- We can determine the mashup's data uncertainty by comparing its RDS with the corresponding IDS.

Now we analyze the corresponding data uncertainty for RDS based on the forward work on data uncertainty of mashup components and their composition relationships.

- The data uncertainty of situational slave data set is the same as the data uncertainty of slave component. And we set it as P(S).
- The data uncertainty of situational master data set is the same as the data uncertainty of master component. And we set it as P(M).
- For the missing data is not the core data part of RDS and always have no strong relations with the slave component or the master component, now we don't consider its data uncertainty.

Then the data uncertainty of RDS is calculated based on the type of Join relationship among data sets and the structure of mashup components. We assume components are sourced from the Web, we also assume they're independent of each other. Because our data uncertainty is computed for the data source among the data space which is composed of the data sources which belong to the same domain, like weather, news, sales and so on, the data uncertainty of slave component and that of master component are independent. So the data uncertainty of join relationship between slave component and master component is as follow:

$$P(MS) = P(M) * P(S)$$

$$\mathrm{P}(US|CM) = 1 - \mathrm{P}(S)$$

We describe data uncertainty of data source with the probability of the data source to be certain. So P(M) is not only the probability of master component to be certain, but also the data uncertainty of mater component; P(S) is the data uncertainty of slave component; P(MS) is the data uncertainty of the join data set between mater component and slave component. P(US| CM) is the probability that the slave component is uncertain when the master component is certain. And we will describe different ways to compute RDS's data uncertainty for different composition situations in mashups.

B. *Data uncertainty of RDS*

In the Slave-slave pattern, every mashup component is independent with each other, and the data uncertainty of mashup components is the maximum data uncertainty of all the slave components. Because we describe data uncertainty of data source with the probability of the data source to be certain, the data uncertainty of mashup components is described by P({S1,S2,…,Sn}).

$$\mathrm{P}(\{S1,S2,...,Sn\}) = \mathrm{MIN}\{\mathrm{P}(S1), \mathrm{P}(S2), ..., \mathrm{P}(Sn)\}$$

For the Master-slave pattern and Master-Master pattern, the data uncertainty is mainly to compute the data uncertainty between master component and slave component, because of the data uncertainty between master component and master component can be transformed to data uncertainty between master component and slave component. In the situation of Master-slave pattern, there are two sources of data uncertainty which are the one that the mater component is uncertain and the other one that the slave component is uncertain when the master component is certain. So the data uncertainty between master component and master component is as follow:

$$\mathrm{P}(\mathrm{M\_S}) = 1 - (\mathrm{P}(UM) + \mathrm{P}(US|CM))$$
$$= \mathrm{P}(\mathrm{M}) - \mathrm{P}(US|CM) = \mathrm{P}(\mathrm{M}) + \mathrm{P}(\mathrm{S}) - 1$$

Where P(M_S) means data uncertainty between a master component and a slave master component, P(UM) means the probability that master component is uncertain and P(US| CM) means the probability that the slave component is uncertain when the master component is certain.

We can consider Master-Master pattern to be the combination of two Master-slave patterns: a selection in one master causes the other master to act as a slave and vice versa.

$$\mathrm{P}(\mathrm{M\_M}) = 1 - (\alpha(\mathrm{P}(UM1) + \mathrm{P}(UM2|CM1)) + (1-\alpha)(\mathrm{P}(UM2) + \mathrm{P}(UM1|CM2)))$$
$$= \mathrm{P}(\mathrm{M1}) + \mathrm{P}(\mathrm{M2}) - 1$$

Where P(M_M) means data uncertainty between two master components; P(UM) means the probability that master component is uncertain; P(UMi|CMj) means the probability that one master component is uncertain when another master component is certain.

In real mashups, we can analyze the mashups into the three patterns, but we also need to set the data uncertainty computation sequence for the mashups. For the computation sequence, we have set three rules:

- Master-Master pattern is computed prior to others.
- Master-Slave pattern is computed prior to Slave-Slave pattern.
- The pairs of mashup components which have smaller data uncertainty are computed prior.

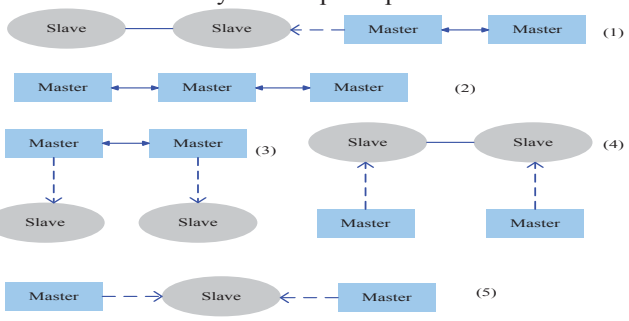


Figure 2. The situations of computation sequence

Based on master role and slave role of mashup components, we categorize the computation sequence into five situations:

- Situation 1: Slave-Slave-Master-Master

  Because there are Master-Master pattern, we compute the pair of Master-Master first. Then this pair play the master role to the second slave component, we compute them as Master-Slave pattern. At last, we compute the forward result and the first slave component as Slave-Slave pattern. The computation sequence is as follow:

  P(slave-slave-Master-Master) = P(slave-(slave-(Master-Master)))

- Situation 2: Master1-Master2-Mater3

  In this situation, we first find the master component which has the lowest data uncertainty. If Master1 or Master3 has the lowest data uncertainty, computation sequence is from the left to the right or from the right to the left. If the middle one has the lowest data uncertainty, then find the one between the other two components which has lower data uncertainty and compute this pair first.

- Situation 3: Slave1-Master-Master-Slave2

  This situation can also be seen as Slave1-Master-Slave2, and the computation sequence depends on the data uncertainty of Slave1 and Slave2. If the data uncertainty of Slave1 is higher than the data uncertainty of Slave2, the computation sequence is as follow:

  P(Slave1-Master-Master-Slave2) = P(Slave1- ((Master-Master)-Slave2))

  where P(Slave1)<P(Slave2). The situation first executes Master-Master computation and then executes Master -Slave computation.

- Situation 4: Master-slave-slave-Master

  In this situation, the computation sequence first executes Master-Slave computation and then executes Slave-Slave computation.

  P(Master-slave-slave-Master) = P((Master-slave)-(slave-Master))

- Situation 5: Master1-slave-Master2

  In this situation, there are two Master components and we assume that the data uncertainty of Master1 is higher than the data uncertainty of Master2. If Master1 and Master2 communicate separately with different parts of Slave component, we can transform the situation to Master1-slave1-slave2-Master2 situation where slave1 and slave2 are different parts of Slave component and P(slave1)=P(slave2)=P(Slave). If the parts of Slave component that Master1 and Master2 communicate separately with have the common part, the computation sequence is as follow:

  P(Master1-slave-Master2) = P(Master1-(slave-Master2))

VI. CONCLUSION

In this paper, we study the data uncertainty of mashup from two levels: mashup components and mashups. The data uncertainty of mashup component is a relative value in the set of all the homogeneous components. In the data level of mashup application, mashups can be seen as the composition of different data source with certain application relationships. So we give the data uncertainty computation of mashups based on these relationships and we show the computation sequence in different composition situations. We will further our work for the complex integration process in mashup application, analyze how the change of data uncertainty in certain mashup component affects the overall data uncertainty of mashups, and study the composition of mashup applications based on data uncertainty.

ACKNOWLEDGMENT

This work was supported by the National Natural Science Foundation of China under Granted No. 60903001

REFERENCES

[1] Mashup, http://en.wikipedia.org/wiki/Mashup_(web_application_hybrid). 2008

[2] Cinzia Cappiello, Florian Daniel, Maristella Matera, Cesare Pautasso, "Information Quality in Mashups," IEEE Internet Computing, vol. 14, no. 4, pp. 14-22, July/Aug. 2010

[3] Al-Masri,E.,and Mahmoud, Q. H., "Discovering the best web service", WWW, ACM press, New York, pp. 1257-1258(2007)

[4] Al-Masri, E., and Mahmoud, Q. H., "QoS-based Discovery and Ranking of Web Services", ICCCN, pp. 529-534 (2007)

[5] Al-Masri, E., and Mahmoud, Q.H., "Investigating Web Services on the World Wide Web", WWW, ACM press, New York, pp. 795-804 (2008)

[6] Programmableweb, http://www.programmableweb.com

[7] Compare web site, http://www.langreiter.com/exec/yahoo-vs-google.html

[8] Abiteboul, Kanellakis P, Grahne G, "On the representation and querying of sets of possible worlds", ACM SIGMOD Record, ACM press, New York, pp. 34-48 (1987)

[9] Green TJ, Tannen V,"Models for incomplete and probabilistic information", IEEE Data Engineering Bulletin, pp.17-24 (2006)

[10] "ORION: A databasesystem for managing uncertain data", in http://orion.cs.purdue.edu

[11] J. Widom, "Trio: A system for integrated management of data, accuracy, and lineage", VLDB, VLDB Endowment, Korea, pp.1151-1154 (2006)

[12] Christopher, Dan Suciu,"Management of Data with Uncertainties", Proceedings of 16th ACM conference on Conference on information and knowledge management, ACM, Lisbon, pp.3-8 (2007)

[13] C.C. Aggarwal and P.S.Yu, "A Survey of Uncertain Data Algorithms and Applications". IEEE Transactions on Knowledge and Data Engineering, IEEE Educational Activities Department, Piscataway, pp.609-623 (2009)

[14] LIU Xuan-Zhe, HUANG Gang, MEI Hong," Consumer-Centric Service Aggregation: Method and Its Supporting Framework", Journal of Software, Beijing, pp.1883−1895 (August 2007)

[15] M. Altinel, P. Brown, S. Cline, R. Kartha, E. Louie, V. Markl, L. Mau, Y.-H. Ng, D. Simmen, and A. Singh. Damia: a data mashup fabric for intranet applications. In VLDB '07, pages 1370–1373. VLDB Endowment, 2007.

[16] Yahoo Pipes. http://pipes.yahoo.com

[17] Microsoft Popfly. http:// www.popfly.com

[18] Apatar, www.apatar.com/

[19] R. Ennals and M. N. Garofalakis. Mashmaker: mashups for the masses. In SIGMOD, pages 1116–1118, 2007.

[20] Junichi Tatemura, Songting Chen, Fenglin Liao,UQBE: Uncertain Query By Example for Web Service Mashup, SIGMOD'08, June, 2008, Vancouver, BC, Canada.

Mohamed A. Soliman, Mina Saleeb, Ihab F. Ilyas.MashRank: Towards Uncertainty-Aware and Rank-Aware Mashups, ICDE, 2010.

Presenting Software License Conflicts through Argumentation

Thomas A. Alspaugh
Computer Science Dept.
Georgetown University
Washington, DC, USA
thomas.alspaugh@acm.org

Hazeline U. Asuncion
Computing and Software Systems
University of Washington, Bothell
Bothell, Washington, USA
hazeline@u.washington.edu

Walt Scacchi
Institute for Software Research
University of California, Irvine
Irvine, California, USA
wscacchi@ics.uci.edu

***Abstract*—Heterogeneously-licensed systems pose new challenges to architects and designers seeking to develop systems with appropriate intellectual property rights and obligations. In the extreme case, license conflicts may prevent a system's legal use. Our previous work showed that rights, obligations, and conflicts can be calculated. But architects benefit from fuller information than simply (for example) a list of conflicts. In this work we demonstrate an approach for presenting intellectual property results in terms of arguments supporting them. The network of argumentation provides not only an explanation of each conclusion, but also a guide to the tradeoffs available in choosing among design alternatives with different licensing results. The approach has been integrated into the ArchStudio software architecture environment. We present an illustrative example of its use.**

I. Introduction

An increasing number of development organizations are adopting a strategy in which software-intensive systems are composed of *heterogeneously licensed* (HtL) components, with different components governed by different software licenses. The components are either open source software (OSS) or proprietary software with open application programming interfaces (APIs), and are combined in an open architecture (OA) in which components with comparable interfaces can be substituted for each other [10]. Under this strategy the development organization becomes an integrator of components largely produced elsewhere, interconnected to achieve the desired result.

The resulting OA systems can achieve reuse benefits such as reduced costs, increased reliability, and potentially increased agility in evolving to meet changing needs. But rather than a single proprietary license as when acquired from a proprietary vendor, or a single OSS license as in uniformly-licensed OSS projects, the resulting system typically has no recognized single software license. Instead it has, strictly speaking, a *virtual license* [2] composed of each component's rights and obligations for that component under its governing license. The rights available for the system as a whole are the intersection of the rights sets for each component. In some cases the licenses may produce conflicting obligations and this intersection is empty, leaving a system that cannot legally be used, distributed, or modified. An emerging challenge is to realize the reuse benefits of HtL systems while managing virtual licenses to ensure that the desired system rights are available for an acceptable set of obligations.

In our previous work (summarized in Section IV) we described and implemented a novel approach for calculating conflicting obligations, unavailable rights, and virtual licenses in an architectural design context. Calculation is necessary because the number of entailments in a typical HtL system is large, the system's architecture is constantly evolving, its design-, distribution-, and run-time architectures are often distinct, component licenses evolve and components are relicensed, and the consequences of infringement can be substantial. Therefore identifying conflicts and virtual licenses through calculation is a substantial boon. But we soon realized that *explaining* them was of even greater value.

We present an approach in which arguments are used to explain the results of right and obligation calculations. The calculations proceed by elaborating a directed acyclic graph (dag) of inferences among rights to obligations for entities in the system architecture. In this work we reimplemented the software that performs the calculations so that the dag is retained in its entirety as the primary calculation product, containing within it the obligation conflicts, unavailable rights, and virtual license for the system under analysis. Then an explanation for a specific result corresponds to the traversal of a path through the dag, starting at the result in question and continuing until the question has been answered.

- *Conflicting obligations:* the traversal branches for each obligation to show the desired rights, license provisions, and architectural entities from which that obligation is produced, and at the root of the traversal shows in what ways the obligations conflict.
- *Unavailable rights:* for each such right, a traversal identifies the exclusive copyright right that subsumes the right in question, the architectural entity to which the right pertains, and why no right in the entity's license grants the right in question.
- *Virtual license:* traversals show the chains of inference by which each right and obligation is entailed by the system architecture, the stated license for each component, and the desired rights for the system as a whole.

The dag calculation algorithm follows the steps of legal

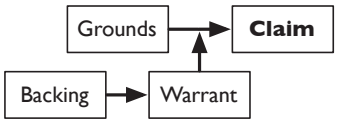


Fig. 1. A *claim*, supported by *grounds*, their pertinence to the claim justified by a *warrant*, whose validity is supported by *backing* (diagram after [14])

reasoning (formalized to support automation) by which an informed analyst would reason out the results. Thus the traversals follow inference paths that follow (in more detail) the paths by which an analyst reasons out the same conclusions.

II. Related Work

The most influential approach for structuring legal arguments is that of Toulmin, who classified the parts of arguments into claims, grounds, warrants, backing, qualifiers, and rebuttals, in a recursive structure with a diagrammatic notation outlined in Figure 1 [14]. His approach has spread beyond the area of legal arguments and is used in general rhetoric and computer science. Toulmin divides arguments into

1) *claims* asserted to be true;
2) for each claim whose truth is disputed, one or more *grounds* supporting it;
3) if it is disputed whether a claim's grounds suffice for it, then a *warrant* stating why the grounds entail the claim;
4) if the warrant is disputed, then *backing* supporting it.

If a ground or backing is disputed, then it is made the claim of a lower-level argument constructed in its support. The recursion of arguments continues as long as grounds or backings are in dispute, or until the original claim is abandoned. (Qualifiers and rebuttals address the degree of strength of arguments, and are not used in the present work.)

Hohfeld sought a theory by which to resolve the imprecise terminology and ambiguous classifications he found in use for legal relationships. In a seminal article published in 1913 and cited to the present day, he set forth a system of eight jural relations intended to express and classify all legal relationships between people. The first four regulate ordinary actions and are *right* ("may"), *no-right* ("cannot"), *duty* ("must"), and *privilege* ("need not"). Each relation has an *opposite* relation whose sense is its opposite, and a *correlative* relation whose sense is its complement. We use Hohfeld's first four jural relations as the basis of our representation of the enactable, testable provisions of software licenses (Section IV).

There has been much work on analysis of laws in AI over the past few decades. A widely-cited example is Sergot et al.'s re-expression of the British Nationality Act as a Prolog program; the resulting program applied the Act to any person's situation and characteristics to determine nationality [12].

A number of researchers have used argumentation to guide decision making, notably Haley et al. who propose an approach for using satisfaction arguments to evaluate and guide evolution of security requirements [7]. Decision choices for which no convincing argument is found are set aside in favor of choices for which stronger arguments have been identified.

III. Licensing Background

A. Intellectual Property (IP)

An individual can own a tangible thing, and have property rights in it such as the rights to use it, improve it, sell it or give it away, or prevent others from doing so, subject to some statutory restrictions. Similarly, an individual can own *intellectual property* (IP) of various types, and have specific property rights in the intangible intellectual property, such as the rights to copy, use, change, distribute, or prevent others from doing so, again subject to some statutory restrictions.

Software licenses are primarily concerned with copyrights Copyright is defined by Title 17 of the U.S. Code and by similar law in many other countries. It grants exclusive rights to the author of an original work in any tangible means of expression, namely the rights to

- reproduce the copyrighted work;
- distribute copies;
- prepare derivative works;
- distribute copies of derivative works; and
- (for certain kinds of work) perform or display it.

Because the rights are exclusive, an author can prevent others from exercising them, except as allowed by "fair use", or can grant others any or all of the rights or any part of them; one of the functions of a software license is to grant such rights, and define the conditions under which they are granted.

B. Software Licenses

Traditional proprietary licenses allow a company to retain control of software it produces, and restrict the access and rights that outsiders can have. OSS licenses, on the other hand, encourage sharing and reuse of software, and grant access and as many rights as possible.

Academic OSS licenses such as the Berkeley Software Distribution (BSD) license, the Apache Software License, and perl's Artistic License [1] grant nearly all rights and impose few obligations. Typical academic license obligations are simply to not remove the copyright and license notices.

Reciprocal OSS licenses impose an obligation that distributed modifications of reciprocally-licensed software be freely licensed under the same license. Examples are the Lesser General Public License (LGPL), Mozilla Public License (MPL), and Common Public License [1].

Some reciprocal licenses additionally require that software combined with the licensed software (for various definitions of "combined") also be freely licensed under the same license. We term such licenses *propagating*; they are also known as *strong copyleft* licenses. Examples are the General Public License versions 2 and 3 (GPLv2, GPLv3) [1].

Some OSS is *multiply-licensed*, or distributed under two or more licenses. The MySQL database software is distributed either under GPLv2 for OSS projects or a proprietary license for commercial projects. The Mozilla Disjunctive Tri-License licenses the core Mozilla components under any of three licenses (MPL, GPL, or LGPL).

C. Licenses and Software Architectures

Certain classes of architectural features affect the application and propagation of license provisions. The most common such features are listed below. A software architecture is composed of components, each of which is a "locus of computation and state" in a system, and connectors which link them and mediate interactions between them.

Software source code components—These can be

- standalone programs,
- libraries, frameworks, or middleware,
- inter-application script code such as C shell scripts, or
- intra-application script code, to creating Rich Internet Applications using domain-specific languages like XUL for the Firefox Web browser [6] or "mashups"[9].

The distinguishing characteristic of a source code component is that its source code is available and it can be modified and rebuilt. Each may have its own explicit license, though often script code connecting programs and data flows has no stated license unless the script is substantial or proprietary.

Executable components—These components are in binary form, with source code not available for access, review, modification, or possible redistribution [11]. If proprietary, they often cannot be redistributed, and so such components will be present in the design- and run-time architectures but not in the distribution-time architecture.

Software services—An appropriate software service can replace a source code or executable component.

APIs—These are not and cannot be licensed, but connections through APIs can be used to limit the propagation of some license obligations.

Software connectors—These are software elements providing a standard or reusable way of communication through common interfaces, such as High Level Architecture, CORBA, or Enterprise Java Beans. Connectors can also limit the propagation of some license obligations.

Methods of composition—These include linking as part of a configured subsystem, dynamic linking, and client-server connections. Methods of composition affect license obligation propagation, with different methods affecting different licenses. How and to what extent this occurs have not been resolved in court or in practice [5], [13].

Configured system or subsystem architectures—These are software systems used as atomic components of a larger system. Their internal architecture may contain subcomponents under several licenses, which may affect the rights and obligations for the configured (sub)system and the overall system containing it. To minimize license interaction, a configured system or subsystem architecture may be surrounded by what we term a *license firewall* [2], namely a layer of dynamic links, client-server connections, license shims, or other connectors that block the propagation of obligations.

D. Heuristics for Designing HtL Systems

HtL system designers have developed heuristics to guide architectural design while avoiding some license conflicts.

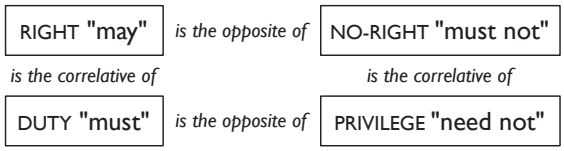


Fig. 2. Hohfeld's four basic relations

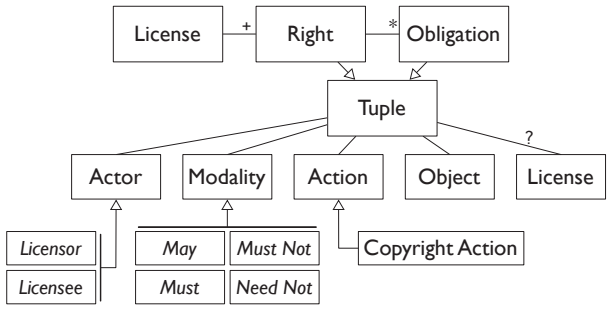


Fig. 3. Metamodel for software licenses

First, it is possible to use a reciprocally-licensed component through a license firewall that limits the scope of reciprocal obligations for specific licenses (depending on how the license provisions are interpreted). Rather than connecting conflicting components directly through static build-time links, the connection is made through a dynamic link, client-server protocol, license shim, or run-time plug-in.

A second approach used by a number of large organizations is to avoid using any components with reciprocal licenses.

Even using design heuristics such as these, keeping track of license rights and obligations across components that are interconnected in complex OAs quickly becomes cumbersome. Organizations wishing to follow a "best-of-breed" component selection policy, without regard to component licenses, face even steeper challenges. Automated support is needed to manage this multi-component, multi-license complexity.

IV. License Rights and Obligations

In our previous work [2] we developed an approach for expressing software licenses that is more formal and less ambiguous than natural language, and that allows us to calculate rights and obligations for an HtL system and identify conflicts arising from the rights and obligations of two or more component's licenses. Our approach is based on Hohfeld's eight fundamental jural relations [8], of which we use *right* ("may"), *duty* ("must"), *no-right* ("must not"), and *privilege* ("need not") (Figure 2). Each relation has a *correlative* relation, which in our context relates an obligation to its necessary right:

- if actor A must perform action X, then A requires the correlative right to perform it, expressed as "A may X";
- if actor A must not perform action X, then A requires the correlative right to not perform it, "A need not X".

We express rights and obligations as tuples (Figure 3):

<actor, modality, action, object, license>

The actor is either the "Licensee" or in a few cases "Licensor" for all the enactable, testable provisions of the licenses

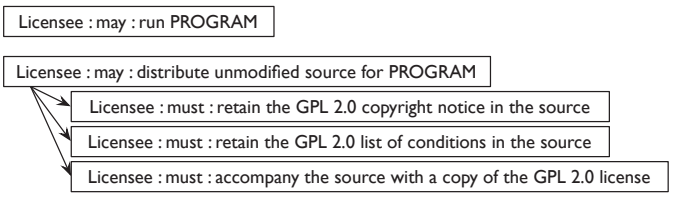


Fig. 4. Some tuples for the GPLv2 license

we have examined [3]. The modality is "may" or "need not" for a right and "must" or "must not" for an obligation. The action is a verb phrase acting on an object, describing what may, need not, must, or must not be done. The object is a module of the system or a related artifact such as a source file, the original version, documentation, and so forth. Typically a license right applies to any of a class of objects distributed under the license, such as any binary file or any modified source file; and the right's obligations will apply to the same object or a related object, such as the right's object's sources or the right's object's originals. For this reason we term rights and obligations as expressed in a license *abstract*, in contrast to a *concrete* right or obligation for one specific entity. Some actions are parameterized by a license as well.

Because copyright rights are exclusive to the copyright holder and licensees, the actions in copyright rights are distinguished from other actions; rights with those actions are only available through the object's license. Rights formed from all other actions are freely and immediately available, unless the object's license obligations restrict them.

A license is expressed as a set of rights, each right associated with zero or more obligations that must be fulfilled be granted it, and possibly a set of overall obligations that must be fulfilled for the license as a whole. Figure 4 sketches two rights from GPL version 2.0 (GPLv2), the first with no obligations and the second with three corresponding obligations.

The details of the license specification approach are described in our earlier work [2], [3].

V. Applying Licenses to Software

A. Calculating the Inference Dag

In order to obtain a particular desired right *r* for a specific module or other entity *e*, in other words a desired *concrete right*, one of two cases must hold:

1) *r* is *not* subsumed by any of the five copyright rights, and does not conflict with any general obligation of *r*'s license *L*. In this case *r* is freely available.
2) *r is* subsumed by an abstract right *R* of the license, with *e* likewise subsumed by *R*'s object. In this case all *R*'s obligations $O_1, O_2, \ldots, O_n$ must be fulfilled, with their objects replaced by whatever function of *e* they signify, in order for *r* to be granted. These could be *e* itself, all sources of *e*, the original version of *e*, and so forth. *n* may be zero, in which case *L* immediately grants *r*.

Figure 5 illustrates one step of the application of a license to obtain a desired concrete right *r*. In the license of *r*'s object

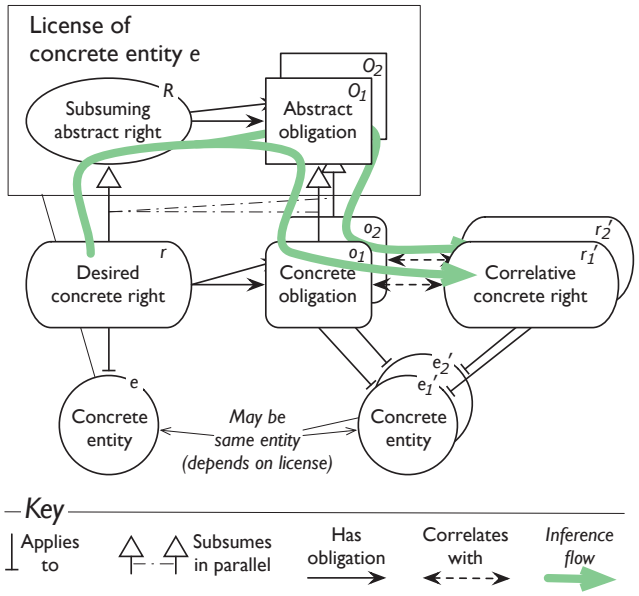


Fig. 5. A step in a rights/obligations inference

e, we search for an abstract right *R* subsuming *r*. The figure shows two obligations O_1 and O_2 of *R*, which we apply to *r*'s object *e* in order to obtain *r*'s concrete obligations o_1 and o_2. Depending on what kind of object O_1 has, o_1 could apply to *e* itself, in which case $e = e'_1$, or to an entity related to *e*, or (if *L* is a propagating license) to another module linked or otherwise connected to *e*. Finally, in order to fulfill o_1 we must have o_1's correlative right r'_1. The same considerations apply for O_2, of course. The heavy arrow shows the flow of inference from desired concrete right through to required concrete obligations and correlative rights.

If r'_1 (r'_2) is immediately available, its branch of the inference is complete. If not, the process recurses from r'_1 (r'_2).

The license rights and obligations for an entire system are calculated by repeating this process for every module of the system. If all modules are under the same license, analogous rights and obligations obtain for every module. If the system is heterogeneously-licensed, however, the calculation is much more varied, and if some of the modules are propagationally licensed then a right for one of those modules can produce obligations for other modules of the system. Such an architecture can easily result in license conflicts, as for example when a license propagates the obligation to be sublicensed under the same license to a proprietary component whose license forbids sublicensing. In such a case, the calculation will fail to produce a simultaneously satisfiable collection of obligations, and no rights will be available for the system as a whole.

Figure 6 shows in Toulmin form a portion of an example inference that produces a conflict, involving a component `e1` obtained under GPLv2 and modified, linked to a component `e2` obtained under the proprietary Corel Transactional License (CTL) [1]. The architectural connection between `e1` and `e2` is one that is interpreted for this inference as propagating

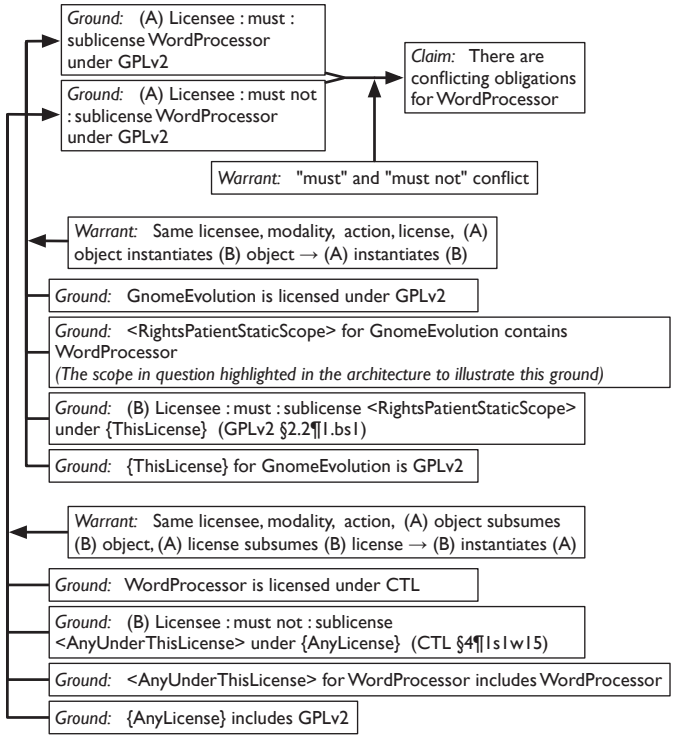


Fig. 6. Toulmin-structured arguments supporting (and explaining) a typical conflict between obligations for a GPLv2 and a proprietary component

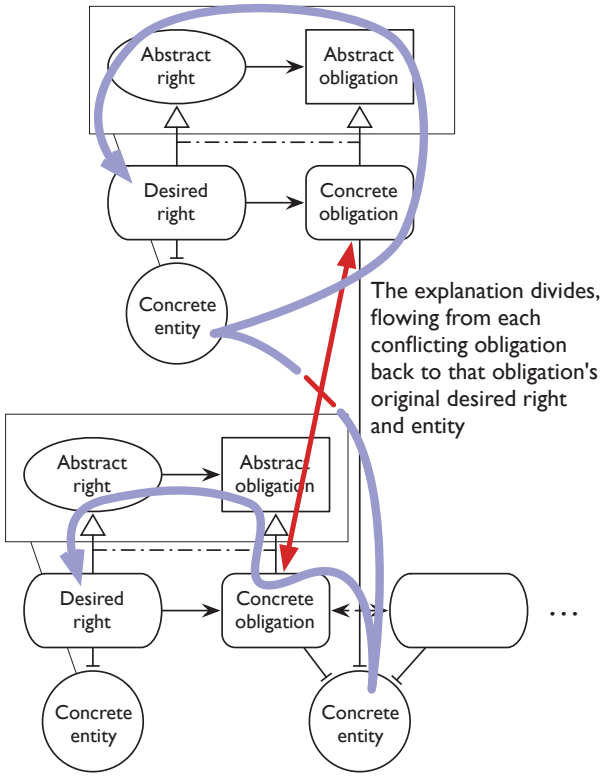


Fig. 7. Divided explanation flow for a conflict between two obligations

GPLv2 obligations, such as a static link. The right to distribute copies of the containing system is desired. In our prototype implementation (Figure 8) these arguments are presented in outline form, with the claim as the root of the outline and its grounds and warrant as its subheads, to be expanded as desired if further explanation is needed. A typical use would be:

1) Why does the WordProcessor component need to be sublicensed under GPLv2?
2) It is in the static-linked scope of the GnomeEvolution component; that component is annotated with the GPLv2 license; and GPLv2 obligates sublicensing under GPLv2 (GPLv2 §2.2¶1.bs1).
3) Why can't the WordProcessor component be sublicensed under GPLv2?
4) The WordProcessor component in the architecture has been annotated with the CTL license, and CTL forbids sublicensing under any license (CTL §4¶1s1w15).

B. Explanation by Argumentation

Figure 7 shows the two explanation flows for a conflict between obligations. Each flow begins at the conflict and explains how one half of the conflicting pair came to be. The connection between the pair is straightforward, as they are identical except for their modalities which are always "must" for one and "must not" for the other.

The flow and the required explanations are analogous for a right-obligation conflict, with the right and obligation again identical except for their modalities, which are always opposites, either "may" and "must not" or "must" and "need not".

After examining the kinds of information that are available in the vicinity of a problem (a conflict or unavailable right), we realized the inferences leading up to it provide the clearest insight into what the problem signifies and why it is present.

- The chains of inference leading up to the problem constitute precisely the portion of the calculation relevant to the problem. No other parts of the calculation—or of the applications of license provisions, determined by the architecture and its annotations, that the calculation identifies—affect whether the problem is present or not.
- The inferences place the problem in the context of licenses, components and their annotations, and architectural configuration — the context in which a designer using the tool is already working.
- Each chain of inference, followed in reverse, provides an unfolding explanation for the problem's presence, which an analyst can explore as far as is helpful in providing understanding and insight.

Each step of a chain of inference is a point at which it can be broken—by replacing a component with one differently licensed, replacing one or more connectors to firewall off a propagating obligation, replacing a build-time component with one provided by users at run time, or other design decisions.

C. Automation

The license metamodel, calculation, and an assortment of license interpretations are implemented in a Java package. The

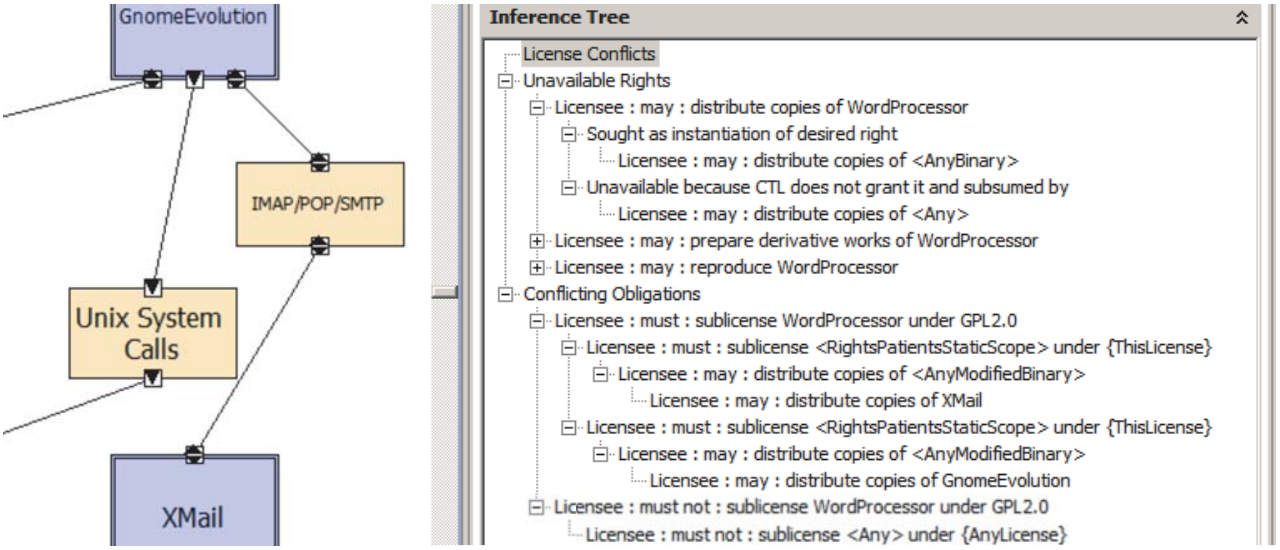


Fig. 8. Prototype explanation results for a CTL-GPL2.0 conflict: (at top) unavailable rights (partially collapsed), (middle) two conflicting obligations.

calculation builds the entire dag, which is then available for presentation in whatever ways are desired. Each abstract right and obligation in a license interpretation has its provenance in the license or interpretation for use in explanations. The package supports the addition and use of new interpretations.

The package is connected into the system design context by its integration into an ArchStudio 4 plugin [4]. The plugin maps features of software architectures onto the license architecture abstraction needed for the virtual license calculation and displays results in the context of the architecture.

The argument grounds drawn from the texts of licenses are implemented through URLs hyperlinking into our collection of software licenses tagged for reference with §-¶-sentence-word numbers [1]. Each URL cites the sentence or phrase from which a right or obligation arises. Word-level `id`s allow references to, for example, #S2.2p1.bs1w11 for the phrase beginning at word 11 of that sentence.

VI. Conclusion

HtL system design and development provide important benefits but impose new demands difficult to meet using only manual methods and human insight. Our approach for supporting HtL development and acquisition automates the calculation of HtL system virtual licenses. We have integrated it into a software architecture tool so it can be applied at the point in the development process when the necessary information is available and the relevant design decisions are made. A key benefit it provides is the automated calculation of license conflicts, desired but unavailable rights, and virtual licenses. But explaining them is of even greater value.

We present a novel approach that presents each conflict in the form of structured arguments showing why each conflict exists and (by implication) points of attack for eliminating it. These arguments provide an informative presentation that brings together all the available information in a compact, evocative form that is easier to interpret, act on, and verify.

Acknowledgments

This research supported by grant #0808783 from the U.S. National Science Foundation, and grant #N00244-10-1-0077 from the Acquisition Research Program at the Naval Postgraduate School. No review, approval, or endorsement is implied.

The authors thank the anonymous reviewers of earlier versions of this paper for their insightful suggestions.

References

[1] T. A. Alspaugh. OSS (and other) licenses, §/¶/sentence/word-numbered. http://www.thomasalspaugh.org/pub/osl-sps/.

[2] T. A. Alspaugh, H. U. Asuncion, and W. Scacchi. Intellectual property rights requirements for heterogeneously-licensed systems. In *17th Int. Requirements Engineering Conference (RE'09)*, pages 24–33, 2009.

[3] T. A. Alspaugh, W. Scacchi, and H. U. Asuncion. Software licenses in context: The challenge of heterogeneously-licensed systems. *Journal of the Association for Information Systems*, 11(11):730–755, Nov. 2010.

[4] E. Dashofy, H. Asuncion, S. Hendrickson, et al. Archstudio 4: An architecture-based meta-modeling environment. In *28th Int. Conference on Software Engineering, Companion Volume*, pages 67–68, 2007.

[5] L. Determann. Dangerous liasons—software combinations as derivative works? *Berkeley Technology Law Journal*, 21(4), 2006.

[6] K. Feldt. *Programming Firefox: Building Rich Internet Applications with XUL*. O'Reilly Media, Inc., 2007.

[7] C. B. Haley, R. Laney, J. D. Moffett, and B. Nuseibeh. Security requirements engineering: A framework for representation and analysis. *IEEE Transactions on Software Engineering*, 34(1):133–153, 2008.

[8] W. N. Hohfeld. Some fundamental legal conceptions as applied in judicial reasoning. *Yale Law Journal*, 23(1):16–59, 1913.

[9] L. Nelson and E. F. Churchill. Repurposing: Techniques for reuse and integration of interactive systems. In *International Conference on Information Reuse and Integration (IRI-08)*, page 490, 2006.

[10] P. Oreizy. *Open Architecture Software: A Flexible Approach to Decentralized Software Evolution*. PhD thesis, Univ. of Calif., Irvine, 2000.

[11] L. Rosen. *Open Source Licensing: Software Freedom and Intellectual Property Law*. Prentice Hall, 2005.

[12] M. J. Sergot, F. Sadri, et al. The British Nationality Act as a logic program. *Communications of the ACM*, 29(5):370–386, May 1986.

[13] M. L. Stoltz. The penguin paradox: How the scope of derivative works in copyright affects the effectiveness of the GNU GPL. *Boston University Law Review*, 85(5):1439–1477, 2005.

[14] S. Toulmin, R. Rieke, and A. Janik. *An introduction to reasoning*. Macmillan, 1984.

A Genetic Approach for Software Architecture Recovery from Object-Oriented Code

Abdelhak-Djamel Seriai
LIRMM, University of Montpellier 2/CNRS
162 rue Ada
F-34392 Montpellier Cedex 5, France
seriai@lirmm.fr

Sylvain Chardigny
MGPS
Port-Saint-Louis,
France
chardigny.sylvain@gmail.com

***Abstract*— Software architecture is recognized as a critical element in the successful development and evolution of software-intensive systems. Despite the important role of architecture representation and modeling many existing systems like legacy or eroded ones do not have a reliable architecture representation. In this paper we present an approach for architecture recovery from object-oriented code. It's based on a genetic algorithm which uses a fitness function measuring the semantic-correctness of software components. Following our model, architecture which is a partition of classes is considered as a chromosome. A group of classes is a gene. This algorithm gives satisfactory results in terms of consistency and adequacy metrics.**

Keywords- Component; software architecture; recovery; component based; object oriented; reverse engineering

I. INTRODUCTION

Software architecture is recognized as a critical element in the successful development and evolution of software-intensive systems [5]. Software architecture expresses the overall structure of a system in an abstract, structured manner. The main goal of a software architectural representation of a system is to identify the major components that constitute this system, and the interactions between these components [1]. According to Garlan [6], software architecture plays an important role in at least six aspects of software development: understanding, reuse, construction, evolution, analysis and management. Despite the important role of architecture representation and modeling, many existing systems do not have a reliable architecture representation. Indeed these systems could have been designed without an architecture design phase, as it is the case of most legacy systems. For other systems, the available representation can diverge from the system implementation. This appears, first, during the implementation phase due to gaps between the expected architecture and the implemented one. These gaps become greater because of lack of synchronization between software documentation and implementation. Taking into account the previous considerations, it is obvious that an approach of architecture recovery allows architects and developers to take advantage of all the benefits of having an architecture model available. In this context, we propose an approach to extract a component-based architecture from object-oriented systems. Our goal is to decrease the need for human expertise which is expensive and not always available. Our process aims at selecting among all the architectures which can be abstracted from a system, the best one according to the semantic-correctness of architecture. Based on the norm ISO 9126 [10], we formulate these characteristics as measurable properties and specify the recovery process as a balancing problem of these ones. Based on this formulation, we developed a recovery architecture process exploiting a hierarchical clustering algorithm. Nevertheless, the result of this process (i.e. architectures) was not completely satisfactory. We studied the process on several case studies (e.g. Jigsaw, ArgoUML, Eclipse, etc.). The results were sometimes offset from the known architectures. This is due to the nature of the clustering algorithm. In fact, this heuristic algorithm is memory less. For some systems, this feature may deter the clustering process from the best architectures. It explores only a limited number of possible ones. It is not a meta-heuristic and it does not use the space exploration of all possible architectures. Thus, our objective in this paper is to propose an alternative formulation of our approach based on a genetic algorithm (GA). This choice is due to the characteristics of this type of algorithm. A GA allows us to consider architecture recovery as a metaheuristic optimization problem. It aims to explore the solution space to identify the best possible. GAs were introduced in the late 1960's by John Holland [7]. They are based on the Darwinian theory of evolution whereby species compete to survive and the fittest get a higher chance to remain until the end and produce progeny. The basic idea of a GA is to start from a set of initial solutions, and to use biologically inspired evolution mechanisms to derive new and possibly better solutions.

The remainder of this paper is structured as follows. Section 2 presents principle of our architecture recovery approach: semantic-correctness driven. The GA encoding of the recovery process is presented in section 3. Case studies are presented in section 4. They present the results of our approach using GA and compare these results to the clustering-based ones. Section 5 discusses related work. Conclusion and future works are given in section 6.

II. SEMANTIC-CORRECTNESS DRIVEN ARCHITECTURE RECOVERY : AN OVERVIEW

In our approach, recovering a component-based architecture of an object oriented system consists of using its implementation code in order to identify the architectural elements. As first step toward this goal, we defined a mapping model of object

oriented concepts (i.e. classes, methods, interfaces, packages, etc.) and architectural ones (i.e. components, connectors, interfaces, etc.). It defines architecture as a partition of the system classes. Each element of this partition represents a component. These elements are named "shape" and include classes which can belong to different object-oriented packages. A shape is composed of two sets of classes: the "shape interface" includes classes linked with others from the outside of the shape, e.g. a method call to the outside; and the "center" composed of the remainder classes of the shape. We assimilate component to shape and component interfaces to "shape interface". Connectors are all links existing between components. Consequently, the architecture configuration is the set of shapes constituting a partition of the system classes. As a result of these considerations, the search-space of architecture recovery problem is composed of all architectures which are partitions of the system classes. This means that, in a system which contains *n* classes, the search-space contains *O (n!)* potential architectures.

Thus to select the best architecture compared to its semantic correctness, we propose to define fitness function measuring this characteristic. An architecture is semantically correct if its elements (components, connectors and configuration) are. We limit ourselves here to study component semantic-correctness and define function to measure it. This study is based on the most commonly admitted definitions of software component rather than architectural one. Indeed architectural component constraints are included in the software component ones. In addition these supplementary constraints make easier a migration from an object-oriented system to a component-based one.

A. Semantic characteristics of a software component

Szyperski defines in [17] a component as a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties. In [9] Heinemann and Councill define a component as a software element that conforms to a component model and can be independently deployed and composed without modification according to a composition standard. In [12] Luer makes a distinction between component and deployable component. He defines a component as a software element that (a) encapsulates a reusable implementation of functionality, (b) can be composed without modification, and (c) adheres to a component model. A deployable component is (a) prepackaged, (b) independently distributed, (c) easily installed and uninstalled and (d) self-descriptive.

In combining and refining the common elements of these definitions and others commonly accepted ones [11], we propose the following definition of a component: A component is a software element that (a) can be composed without modification, (b) can be distributed in an autonomous way, (c) encapsulates an implementation of functionality, and (d) adheres to a component model. In our approach, the definition of a component model is the Luer one [12]: a model component is the combination of (a) a component standard that governs how to construct individual components and (b) a composition standard that governs how to organize a set of components into an application and how those components globally communicate and interact with each other. As compared to the definitions of Luer and Heineman and Councill, we intentionally do not include the criterion that a component must adhere on a composition theory and the properties of component self-descriptive, pre-packaged and easy to install and uninstall. These are covered through the criterion that a component must adhere to a component model and does not need to be repeated. In conclusion, according to our software component definition, we identify three semantic characteristics of software components: composability, autonomy and specificity. The specificity of a component means that it must contain a limited number of functionalities.

B. Refinement model of the semantic correctness

In the previous section, we have identified three semantic characteristics that we propose to evaluate. To do so, we adapt the characteristic refinement model given by the norm ISO-9126 [ISO]. According to this model, we can measure the characteristic semantic correctness by refining it in the previous three semantic characteristics which are consequently considered as sub-characteristics.

1) From characteristic to properties

Based on the study of the semantic sub-characteristics, we refine them into a set of component measurable properties. Thus, a component is autonomous if it has no required interface. Consequently, the property number of required interfaces should give us a good measure of the component autonomy. Then, a component can be composed by means of its provided and required interfaces. However, a component will be more easily composed with another if services, in each interface, are cohesive. Thus, the property average of service cohesion by component interface should be a correct measure of the component composability. Finally, the evaluation of the number of functionalities is based on the following statements. Firstly a component which provides many interfaces may provide various functionalities. Indeed each interface can offer different services. Thus the higher the number of interfaces is, the higher the number of functionalities can be. Secondly if interfaces (resp. services in each interface) are cohesive (i.e. share resources), they probably offer closely related functionalities. Thirdly if the code of the component is closely coupled (resp. cohesive), the different parts of the component code use each other (resp. common resources). Consequently, they probably work together in order to offer a small number of functionalities. From these statements, we refine the specificity sub characteristic to the following properties: number of provided interfaces, average of service cohesion by component interface, component interface cohesion and component cohesion and coupling.

2) From properties to metrics

According to our object-component/architecture model, component interfaces are assimilated to shape interface. Therefore, the *average of the interface-class cohesion* gives a correct measure of the *average of service cohesion by component interface*. Secondly the *component interface cohesion*, the *internal component cohesion* and the *internal*

component coupling can respectively be measured by the properties *interface class cohesion*, *shape class cohesion* and *shape class coupling*. Thirdly in order to link the *number of provided interfaces* property to a shape property, we associate a component provided interface to each shape-interface class having public methods. Thanks to this choice, we can measure the number of provided interfaces using the *number of shape interface classes having public methods*. Finally, the *number of required interfaces* can be evaluated by using coupling between the component and the outside. This coupling is linked to shape external coupling. Consequently, we can measure this property using the property *shape external coupling*. In order to measure these properties, we need to define metrics. The properties *shape class coupling* and *shape external coupling* require a coupling measurement. We define the metric *Coupl(E)* which measures the coupling of a shape *E* and *CouplExt(E)* which measures the coupling of *E* with the rest of classes. They measure three types of dependencies between objects: method calls, use of attributes and parameters of another class. Moreover they are percentages and are related through the equation: $CouplExt(E) = 100 - Coupl(E)$. Due to space limitations, we do not detail these metrics. Shape properties *average of interface-class cohesion*, *interface-class cohesion*, and *shape-class cohesion* require a cohesion measurement. The metric "Loose Class Cohesion" (LCC), proposed by Bieman and Kang [2], measures the percentage of pair of methods which are directly or indirectly connected. Two methods are connected if they use directly or indirectly a common attribute. Two methods are indirectly connected if a connected method chain connects them. This metric satisfies all our needs for the cohesion measurement: it reflects all sharing relations, *i.e.* sharing attributes in object oriented system, and it is a percentage. Consequently, we use this metric to compute the cohesion for these properties. The refinement model is summarized in Fig.1.

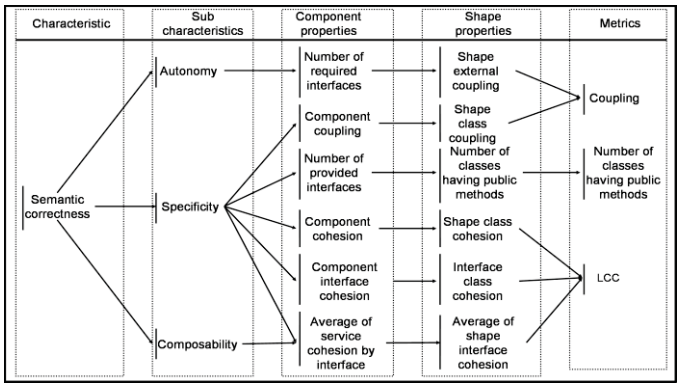


Fig.1. Refinement model of the semantic-correctness characteristic of component

3) Evaluation of the semantic correctness

According to our refinement model of semantic-correctness of component, we define the functions *Spe*, *A*, *C* which measure respectively specificity, autonomy and composability of this component. In these functions *nbPub(I)* is the number of interface classes having a public method and IiI is the shape interface cardinality.

$$Spe(E) = \frac{1}{5} \cdot \left(\frac{1}{|I|} \cdot \sum_{i \in I} LCC(i) + LCC(I) + LCC(E) + Coupl(E) + nbPub(I) \right)$$

$$A(E) = couplExt(E) = 100 - Coupl(E)$$

$$C(E) = \frac{1}{|I|} \sum_{i \in I} LCC(i)$$

The evaluation of the *semantic correctness* characteristic is based on the evaluation of each sub-characteristic. That is why we define this function as a linear combination of each sub-characteristic evaluation function (*Spe, A*, and *C*):

$$S(E) = \frac{1}{\sum_i LCC(i)\mu_i} \left(\mu_1 Spe(E) + \mu_2 C(E) + \mu_3 A(\right.$$

This form is linear because each of its parts must be considered uniformly. The weight μ_i associated with each function allows the software architect to modify, as needed, the importance of each sub-characteristic.

III. GENETIC MODEL FOR ARCHITECTURE RECOVERED FROM OBJECTI-ORIENTED CODE

GA starts derivations from an initial solution called the initial population and then generates a sequence of populations. Each derived population is obtained by "mutating" the previous one. Elements of the obtained solutions are called chromosomes. The fitness of each chromosome is measured by an objective function called the fitness function. Each chromosome (possible solution) consists of a set of genes. At each generation, the process consists to apply some genetic operators which are crossover, mutation and selection in order to generate the next generation. On each chromosome, the algorithm applies two operators: crossover and mutation. Each operator is applied following a specific probability given as an input parameter of the algorithm. During crossover, two chromosomes are selected using a selection method that gives priority to the fittest ones; they exchange some of their genes giving birth to two other chromosomes. Each selected pair of chromosomes produces a new pair of chromosomes that constitute the next generation. Mutation consists of changing randomly one or more genes in a chromosome. Finally a selection is operated on chromosomes to choose the next generation. This selection can increase or reduce or keep stable the size of the population. The algorithm stops if a convergence criterion is satisfied or if a fixed number of generations is reached. The implementation of GA to recover architecture requires specifying how solutions are encoded into chromosomes, how the three genetic operators crossover, mutation and selection are defined and which fitness function and initial population to be used. As we have already defined the function in the section II.B.3, we address the remaining questions in the following sections.

A. Encoding architecture as chromosome

Our genetic model of architecture must adhere to our object-component/architecture mapping model indicating that

an architecture is a partition of classes (cf. section II). This can be translated following different possible formulations. One of these formulations is to represent architecture as a chromosome. Thus, in this model a chromosome is a partition of classes. Another formulation is to model shape as chromosome and architecture as the whole population. This choice makes difficult to check partition property. Instead of the basic idea of GA which aims to optimize one element in a population, this model aims to optimize one population. We opt for representing architecture as a chromosome. Therefore this requires defining the genes which constitute the chromosomes. Again several options are available. Thus a gene may represent a class and the value of the gene may represent the shape which contains this class. An alternative formulation would be to represent a shape as a gene and then the value of the gene would be a set of classes. We opt for the second formulation because it makes easier to check the partition property. Indeed the union of the gene values must be all the system classes and each intersection of gene values must be empty.

B. *Definition of the genetic operators*

In order to apply GA we need to define the genetic operators. These are the selection, the crossover and the mutation operators. The process of evolution starts by selecting several pair of chromosomes whose the number vary according to a probability *PC*. Then the crossover is applied on each pair to generate two new chromosomes. The mutation is applied to each chromosome (new and old) with the probability *PM*. *PC* and *PM* are given as given as parameters of GA. Finally some chromosomes are selected for the next generation. We present in the following each of these three operators.

1) *Selection operator*

There are two selection operators which are used in GA. The first one selects the pair of chromosomes for the crossover and the second selects the next generation among the chromosomes. The selection of the chromosome pair is done according to the roulette-wheel technique [7]. Each chromosome is assigned a portion of the wheel that is proportional to its fitness. A marble is thrown and the chromosome where the marble halts is selected. The selection of the chromosomes for the next generation is done according to two criteria: the age of the chromosomes and their fitness. The new chromosomes are automatically added to the next generation. As we decided to keeps the population size constant, the other chromosomes are selected among the old chromosomes according to the fitness function.

2) *Crossover operator*

A standard way to perform the crossover operation on chromosomes is to cut each of the two parent chromosomes into two subsets of genes (shapes in our case). Two new chromosomes are created by interleaving the subsets. If we apply such operation, it is possible that the resulting chromosomes can no longer represent well-defined partitions. Two specific problems can occur. If the intersection of two shapes is not empty, then the solution is inconsistent. The second problem is when the solution is incomplete. This occurs when the union of all the shapes does not contain all the system classes. In both cases, the architecture represented by the chromosome is not even a partition. Figure 2(a) illustrates these two situations. To preserve the consistency and the completeness of the offspring, we propose a crossover operator based on the operator defined for grouping problems [4]. To obtain an offspring, we select a random subset of shapes from one parent and add it to the set of shapes of the second parent. By keeping all the shapes of one of the parents, completeness of the offspring is automatically ensured. To guarantee consistency, we eliminate from the older shapes, the classes contained in the added shapes. Figure 2(b) illustrates the new crossover operator.

3) *Mutation operator*

Mutation is a random change in the genes that happens with a small probability. In the case of our architecture recovery model, the mutation operator randomly moves some classes from one shape to another one. This mutation operator keeps the partition property safe.

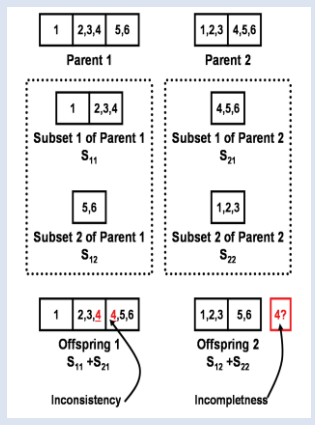


Fig.2 (a) insufficient of the standard crossover

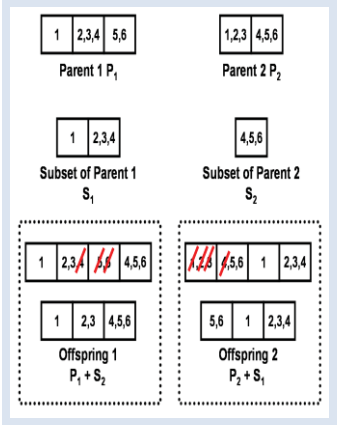


Fig.2 (b) Crossover that preserves consistency and completeness

C. *Choice of the initial population*

The initial population is the set of chromosomes which is used at the start of the GA. The choice of this population is often randomly made. Nevertheless better the initial population is better the solutions is. We opt for an initial population which represents partition obtained by the strongly-connected components of the system classes. In this graph whose vertices are the system classes, there is an arc between two vertices A and B if the class A uses the class B, i.e. uses an attribute, a parameter or a return variable whose type is B and creates or uses an object whose type can be B. A Strongly-connected component is a subset of the graph vertices where any vertex can be reached from any other vertex by a path. These strongly-connected components are a partition of the graph vertices and consequently a partition of the system classes. This partition is an approximation of the system architecture. We use it as an initial chromosome for our GA.

IV. CASE STUDY

As case studies, we validate the genetic algorithm implementation of our architecture recovery process on many systems with different sizes: small systems whose number of classes is less than 50 (e.g. JPhotoAlbum with 17 classes),

systems of medium size where the number of classes is between 50 and 500 (e.g. Jigsaw with 300 classes) and larger systems with more than 1,000 classes. (e.g. ArgoUML with over than 1500 classes). In most cases, the results show that the recovered architectures are closer to the known architectures than those obtained by the clustering algorithm. The difficulty was to choose the adequate parameters for GA. Due to space limitation, we give below only the case study of the Jigsaw system which is a Java based web server.

A. GA parameters

To execute the GA we have to determine some parameters. These are the elements of the initial population, the rate of crossover and mutation, the size of the initial population, and the number of generation. We launch several tests in order to analyze the impact of each parameter on the result. Firstly, we choose to use an initial population based on randomize partitions plus one partition calculated from the strongly-connected components of the system classes. Secondly, the tests realized show that the elements of the population become similar all along the process. This is due to our crossover operator. To avoid having only one element in the population after some generation, we choose a great rate of mutation. Indeed we choose to put the mutation and the crossover rate to 80 %. Nevertheless, this choice of rate is not enough to palliate totally the activity of the crossover operator. Consequently we choose a big initial population (i.e.100) which reduces the risk to obtain a generation composed of only one duplicated element. In order to keep a correct execution time for the test, we choose to do 100 generations.

B. Results

Fig.3 presents the recovered and the known architectures of Jigsaw. The comparison of these architectures shows that most of components of the recovered architecture are the same or sub-components of the known ones. Therefore the obtained solution is relevant according to the known architecture of Jigsaw. The recovered architecture has fitness function score of 80.2 %. This shows the correlation between our fitness function and the relevance of the recovered architecture solution compared to the expected one.

We validate the consistency of our approach by measuring the similarity of the recovered architectures of Jigsaw. We use the similarity measure proposed by Mitchell [18]. For each link between two classes it measures the number of solutions (architectures) for which this link is included in a component. To obtain the percentage of inclusion in a component, this value is then divided by the number of compared solutions. The different results are then aggregated at the levels: {[0, 0], (0, 10], (10, 75), [75, 100] }. These levels correspond to a similarity degree zero, low, medium and high.

Table 1 shows degree of similarity obtained over 99 executions of the recovery process. It shows that 81% of classes are in the zero or high categories. This demonstrates that the vast majority of classes are respectively always separated or together. Only 6.4% of class relationships are in the category medium. This shows that class neighborhoods are stable. Changes are due to classes that are on the borders of two components: the content of these classes is highly dependent on two distinct components. Therefore a significant portion of these class methods can be specified as connectors.

Zero (%) $S = 0\%$	Low (%) $0\% < S <= 10\%$	Medium (%) $10\% < S < 75\%$	High (%) $S <= 75\%$
22.9	12.6	6.4	58.1

TAB 1. A measure of degree of similarity of obtained solution on Jigsaw

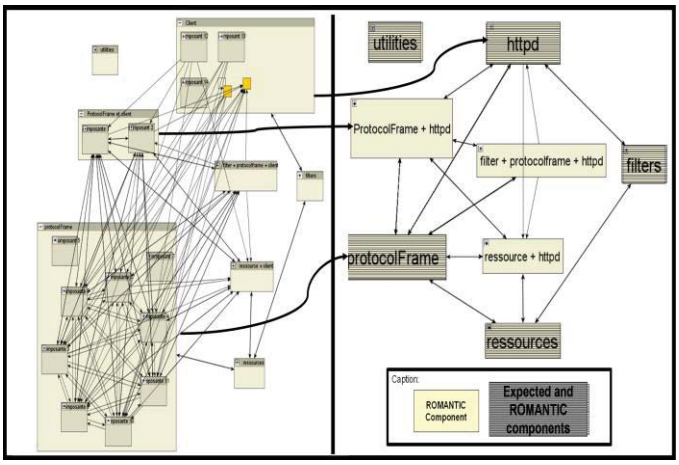


Fig.3 Recovered and known architectures of Jigsaw system

The result of clustering and genetic algorithms is significantly different. This difference is due to the way how the process explores the solution space. On the one hand, clustering explores one solution per iteration. On the other hand, GA explores several solutions (100 in our test) per iteration and the number of genetic operations done by iteration is limited to 2 operations by element of the population. It is clear that GA explores a bigger space than clustering algorithm. GA has a better ratio between the execution time and the quality of the resulting solution.

V. RELATED WORK

Various works are proposed in literature in order to recover architecture from an object-oriented system [15]. We distinguish these works according to two criteria: the input and the technique. Firstly the inputs of the recovery approaches are various. Most often it works from source code representations, but it also considers other kinds of information which for most of them are non-architectural. We can cite, for example, human expertise, which is used in an interactive way in order to guide the process [13], and physical organization, e.g. files, folders and packages [8]. Some works use architectural input. Medvidovic [13] uses styles in Focus in order to infer a conceptual architecture. Finally most works are based on the human expertise: some use the expertise of the architect which uses the tools as an input whereas others use the expertise of the one which proposed this approach. In our approach we use architectural semantic in order to reduce this need of human expertise. Secondly the techniques used to recover architecture

are various and can be classified according to their automation level. Firstly some approaches are quasi manual. For example, Focus [13] proposes a guideline to a hybrid process which regroups classes and maps the extracted entities to conceptual architecture obtained from an architectural style according to the human expertise. Secondly most approaches propose semi-automatic techniques. It automates repetitive aspects of the recovery process but the reverse engineer steers the iterative refinement or abstraction, leading to the identification of architectural elements. Thus ManSART [8] tries to match source code elements on the architectural styles and patterns defined by reverse engineers. Our approach is quasi-automatic too. The main difference with other quasi-automatic approaches is that it refines the commonly used definitions of components into semantic characteristics and refinement models whereas others works use the expertise of the authors in order to define rules driving the process. Some works aim to find the best grouping of elements to subsystems, i.e., the best clusters of an existing software system. Some of these works use a genetic algorithm to compute the best partition [3, 14]. For example, in [14] the problem representation uses chromosomes where each gene represents a class and contains the number of the corresponding cluster. Among these approaches of software clustering, the work of Mancoridis and Mitchell [3] is close to our approach. They introduced the concept of software modularization as a clustering problem for which search is applicable. Their tool Bunch uses a variety of search algorithms. Result is a graph of dependence between modules. This is not an architectural view of the system.

VI. CONCLUSION

We presented in this paper an approach of architecture recovery of object-oriented systems. Architecture recovery is formulated as a search-based problem based on a genetic algorithm (GA). To use genetic algorithms, we adapted our object-component/architecture model to manipulate the architectures (solutions) as chromosomes and group of classes as genes. The properties of this algorithm make it particularly efficient in cases where the computation time is less important compared to the quality of the result. Case studies show that fitness function score is proportional to the relevance of the obtained architectures compared to the expected ones.

As a perspective of this work, we intend to define a method to combine, for a given system, the results obtained by using the genetic implementation of our architecture recovery process with those obtained by the use of the implementation based on simulated annealing [16]. Our goal is to get more relevant architectures in all use cases.

REFERENCES

[1] Kaz1 : Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice,Addison-Wesley, 1998, ISBN 0-201-19930-0.J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73

[2] J. M. Bieman and B.-K. Kang, “Cohesion and reuse in an object-oriented system,” in Proc. of the Symp. On Software reusability,SSR ’95, pp. 259–262, 1995.

[3] D. DOVAL, S. MANCORIDIS et B. S. MITCHELL. Automatic clustering of software systems using a genetic algorithm. In STEP ’99 : Proceedings of the Software Technology and Engineering Practice, page 73, Washington, DC, USA, 1999. IEEE Computer Society.

[4] Emanuel FALKENAUER. Genetic Algorithms and Grouping Problems. John Wiley & Sons, Inc., New York, NY, USA, 1998.

[5] Garlan, D., Shaw, M.: "An Introduction to Software Architecture," In V. Ambriola and G. Tortora (ed.), Advances in Software Engineering and Knowledge Engineering, Series on Software Engineering and Knowledge Engineering, Vol 2, World Scientific Publishing Company, Singapore, pp. 1-39, 1993.

[6] Garlan. Software architecture: a roadmap. In ICSE – Future of SE Track, pp. 91–101, 2000.

[7] Holland, J. Adaptation in Natural and Artificial Systems, Ann Arbor, MI: The University of Michigan Press, 1975.

[8] D. R. Harris, H. B. Reubenstein, and A. S. Yeh, “Reverse engineering to the architectural level,” in Proc. of ICSE, pp. 186–195, ACM, Inc., 1995.

[9] G. Heinemann and W. Councill, Component-based software engineering. Addison-Wesley, 2001.

[10] ISO/IEC-9126-1 in Software engineering - Product quality - Part 1: Quality Model, ISO-IEC, 2001.

[11] I. Jacobson, M. Griss, and P. Jonsson, Software Reuse. Addison Wesley/ACM Press, 1997.GP

[12] C. Luer and A. van der Hoek, “Composition environments for deployable software components,” tech. rep., 2002.

[13] N. Medvidovic and V. Jakobac, “Using software evolution to focus architectural recovery,” Automated Software Engineering, vol. 13, pp. 225–256, 2006.

[14] Olaf SENG, Markus BAUER, Matthias BIEHL et Gert PACHE. Search-based improvement of subsystem decompositions. In GECCO ’05 : Proceedings of the 2005 conference on Genetic and evolutionary computation, pages 1045–1051, New York, NY, USA, 2005. ACM.

[15] D. Pollet, S. Ducasse, L. Poyet, I. Alloui, S. Cimpan, and H. Verjus, “Towards a process-oriented software architecture reconstruction taxonomy,” in Proc. of the CSMR, pp. 137–148, 2007.

[16] Sylvain Chardigny, Abdelhak Seriai, Mourad Oussalah, Dalila Tamzalit: Search-Based Extraction of Component-Based Architecture from Object-Oriented Systems. ECSA 2008: 322-325

[17] C. Szyperski, Component Software. ISBN: 0-201-17888-5, Addison-Wesley, 1998.

[18] Brian S. Mitchell et Spiros Mancoridis. On the evaluation of the bunch search-based software modularization algorithm. Soft Comput., 12(1) :77–93, 2008.

An Ontology based Method for Building Understandable Hierarchical Classification Structure for Software Assets Browsing

Ge Li, Zhi Jin
Software Institute, EECS, Peking University, P.R.China
Key Laboratory of High Confidence Software Technologies,
Ministry of Education of P.R.China
{lige, zhijin}@sei.pku.edu.cn

***Abstract*—Software asset management (SAM) represents an important role for software development and maintenance. Many software companies have been using a SAM system to help to control costs and optimize software investments across their organization and throughout all stages of their life cycles. During all the process in SAM, how to classify the software assets reasonably and build a classification to help developers find their desired software asset effectively is an import part. However, how to build a reasonable classification is a dilemmatic problem for SAM system managers. Because it is difficult for them to find a Hierarchical Classification Structure (HCS) with good super-ordinate and sub-ordinate word relationships, at the same time, it is also difficult for them to build a Hierarchical Classification Structure (HCS) using the words selected by themselves. In this paper, we proposed an ontology based HCS modeling method for SAM system managers. With this method, the managers can build an understandable HCS for their users to support the browsing of software assets. We also present a case study to illustrate the availability of our method.**

Keywords-software asset management; ontology; classification; browsing-based restrieval;

I. INTRODUCTION

Software asset management (SAM) represents an important role for software development and maintenance. Administered through an ongoing plan, SAM makes it easier to identify what you have, where it's running, and whether redundancy may exist [1]. Many software companies have been using a SAM system to help to control costs and optimize software investments across their organization and throughout all stages of their life cycles. A well-deployed SAM system can help companies cut costs, improve security and compliance, and anticipate future software needs. SAM system also helps organizations accurately capture the costs and benefits associated with IT projects that enable a competitive advantage.

During all the process in SAM, how to classify the software assets reasonably and build a Hierarchical Classification Structure (HCS) to help the developers to find their desired software asset effectively by browsing–based retrieval is an import part. Because, as we have known, the browsing–based retrieval is more instructive for developers and easier for them to find new assets, more and more developers prefer to find their desired assets by browsing the assets library by following a HCS.

However, how to build a reasonable HCS is a dilemmatic problem for SAM system managers. Because, strictly speaking, in a good HCS, the relationship between each super-ordinate word and sub-ordinate word should be an "is-a-kind-of" relation. However, it is usually very difficult for system managers to find a strict taxonomy with only this kind of relation to be used as a HCS directly. At the same time, it is also very difficult for them to build a HCS from scratch using the words selected by them, because they could not make sure whether the relationship among all the selected words is appropriate to be used in a HCS or not.

In this paper, we proposed an ontology based HCS modeling method for SAM system managers. In our method, an ontology is proposed for HCS building. This ontology confined the kinds of relationship between each super-ordinate word and sub-ordinate word in the HCS. Using the instances defined according to this ontology the system managers can build a HCS with controlled understandability (in another paper [2], we gave a HCS generating algorithm to support building the HCS automatically). For the relationships among the words in HCS are defined apparently, the understandability of the whole HCS can be controlled. Furthermore, it becomes easier for the system managers to make it clear that which part of HCS dragged down the understandability of the whole HCS, and then it becomes easier for them to make some improvements.

The rest of this paper is organized as follows: section Ⅱ presents the motivation of our paper with a real using example; section III discussed the different levels of understandability and the corresponding relationships in HCS; Section IV presented our ontology and the HCS building process; In section V, we presented a case study to illustrate the availability of our method; Section VII draws the conclusion for our work.

II. MOTIVATION EXAMPLE

In SAM systems, the HCS usually help the developers to find their desired software asset by browsing–based retrieval

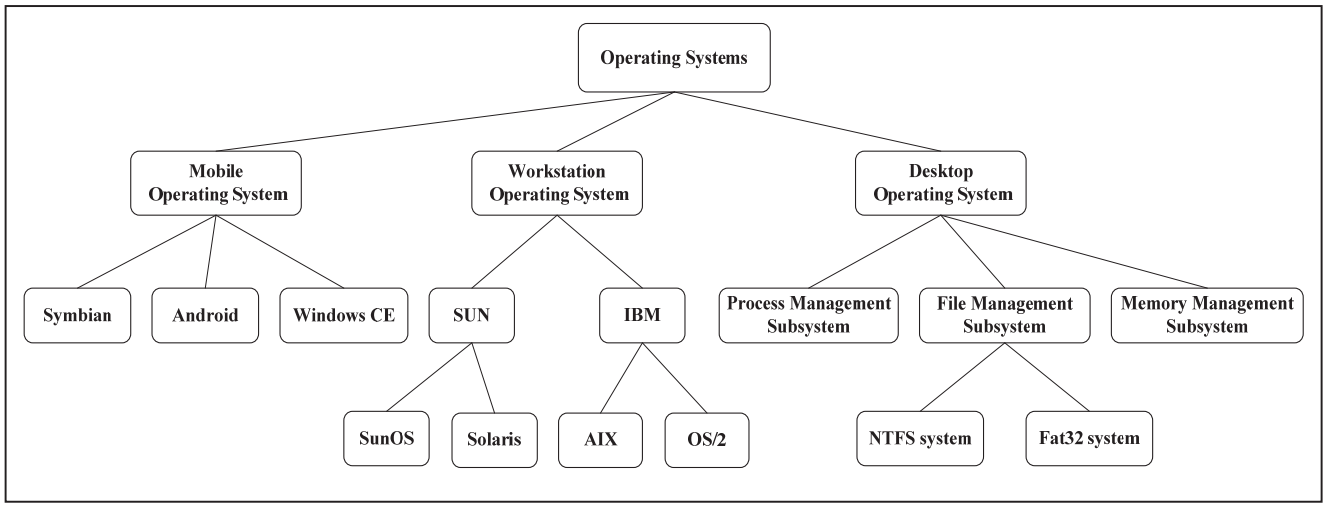


Figure 1. Example of a HCS.

[3][4][5]. HCS is a hierarchical tree structure, in which each node is labeled with a word and represents a set of assets based which related to this word. Navigated by HCS, the browsing retrieval process becomes a process to follow a route in the HCS[6][7]. In a certain browsing-based retrieval step, the developer is usually presented with a screen containing many word-links (e.g. a hyper-link which is labeled by a word in HCS). Each word-link represents a sub-node of a certain node in the HCS. After an overview of the screen, the developer selects a link that he or she thinks the desired assets should be contained, and clicks on the link. This will result in a step of retrieval, in which all the software assets that related to this word are returned as retrieval results. Then the developer can review all the returned assets to check whether there are the desired assets among them or not. If he or she locates all the desired assets, the retrieval process will end; if the returned results are not so confined that the developer cannot find the desired assets, he or she will continue the retrieval process. In the next step, some further links, which represent the sub-nodes, are displayed on the next screen. The developer will keep on the same clicking, retrieving and reviewing until the set of retrieved assets is so confined that the retriever can easily determine whether or not the desired assets as are contained in this retrieved asset set.

From the above analysis, we can see that during the browsing-based retrieval process, developers usually using the HCS to make it clear what assets are there in the SAM system. In each retrieval step, when developer selects a word-link from the screen, he or she always assumes that the word-link could lead him or she to find the desired assets, in other words, the desired assets should be "contained" in the link. For example, for a sample HCS, as Figure 1 shows, that presents a classification for the assets from the "Operating System" aspect [8][9][10]. During the browsing process navigated by this HCS, if a developer selects and clicks the link labeled by "Mobile Operating System", it shows that he or she considers the desired assets are related to the word "Mobile Operating System", or the desired assets should be "contained" in the assets set related to the word "Mobile Operating System". If, in the next step, the developer selects and clicks the link labeled by "Android", it also shows that he or she thinks the desired assets should be contained in the assets set related to the word "Android". So, we can say that, in the corresponding retrieval step, the developer holding an assumption: the assets set related to the word "Mobile Operating System" should include the assets set related to the word "Android". In fact, this assumption is very important. It just because of holding this assumption, the developers could be navigated by the HCS to find their desired assets.

However, what kinds of semantic relationships between each super-ordinate word (e.g. "Mobile Operating System") and sub-ordinate word (e.g. "Android") in the HCS could make the retriever consider "the assets set related to the super-ordinate word should include the assets set related to the word sub-ordinate word? This is a dilemmatic problem for SAM system managers.

Strictly speaking, the "is-a-kind-of" is a good candidate kind of relationship for each super-ordinate and sub-ordinate word pair in the HCS, just like the words defined in some predefined taxonomy ontologies. For example, as Figure 1 shows "Android" is-a-kind-of "Mobile Operating System", so, by these two words developers could be navigated. However, it is really very difficult for SAM system managers to find practicable predefined taxonomy ontology to be used as a HCS. Also, it is very difficult for system managers to build a HCS using only "is-a-kind-of" relationship. For, different from taxonomy ontology, HCS is a tool used to navigate the discovery of the software assets, but not used to present taxonomy knowledge.

At the same time, it is also very difficult for system managers to build a HCS from scratch using the words selected by themselves, because, in the literature, there still no method discuss about what kinds of relationships could make the retriever consider "the assets set related to the super-ordinate word should include the assets set related to the word sub-ordinate word. So, how to modeling the word relationships in HCS and how to build an understandable HCS is remained quite a problem for SAM system.

III. OUR APPROACH

In this paper, we proposed an ontology based HCS modeling method for SAM system managers. An upper level ontology is proposed to confine the kinds of relationship between each super-ordinate and sub-ordinate word pair in the HCS. Using the instances defined according to this ontology, the managers can build an understandable HCS for their users to support the browsing of software assets. In this section, we will discuss the different levels of understandability of HCS, and then present the upper level ontology used to modeling the HCS. From this section on, we will call the developers that are retrieving the SAM system as retrievers.

A. Retriever Assumptions and Relationships Analysis

As we have discussed in section II, during the retrieving process, the retrievers are always holding some assumption: the assets set related to the super-ordinate word should include the assets set related to the sub-ordinate word. And, because of holding this assumption, the developers could be navigated by the HCS to find their desired assets. In fact, besides of this assumption, there are several other assumptions holding by the developers during the retrieval process. According to the degrees of strictness of this assumption, we classified them into 4 different levels, named Retriever Assumption 1-4：

Retriever Assumption I: During the retrieving process, the retrievers are always assuming that the assets set related to the super-ordinate word should include all the assets related to the sub-ordinate word.

In our method, in order to modeling the right semantic that meets Retriever Assumption I. we defined three kinds of relationships for each super-ordinate and sub ordinate word pair:

IsaKindof: for word W_1 and word W_2, if the concept represented by W_1 is a specialization of concept represented by W_2, and the concept represented by W_2 is a generalization of concept represented by W_1, then we have W_1 is a kind of W_2, denoted as: W_1 *IsaKindof* W_2 .

For example, in Figure1, "Android" is a kind of "Mobile Operating System". According to the definition of IsaKindof relationship, we have: *"Android" IsaKindof "Mobile Operating System"*. During the retrieving process, if the retrieval result had been confined to the assets set related to "Mobile Operating System", then the words "Android" means the assets set that related to the "Android Mobile Operating System", this set is included in the set of assets related to the super-ordinate word ("Mobile Operating System"), so we can see, the semantic of *"Android" IsaKindof "Mobile Operating System"* can meet the definition of Retriever Assumption I.

IsaPartof: for word W_1 and word W_2, if any instance of the concept represented by W_1 is a part of some instance of the concept represented by W_2, then we have W_1 is a part of W_2, denoted as: W_1 *IsaPartof* W_2 .

For example, in Figure1, "File Management Subsystem" is a part of "Desktop Operating System". According to the definition of IsaPartof relationship, we have: *"File Management Subsystem" IsaPartof "Desktop Operating System"*. During the retrieving process, if the retrieval result had been confined to the assets set related to "Desktop Operating System", then the words "File Management Subsystem" means the assets set that related to the "File Management Subsystem in Desktop Operating System", this set is included in the assets set related to the super-ordinate word ("Desktop Operating System"), so we can see, the semantic of *"File Management Subsystem" IsaPartof "Desktop Operating System"* can meet the definition of Retriever Assumption I.

[Attribute]Is: for word W_1, attribute A_1 and word W_2, if A_1 is an attribute of W_1 (denoted as: $W_1.A_1$), and if the word W_2 is a kind of $W_1.A_1$, in other words, if W_2 *IsaKindof* $W_1.A_1$ then we have W_1 *[*A_1*]Is* W_2**.**

For example, in Figure1, "Manufacturer" is an attribute of "Workstation Operating System" (although the words "Manufacturer" is not appeared in the HCS), and "IBM" is a kind of "manufacturer" of "Workstation Operating System". According to the definition of [Attribute]Is relationship, we have:*"Workstation Operating System" [Manufacturer]is "IBM"*. Here the word "IBM" means the workstation operating systems that were manufactured by IBM. During the retrieving process, if the retrieval result have been confined to "Workstation Operating System", then the words "IBM" means the assets set that related to the "Workstation Operating System that manufactured by IBM", this set is included in the assets set related to the super-ordinate word ("Workstation Operating System"), so we can see, the semantic of *"Workstation Operating System" [Manufacturer]Is "IBM"* can meet the definition of Retriever Assumption I.

So, the above three kinds of relationship (***IsaKindof, IsaPartof, [Attribute]Is***) give the right semantic that meets Retriever Assumption I. Based on Retriever Assumption I, we defined the Retriever Assumption II and III:

Retriever Assumption II: During the retrieval process in one aspect, the retrievers are always assuming that the desired assets are only belonging to one subset. In other words, in each retrieving step, when selecting the next work-link, the retrievers are always assuming that, the desired assets are only related to only one sub-ordinate word.

If there are two sub-ordinate words, and each of them looks likely to be related to the desired assets, then the retriever will be confused by which should be selected as the next one. For example, in Figure1, if a retriever selected to search desired assets from the "Operating System" aspect, and had selected "Operating System" in the first step, and then faces with three sub-ordinate words ("Mobile Operating System", "Workstation Operating System" and "Desktop Operating System"), and if the assets he or she desired are likely to be included in any one of them or any two of them, the retriever may be confused by which one should be selected.

Therefore, we defined the forth kind of relationship which used to modeling the right semantic that meets Retriever Assumption II:

DisjointWith: for word W_1 and word W_2, if there hasn't any common instance for the concept represented by W_1 and the concept represented by W_2, then we have W_1 disjoint with W_2, denoted as: W_1 *DisjointWith* W_2 . To facilitate representation,

for a words set $W_S = \{W_i, i \in \{1,2,...,n\}\}$, if for any two words W_i and W_j in the set, we have W_i *DisjointWith* W_j, then we can donate the relationship as: *DisjointWith*$\{W_i, i \in \{1,2,...,n\}\}$ or *DisjointWith* W_S.

For example, for the words "Symbian", "Android" and "Windows Mobile" in Figure1, if there haven't any common instance for the concept represented by each of them. According to the above definition, we have: *DisjointWith {"Symbian","Android", "Windows Mobile"}*. Which means the assets sets related to word "Symbian", "Android" and "Windows Mobile" is isolated to each other; there isn't any intersection between each pair of them. In this case, during the retrieving process, if a retriever selected to search from the Operating System aspect, and confined the retrieval result to the assets related to "Mobile Operating System", he or she could make a clear choice of which should be selected. So, we can see that the semantic of *DisjointWith {"Symbian", "Android", "Windows Mobile"}*. can help us to build HCS that meets the definition of Retriever Assumption II.

Retriever Assumption III: During the retrieval process in one aspect (e.g. the Operating System aspect), on each retrieving step, if the current word-link include many sub-ordinate word-links, the retrievers are always assuming that the desired assets must related to at least one of them, but not exist as a isolated leaf.

For example, in Figure1, if a retriever selected to search desired assets from the "Operating System" aspect, and had selected "Desktop Operating System" in the first step, and then faces with three sub-ordinate words ("File Management Subsystem", "Memory Management Subsystem" and "Process Management Subsystem"), the retriever may hold a assumption: all the assets related to "Desktop Operating System" had been divided into three subsets, which related to three sub-ordinate words separately. And, no isolated outside of these three subsets.

Therefore, we defined the fifth kind of relationship which used to modeling the right semantic that meets Retriever Assumption III:

CoveredBy: for word W_1 and a words set $W_S = \{W_i, i \in \{1,2,...,n\}\}$, if any instance of the concept represented by W_1 must belongs to at least one instances set of the concept represented by some W_i in W_S, then we have W_1 is covered by W_S, denoted as: W_1 *CoveredBy* W_S .

For example, in Figure 1, if in the current SAM system, all the assets related to "Desktop Operating System" are divided into three subsets, and each subset related to three sub-ordinate words ("File Management Subsystem", "Memory Management Subsystem" and "Process Management Subsystem") separately, And there isn't any isolated asset outside of these three subsets. Then we have: *"Desktop Operating System" CoveredBy {"File Management Subsystem", "Memory Management Subsystem", "Process Management Subsystem"}*.

Besides the three assumptions mentioned above, we proposed a supplementary retriever assumption for retriever, which could be referenced in HCS building:

Retriever Assumption IV: During the retrieval process in one aspect (e.g. the Operating System aspect), in any consecutive two-step retrieving, the retrievers may assume that the two step's retrieving are navigated by the same kind of relationships.

For example, in Figure 1, if a retriever selected to search desired assets from the "Operating System" aspect, and had selected "Mobile Operating System". For the relationship between "Mobile Operating System" and "Operating System" is IsaKindOf, so the retriever maybe assume that, the relationship in the next step is as same as the relationship in the last step.

We proposed 4 retriever assumptions above, as we can see that, different retriever assumptions could be superimposed together in one HCS. And, the more assumptions the HCS meets, the more understandable it is. So, in order to help the SAM system managers to build more understandable HCS, based on the relationships we mentioned above, we proposed an upper level ontology to guide the HCS modeling.

B. Upper Level Ontology for HCS Modeling

According to the above analysis, we proposed an upper level ontology for HCS modeling.

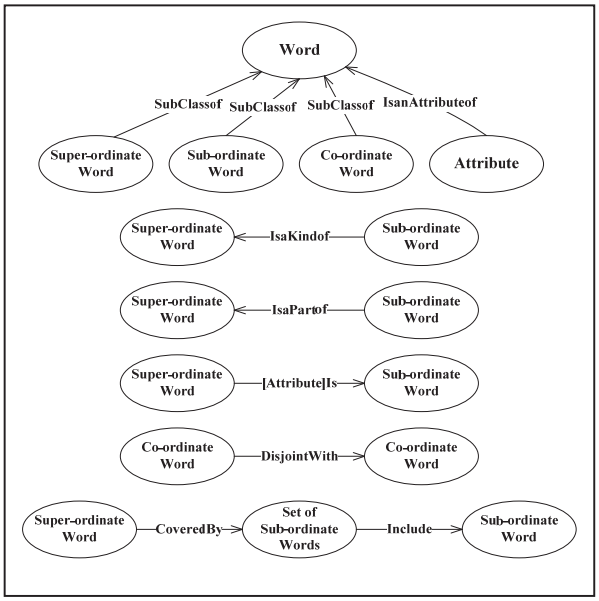


Figure 2. Upper Level Ontology for HCS Modeling.

As Figure 2 shows, the upper level ontology consists of 4 elements: Words, Attributes, Relations and Axioms. In which:

- **Words** represents a set of words that will be used in HCS; there are 3 kinds of words (Super-ordinate word, sub-ordinate word, co-ordinate word).
- **Attributes** is a collection of some small attribute sets, and each small set in the collection includes all the attributes of one word;
- **Relations** represents a set of relationships among different kinds of words, only 5 kinds of relationships (as mentioned in above) can be used in the instance

level ontology to represent the relationships between each words pair ;

Beside the above 5 kinds of relationships, in order to facilitate the representation of attribute, we defined *IsanAttributeOf* as a supplementary relationship :

Wp IsanAttributeOf Wq: to represent that the word Wp is an attribute of the word Wq.

e.g. "Manufacturer" IsanAttributeOf "Workstation Operating System";

- **Axioms** represents a set of axioms; each of them is a constraint on the words or relationships (e.g. about the transferability, symmetry, etc). Each constraint can be expressed in Prolog-like rule [11].

As an example, the following is an instance ontology for the HCS in operating system aspect showed in Figure1:

TABLE I. AN INSTANCE ONTOLOGY FOR OPREATING SYSTEM ASPECT

Operating System = {Words; Attributes; Relations; Axioms}

Words = {
"Operating System", "Mobile Operating System", "Workstation Operating System", "Desktop Operating System", "Symbian", "Android", "Windows Mobile", "SUN", "IBM", "SunOS", "Solaris", "AIX", "Operating System/2", "Process Management Subsystem", "File Management Subsystem", "Memory Management Subsystem", "NTFS system", "Fat32 system"
}

Attributes ={
"Manufacturer" IsanAttributeOf "Workstation Operating System";
}

Relations ={
"Mobile Operating System" IsaKindOf "Operating System";
"Workstation Operating System" IsaKindOf "Operating System";
"Desktop Operating System" IsaKindOf "Operating System";
"Symbian" IsaKindOf "Mobile Operating System";
"Android" IsaKindOf "Mobile Operating System";
'Windows Mobile" IsaKindOf "Mobile Operating System";
"Mobile Operating System" CoveredBy {"Symbian", "Android", "Windows Mobile"};
"DisjointWith {"Symbian", "Android", "Windows Mobile"};
"IBM" IsaKindOf "Manufacturer";
"SUN" IsaKindOf "Manufacturer";
"Workstation Operating System" [Manufacturer]Is "SUN";
"Workstation Operating System [Manufacturer]Is "IBM";
DisjointWith {"SUN", "IBM"};
"SunOS" IsaKindOf "SUN";
"Solaris" IsaKindOf "SUN";
"AIX" IsaKindOf "IBM";
"Operating System/2" IsaKindOf "IBM";
"Process Management Subsystem" IsaPartOf "Desktop Operating System";
"File Management Subsystem" IsaPartOf "Desktop Operating System";
"Memory Management Subsystem" IsaPartOf "Desktop Operating System";
"Desktop Operating System" CoveredBy {"Process Management Subsystem", "File Management Subsystem", "Memory Management Subsystem"};
"NTFS sytem" IsaKindOf "File Management Subsystem";
"Fat32 sytem" IsaKindOf "File Management Subsystem"
}

Axioms = {
$\exists w_p$ *IsaKindof* $w_q \rightarrow \neg\exists w_q$ *IsaKindof* w_p;
$\exists w_p$ *IsaPartof* $w_q \rightarrow \neg\exists w_q$ *IsaPartof* w_p;
$\exists w_p$ *[Attribute]Is* $w_q \rightarrow \neg\exists w_q$ *[Attribute]Is* w_p;
$\exists w_p$ *CoveredBy ws* $\rightarrow \neg\exists$ *ws CoveredBy* w_p;
$\exists w_p$ *DisjointWith* $w_q \leftrightarrow w_q$ *DisjointWith* w_p;
$\exists a_r$ *IsanAttributeof* $w_p \wedge \exists w_q$ *IsaKindof* $w_p \leftrightarrow a_r$ *IsanAttributeof* w_q;
$\exists w_q$ *IsaKindof* $w_p \wedge \exists w_p [w_p.a_r]$ *Is* $w_s \leftrightarrow w_q [w_q.a_r]$ *Is* w_s;
$\exists w_q$ *IsaKindof* $w_p \wedge \exists w_r$ *IsaPartof* $w_p \leftrightarrow w_r$ *IsaPartof* w_q;
$\exists w_q$ *IsaKindof* $w_p \wedge \exists w_r$ *IsaKindof* $w_q \rightarrow w_r$ *IsaKindof* w_p;
$\exists w_q$ *IsaPartof* $w_p \wedge \exists w_r$ *IsaPartof* $w_q \rightarrow w_r$ *IsaPartof* w_p;
$\exists w_p$ *DisjointWith* $w_q \wedge \exists w_r$ *IsaKindof* $w_q \rightarrow w_r$ *DisjointWith* w_p;
$\exists w_p$ *DisjointWith* $w_q \wedge \exists w_r$ *IsaPartof* $w_q \rightarrow w_r$ *DisjointWith* w_p;
}

In practice, this ontology can be represented in any ontology representation language, such as the RDF(S), the DAML+OIL and so on.

Using the proposed upper level ontology can help the SAM system managers to define the relationships among words clearly. As we have discussed, in order to build more understandable HCS, we should try to make every super-ordinate and sub-ordinate word pair meets more retriever assumptions. In the following section we will use a case study to illustrate the using of the proposed ontology.

IV. CASE STUDY

This section uses a case study to illustrate the availability of our ontology. In this case study, we will present how the HCS in the Figure1 are built up step by step, in this process, as we can see that, with more retriever assumption been met, the understandability of the HCS will be improved step by step.

In the first step, we build a HCS that only meets retriever assumption I. During the modeling of this HCS, we only used 3 kinds of relationships: *IsaKindOf, IsaPartOf, [Manufacturer]Is*. We can see that, though the HCS is usable, and meets retriever assumption I, it is still not easy to be understood, for the sub-ordinate words of "Operating System" are so diversiform that confusing the retrievers.

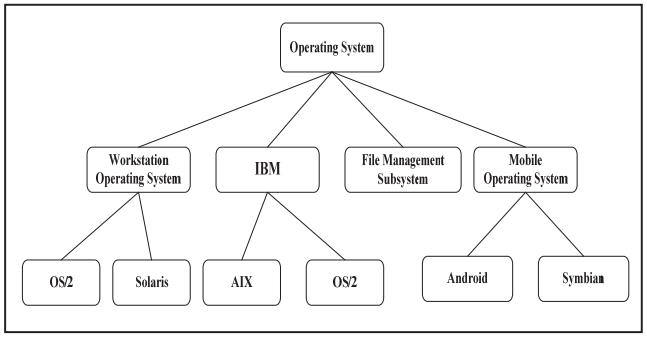


Figure 3. HCS Meet Retriever Assumption I

In the second step, we try to add the relationship of *DisjointWith* to the model of HCS, and try to make the HCS to meet retriever assumption II. As Figure 4 shows, the

understandability is better than Figure 3, for in each retrieving step, the retriever only need to select only one sub-ordinate word-link, but do not need to make trade-offs among several possible word-links, so it becomes more clear and straight for retrievers.

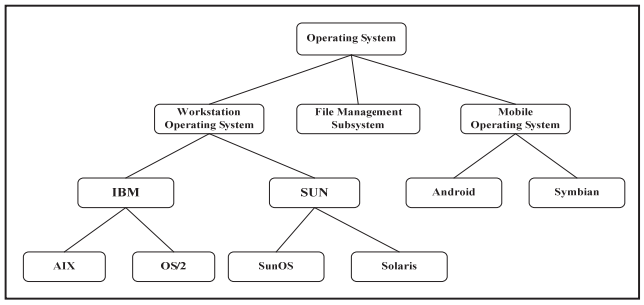


Figure 4. HCS Meet Retriever Assumption II

In the third step, we try to add the relationship of *CoveredBy* to the model of HCS, and try to make the HCS to meet Retriever Assumption III, and the HCS becomes to as Figure 1 shows. We can see that, the understandability is better than Figure 4, for in each retrieving step, the retriever only need to select the next sub-ordinate word-link, but do not need to care about whether the desired asset is an isolated leaf, and isn't related to any sub-ordinate word-link.

However, for the HCS in Figure 1, there is still room for improvement, because it doesn't meet the Retriever Assumption IV. In order to make the HCS to meet Retriever Assumption IV, we could delete the words "IBM" and "Sun" from the HCS, just as Figure 5 shows.

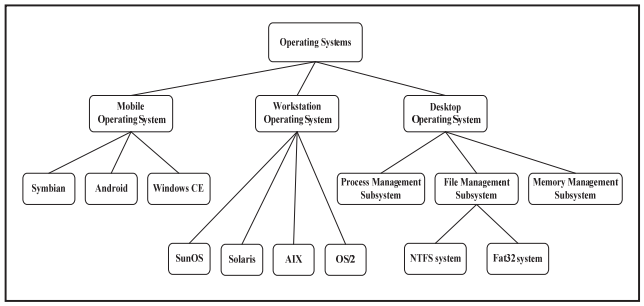


Figure 5. HCS Meet Retriever Assumption II

So, we can see that, different retriever assumptions could be superimposed together, the more assumptions the HCS meets, the more understandable it is. By using more kinds of relationships, we can improve the understandability step by step.

V. DISCUSSION AND CONCLUSION

Software asset management (SAM) represents an important role for software development and maintenance. In SAM, how to classify the software assets reasonably and build a Hierarchical Classification Structure (HCS) and help the developers to find their desired software asset effectively by browsing–based retrieval is an import part. However, how to build a reasonable HCS is a dilemmatic problem for SAM system managers.

In this paper, we analyzed the different level of understandability of HCS，proposed 4 retriever assumptions, and discussed the kinds of relationship between each super-ordinate and sub-ordinate word in the HCS. We proposed an ontology based HCS modeling method for SAM system managers, which an ontology is proposed for HCS building. This ontology confined the kinds of relationship between each super-ordinate and sub-ordinate word in the HCS. Using the instances defined according to this ontology, the system managers can build a HCS with controlled understandability. And, if more kinds of relationships are used in the modeling, more assumptions in the HCS will be met, and more understandable the HCS is. Also, this ontology can be combined with the HCS generating method (we published in 2007[2]), to build a HCS automatically. Besides these, our method make it easier for the system managers to make it clear that which part of HCS dragged down the understandability of the whole HCS, and then it becomes easier for them to make some improvements.

VI. ACKNOWLEDGMENTS

This research was sponsored by the National Grand Fundamental Research 973 Program of China under Grant No. 2011CB302704, and the National Natural Science Foundation of China under Grant No. 60803010.

[1] http://www.microsoft.com/sam/en/us/overview.aspx

[2] Ge Li, Lu Zhang, Bing Xie, Weizhong Shao, Ontology Based Classification Generating Method for Browsing-Based Component Retrieval. 19th International Conference on Software Engineering and Knowledge Engineering (SEKE), Boston, USA, July 9-11, 2007.

[3] Ge Li, Lu Zhang, Yan, Li, Bing Xie and Weizhong Shao, Shortening Retrieval Sequences in Browsing-based Component Retrieval Using Information Entropy, Journal of Systems and Software (JSS), Vol. 79, No. 2, 2006, pp.216-230.

[4] M. Casanova Paez, R. Van Der Straeten, and V. Jonckers, "Supporting evolution in component-based development using component libraries," in 7th European Conference Software Maintenance and Reengineering, Benevento, Italy, 2003, pp. 123–132.

[5] Y. Ye and B. Reeves, "An active and intelligent agent for component location," in Software Symposium 2000 (SS2000). Kanazawa, Japan: Software Engineer Association, 2000, pp. 67–74.

[6] Y. Ye and G. Fischer, "Context-aware browsing of large component repositories," in 16th International Conference of Automated Software Engineering (ASE'01), Coronado Island, CA, 2001, pp. 99–106.

[7] B. Fischer, "Specification-based browsing of software component libraries," Journal of Automated Software Engineer, vol. 7, no. 2, pp.179–200, 2000.

[8] C. G. Drummond, D. Lonescu, and R. Holte, "A learning agent that assists the browsing of software libraries," IEEE Transaction on Software Engineering, vol. 26, no. 12, pp. 1179–1196, 2000.

[9] M. Hertzum and E. Frokjaer, "Browsing and querying in online documentation: A study of user interfaces and the interaction process," ACM Transaction on Computer-Human Interaction, vol. 3, no. 2, pp. 136–161, 1996.

[10] C. Olston and E. H. Chi ScentTrails, "Integrating browsing and searching on the web," ACM Transaction on Computer-Human Interaction, vol. 10, no. 3, pp. 117–197, 2003.

[11] I. Bratko, PROLOG Programming for Artificial Intelligence, third edtion ed. Pearson Education Limited, 2000.

Mapping Non-Functional Requirements to Cloud Applications

David Villegas and S. Masoud Sadjadi
School of Computing and Information Sciences
Florida International University
Miami, Florida
{dvill013, sadjadi}@cs.fiu.edu

***Abstract*—Cloud computing represents a solution for applications with high scalability needs where usage patterns, and therefore resource requirements, may fluctuate based on external circumstances such as exposure or trending. However, in order to take advantage of the cloud's benefits, software engineers need to be able to express the application's needs in quantifiable terms. Additionally, cloud providers have to understand such requirements and offer methods to acquire the necessary infrastructure to fulfill the users' expectations. In this paper, we discuss the design and implementation of an Infrastructure as a Service cloud manager such that non-functional requirements determined during the requirements analysis phase can be mapped to properties for a group of Virtual Appliances running the application. The discussed management system ensures that expected Quality of Service is maintained during execution and can be considered during different development phases.**

I. Introduction

The emergence of cloud computing responds to a increasing trend in web application emergence and utilization. The wide adoption of Internet has resulted in systems that need to accomodate millions of users [1] and provide capabilities that until now were only required by critical, high availability or high throughput software. The practice of Software Engineering provides methodologies to ensure such characteristics are met, but it is necessary to review how they fit in this new paradigm. In this paper, we explore the applicability of traditional processes to the incipient field of cloud computing from the perspective of our research in an Infrastructure as a Service (IaaS) cloud manager.

Internet has resulted in rapid cycles of software development, deployment and consumption by users. The rising number of subscribers, better network connectivity and bandwidth, and the growing connectedness between users have created new dynamics where applications can be rapidly discovered and consumed. However, the benefits produced by these circumstances are hindered when expected requirements are not met. Nowadays, cloud computing is often employed as a solution to this problem. Capabilities such as pay-per-use, scalability or elastic provisioning of resources can help to overcome these new challenges. Nevertheless, application developers need to recognize how to apply Software Engineering methods to the cloud in order to successfully map their needs to fulfill service expectations.

There are two interrelated points that we believe have to be considered to successfully make use of clouds to develop applications that respond to the new demands generated in this field. First, developers must understand which non-functional requirements take renewed importance in cloud applications so that they can be accounted for during the requirements analysis phase. Second, cloud providers need to define better guarantees for their services, so developers can design their systems accordingly. We believe that providing a solution to these problems will result in a more dependable use of clouds to confront the new challenges of this era.

In this paper we consider the concept of *Distributed Ensembles of Virtual Appliances* (DEVAs), introduced in [2], as a model to represent complex systems with Quality of Service (QoS) guarantees. We discuss how a software architecture can be mapped to a DEVA, and how through the use of performance modeling and prediction we can make certain assurances about its behavior in the cloud in order to address its non-functional requirements. We finally present a case study were we demonstrate the feasibility of our approach to model the expected number of requests per second and response time of a web application hosted in the cloud.

II. Background

We define a cloud application as any software that runs on a distributed system that complies with the definition of a *cloud*. Such systems ([3], [4]) possess certain common capabilities such as on-demand provisioning, resource elasticity or pay-per-use billing model. Therefore, cloud applications can be deployed on remote resources with a minimal cost, and scaled dynamically when user demand grows.

We consider three main actors in our scenario: application users, application providers, and cloud providers. In this proposed division, application providers also have the role of cloud users, even though in certain cases it would be possible that application and cloud providers are the same individual or organization. The cloud is usually divided in Software, Platform and Infrastructure as a Service [3] —SaaS, PaaS and IaaS respectively. Application providers are in charge of implementing the SaaS layer, while the PaaS and IaaS layers are supplied by cloud providers.

A DEVA [2] is a group of Virtual Appliances and virtual network devices, where individual and composite policies can be defined for elements. Virtual Appliances [5] are Virtual

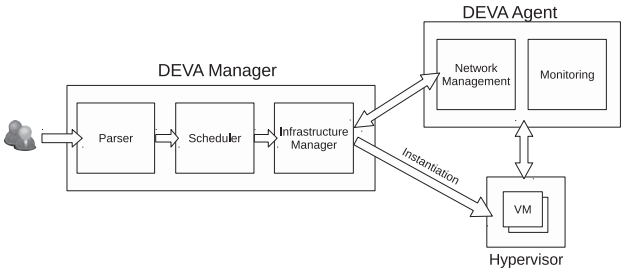


Fig. 1. General architecture

Machines with specific functions, usually containing a particular software and configuration; for simplicity, we'll use the more general term VM to refer to them. Figure 1 illustrates the architecture of the DEVA Manager. A user sends a specification for a list of VMs and their associated QoS requirements, which may consist of CPU, memory and required software for individual VMs, and network bandwidth and latency for the network links. Then, the Manager instantiates the ensemble across heterogeneous resources, which may be located in different administrative domains. A group of agents monitors each VM's behavior and provides the requested QoS and network isolation.

III. Requirement analysis and architectural design

In Software Engineering, the requirements analysis phase is in charge of determining the functional and non-functional requirements of the system based on the client's needs. In particular, non-functional requirements [6] describe the characteristics of the system not related to its functionality. These requirements shape the architecture of the system during the design phase.

In this paper we target a class of applications that are specially suited to be hosted in the cloud and have a prevalent set of non-functional requirements. Identifying them allows developers to ensure that they are addressed during the requirement analysis phase, and establish a set of requisites that must be met by cloud providers in order to quantify their service and ascertain the application goals are met successfully. We enumerate the most salient ones next.

Response time

This requirement describes how much time it takes from the moment a user sends a request to the system, until a complete response is provided. In web applications, this comprehends request transmission and processing, and response transmission. The factors that account for it are resource capabilities —processing power, memory, disk, network latency and bandwidth— and the load produced by other processes running in the server or the number of concurrent requests. For complex requests, this may also involve calls to external systems, or to other subsystems, in which case the host's internal network characteristics and other resources' load may be taken into account.

Uptime

The total time the service is available. It may be expressed as a percentage. When considering this requirement, it is necessary to take into account the provider's own uptime. For example, if a provider has an uptime of 99.5%, it would be impossible to deploy an application with a higher uptime. Other factors involve the recoverability of the system (*i.e.,* how much time it takes to restart the service after a failure happens).

Requests per unit of time

This requirement describes the number of requests the system can handle successfully per unit of time, and can also be referred to as the system's throughput. Resource allocation and usage has an impact in this parameter. Additionally, the number of requests can have an impact in the response time requirement (*i.e.,* a high number of requests will result in a deterioration of the overall response time).

Fault tolerance

One of the system's properties is how it can withstand errors, either hardware or software-based. In the case of cloud, non-software errors can be generated either at the physical or the virtual machines hosting the service. While the first case is usually out of the developer's control, virtual machine faults can be handled by different means, for example by spawning new instances, or having backup VMs to respond to failures.

Security

Security is another requirement that can be applied to the cloud provider or to the developed system. In the first case, the application developer is under the provider's security measures such as physical infrastructure access policies or network isolation mechanisms. Alternatively, security in the instantiated VMs must be handled by the cloud user.

Operational cost

In traditional systems, hardware was determined based on the application's initial requirements. Changes in requirements would typically result in costly upgrades involving the acquisition of new physical machines and installation and configuration of the application to run on them. In cloud systems, resources can be upgraded almost instantaneously, meaning that cost can be considered a changing variable. This allows defining tradeoffs to architectural (static) and operational (dynamic) behavior.

During the requirements analysis, it is the job of the software engineer to give appropriate values to each of the non-functional requirements according to the user's expectations. Each of these parameters needs to be reflected in one or more architectural decisions and tradeoffs.

IV. Mapping Requirements to DEVAs

The original implementation of the DEVA Manager accepts three types of parameters: nodes (VMs and virtual network devices), edges between nodes, and element annotations. Basic annotations describe node and edge characteristics such as VM processor power or memory size, and link bandwidth and latency, respectively.

An application developer could map the assigned non-functional requirement values to any of the discussed DEVA parameters in order to ensure the application's operational guarantees. For example, the number of desired requests per second would influence the assigned latency for the links between VMs; alternatively, the targeted response time could translate to a minimum processing power for the VMs in the ensemble.

However, this process is complicated and error-prone: the relationship between non-functional requirements and low level values is in many cases difficult to determine, and many factors can take part in the fulfillment of one individual parameter. Thus, we propose an extension to our system where non-functional requirements can be directly mapped to the execution platform, not only during the deployment phase, but also along the whole design process.

Our proposed approach in this paper extends DEVA annotations with new high-level values that correspond to non-functional requirements. An underlying application-dependent model is in charge of translating high-level parameters to low-level ones, and performing the required operations on the appropriate elements. We describe our system design next, and discuss its implementation.

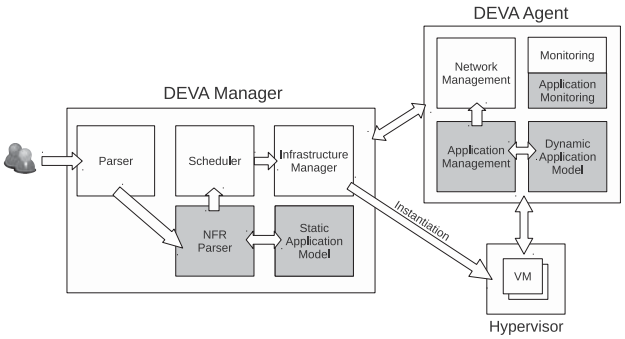


Fig. 2. Extended architecture

A. Processing annotations for DEVAs

Once the user defines a set of nodes, edges and annotations, a request is sent to the DEVA manager, which parses it and assigns the virtual resources to physical hosts that can comply with the requested QoS parameters. At each host, a number of VMs and virtual network links are created so that the ensemble appears as an isolated group of machines with dedicated resources. Additionally, a set of DEVA Agents are in charge of monitoring the infrastructure usage and ensuring global and individual QoS.

Requirements in the request are realized in two phases: first, the scheduling module of the manager chooses the target resources so that the ensemble requirements can be met. This implies selecting hosts with enough free memory and CPU for the VMs and ensuring the underlying physical network will be able to support the requested bandwidth and latency. Second, control measures are applied to constraint resource usage so that shared requests don't interfere among them. VM resources are adjusted by a Virtual Machine Monitor such as Xen or VMWare running in the target host, while network resources are controlled by a DEVA Agent. Agents apply shaping and monitoring to manage network QoS.

In order to implement high-level annotations representing non-functional requirements, we need to extend the DEVA manager and agents so that the new parameters can be translated into the existing ones. Figure 2 shows the system architecture with the new components in a darker shade.

First, the manager needs to translate non-functional requirements into values that can be considered as part of the scheduling process. We provide a non-functional requirements (NFR) Parser that is in charge of converting high-level values to low-level ones. For this, a Static Application Model is employed. Such model is dependent on the type of application, and can be defined either theoretically or empirically. We define a global annotation for the request, where the user can specify the application type. The value of this annotation will determine the model to use in the scheduler.

The Non-Functional Requirements Parser generates a set of requirements for nodes and connections based on the translated parameters, and these are fed to the scheduler, which takes them into account to generate a list of physical machine candidates. Each of the candidates is assigned a number of VMs to host. Finally, the Infrastructure manager, implemented in our system by OpenNebula [7], sends instantiation requests to Hypervisors and the DEVA Agents in charge of the dynamic behavior of the application.

After VMs are instantiated, DEVA Agents create a virtual network in the hosting machines. In our proposed architecture, we extend the system by adding three new components in the agents: First, we define an additional monitoring module with application dependent rules. While the basic component reads values such as CPU, memory and bandwidth usage, new values need to be contemplated in order to track non-functional requirements compliance. Examples of this are requests per second for a web server or database transactions for a database server. The application-dependent monitoring module can be extended based on different applications. All agents send the monitored data to the DEVA Manager, where the data is aggregated to determine high-level actions.

The second change in the DEVA Agents consists in an Application Management module similar to the existing Network Management component. While the later one is in charge of determining low-level actions to maintain network QoS, the new subsystem needs to consider high-level requirements and send them as an input to the other module. The third modification of the agent, the Dynamic Application Model, provides

the mapping based on a model of the application's behavior. Contrarily to the Non-Functional Requirements Parser and the Static Application Model, the components in the agent can also consider the runtime state of the application.

B. Model-based translation of non-functional requirements

There are two modules with the task of translating non-functional —high level— to infrastructure or low-level requirements. As stated in the last section, the first one considers the static behavior of the application and provides the necessary criteria to the scheduler, while the second one takes into account the dynamic state of the application. There are different approaches in the literature to modeling application performance such as [8] or [9], which can be divided into the categories of empirical, theoretical and on-line simulation models.

The first category corresponds to those models created from the application's off-line execution. Requirements can be inferred by observing the behavior of the system under different conditions and creating a model that can be later used to obtain approximate parameters to provide to the underlying management system. These models are usually measured by treating the application as a black-box (*i.e.*, without employing any knowledge of the internal implementation or design).

The second category consists of creating a mathematical model to relate the application's characteristics to its requirements. In this case, knowledge about the internal implementation is used to quantify the application's behavior based on available resources.

Finally, some models perform on-line (runtime) simulation of the application in order to find its behavior for a certain input. Simulations can be either event-based, for which an abstract model is created to approximate the behavior under certain conditions, or real-time, where a part or the whole application is executed to predict how the real system would behave.

Our system does not make any assumptions about the models used to map non-functional requirements to low-level ones. In fact, any of these could be employed either for the static or the dynamic modules in the manager and the agents. The basic prerequisite is that the used model understands the application's requirements and is able to determine a set of values that can be expressed via DEVA annotations. Some models may consider individual components of the system separately, while others contemplate complex relations between modules and how changes in one may affect others.

C. Non-functional requirements fulfillment

The modules added to the system allow the translation of non-functional requirements to low-level ones by using an application model. However, the DEVA Manager and agents need to perform the appropriate actions in order to fulfill the requested requirements. We classify these actions in two areas: resource allocation, and resource control. These categories also correspond to static and dynamic management, respectively.

The first type of actions is decided and enforced by the DEVA Manager based on the initial ensemble request and the model mapping. After parsing the user's input, non-functional requirements are translated into a set of low-level QoS values, which can be in turn used by the scheduler component to assign virtual elements to physical infrastructure. In our implementation in [2], the scheduler executes a metaheuristic algorithm to iteratively choose a near optimal placement of the requested DEVA in the available resources. This mapping would ensure that non-functional requirements are met by employing the appropriate infrastructure. Additionally, the DEVA Manager sends a description of the requested network links to each agent. Agents perform traffic isolation and shaping operations on the physical links to multiplex their usage among ensemble members, and when needed, create tunnels between physical hosts in different domains to build a virtual overlay network.

However, static allocation is not enough to respond to the runtime behavior of the application. While some values can be applied during the instantiation phase, most of the non-functional requirements need to be considered in terms of the application's dynamic usage. The DEVA agent is in charge of monitoring the system's running state and execute the appropriate control mechanisms. In many cases, these actions have associated trade-offs which need to be considered. Examples of control mechanisms run by the agents are dynamic bandwidth or CPU adjustment, provisioning of additional VM instances or VM migration.

V. Experimental Validation

In order to validate the proposed architecture, we have implemented a prototype extending the original DEVA Manager and agent. There are two main goals for this section:

1) Demonstrate the feasibility of translating high-level, non-functional requirements into a deployed ensemble of VMs.
2) Show how high-level QoS requirements are met during a DEVA lifecycle.

The experiment includes provisioning a test application through the DEVA Manager in order to determine how a set of non-functional requirements defined through the requirements analysis phase can be fulfilled during runtime. We have developed a three-tiered web application to illustrate the process.

A. The Chirper Application

In our test scenario, an fictitious company wants to develop an internal messaging systems so that their employees can communicate without having to use third party applications. They decide to deploy this solution in their private cloud so that they can take advantage of their in-house resources. The application, which we call *Chirper*, stores profile information for users, and enables them to post short messages to a common virtual board and query others' messages.

The application has two main components: the first one is a web server running the CherryPy[1] python web server;

[1] http://www.cherrypy.org

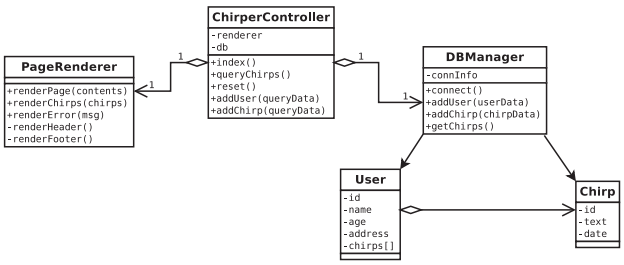


Fig. 3. Chirper class diagram

the second is a PostgreSQL[2] database with the users and messages information. The application will be accessed from the company's intranet.

As the first step, we perform the requirements elicitation to come up with the different functional and non-functional requirements. In this case, users need to be able to register in the application through a form, and then query either a specific user entries or last 50 messages in the database. We focus on a subset of the typical non-functional requirements explained in Section III: after exploring their users' behavior, our fictitious company estimates that the application should be able to respond to a peak of 40 requests per second, and that any request should be served in less than 500 milliseconds through the internal network.

In the second step, we define the application's architecture and implement it. In our approach, we follow the Model-View-Controller (MVC) architecture: a front-end interface where the user can interact with the system, a controller to submit and request data to the database, and the database layer itself. Figure 3 shows a class diagram of the system. The application receives requests through different URLs, which are mapped by CherryPy to the appropriate functions in the Chirper-Controller object. This class handles each request separately, performing input validation, then retrieving the requested information by calling the DBManager class, and finally rendering the response through the PageRenderer instance. The DBManager uses the SQLObject[3] Object-Relational Mapping (ORM) library to access the PostgreSQL database and perform selection and insertion operations. Finally, the PageRenderer class has methods to produce HTML code to return to the user.

In order to simplify scalability and be able to assign physical resources separately to each of the components, the web and database servers are deployed as different Virtual Appliances. Each of the VMs runs CentOS 5.3 and the software requirements needed by the application, which consist of Python 2.4 and CherryPy 3.2 for the Web appliance, and PostgreSQL 8.4 for the Database appliance. When the VMs are provisioned in the cloud, the DEVA Manager is able to define the resource allocation by sending commands to the VM Hypervisor and the DEVA agents. There are three main parameters that can be configurable once the VMs are instantiated in the infrastructure: maximum CPU share assigned to a VM, amount of memory, and bandwidth allocation between pairs of VMs.

[2]http://www.postgresql.org
[3]http://www.sqlobject.org

B. Performance Modeling

Once the application complies with the specified functional requirements, a model is created to account for the expected performance. In this example, a simple black-box model is defined by benchmarking the application externally. We deploy both appliances in the private cloud, consisting of a cluster of machines with Pentium 4 processors and 1 GB of memory running the Xen 3.0 hypervisor, and a third VM to act as a user. Physical machines are interconnected with 1 Gbps ethernet links and a dedicated network switch.

Initially, each VM is assigned a quota of 100% of the CPU, 1GB of disk, and 768 Mb of RAM. We run the Apache Benchmark tool to send 1000 requests with a level of concurrence of 10 to the service for each tested configuration. CherryPy is set up to spawn 10 serving threads without page caching mechanisms. We measure the request time and number of requests served per second for the operation of querying the last 50 messages in the database. We consider CPU and bandwidth as the VMs' configuration parameters. Memory allocation was discarded since the application doesn't require a high amount of main memory and consequently its performance doesn't depend on this parameter (our tests demonstrated 40 Mb were enough for the application to function at maximum capacity).

In the first set of runs, we calculate the application's behavior depending on the CPU quota. After running the tests, we determined that the Database appliance is not CPU bound, and therefore, there is no difference in performance with different values. Figure 4 shows the number of handled requests per second and the milliseconds taken for each request when the CPU allocation for the Web appliance is changed from 25% to 100% in intervals of 25%. As the figure shows, the number of served requests per second is directly proportional to the CPU allocation, while the time taken to respond to each request decreases with a higher CPU quota.

As the second set of measurements, we explore the application's behavior in relation to the allocated bandwidth. There are two links considered in this benchmark: the incoming connection to the Web appliance, and the private connection linking it to the database. Each of them can be constrained and isolated independently by the DEVA agents in the hosting machines. By doing this, each DEVA can perform independently of the rest, and network traffic from different applications is separated so that different VMs can multiplex the physical channel. We test the application with symmetric network assignments —*i.e.* same incoming and outgoing rate— from 100 Kb/s to 500 Kb/s with increments of 100 Kb/s. Figure 5 shows the results in requests per seconds and milliseconds per request for the incoming link (*in*) and the private one connecting both VMs (*priv*).

As it can be observed, the number of served requests per second depends on the available bandwidth, up to approximately 550 Kb/s for the incoming link (not shown in the graph)

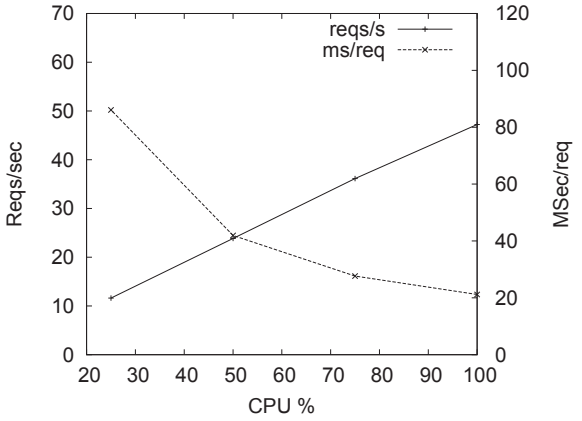


Fig. 4. Application model according to Web Appliance CPU allocation

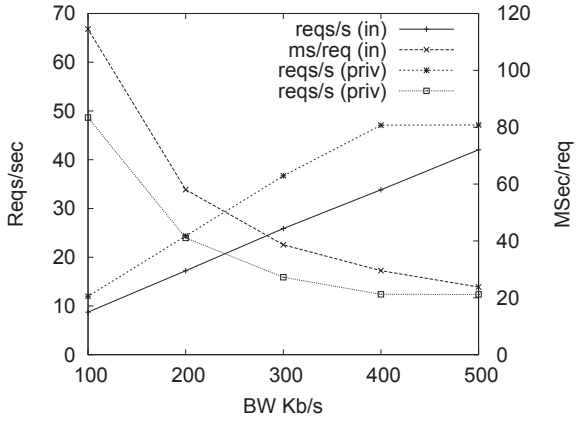


Fig. 5. Application model according to Incoming and Private bandwidth allocation

and 400 Kb/s for the private link. Lower bandwidth results in reduced requests per second and higher response time.

C. Integrating the model with the DEVA manager

The final step consists of integrating the experimental performance model into the DEVA manager and making the appropriate changes so that user requests can specify high-level requirements as parameters. We add the logic to translate such submission into low level parameters by employing the models and specifying an additional global parameter, *applicationType='Chirper'* so that the DEVA Manager knows which model to apply.

After that, we test the system by sending a request to the DEVA Manager specifying the two described Virtual Appliances and the desired non-functional requirements of 40 requests per second and 500 ms of maximum response time. The manager identifies this request to instantiate the *Chirper* application, and translates the requirements to 75% of CPU for the Web appliance and 25% of CPU for the Database appliance, 64 Mb or RAM for each VM, 500 Kb/s for incoming bandwidth and 400 Kb/s for private bandwidth. Finally, it decides that both VMs can be assigned to a single physical machine and provisions them accordingly.

VI. Conclusions and Future Work

As the cloud becomes more mainstream as a method to host applications, developers will need to consider how different providers —or in-house solutions— will be able to fulfill the final users' needs. Similarly, providers need to be able to give reliable guarantees for the Quality of Service of software deployed on their infrastruture. In this paper, we addressed this problem from both the developer's and cloud provider's perspectives. We showed how an example application with concrete requirements can be developed and deployed in a cloud manager that takes high-level non-functional requirements into consideration.

However, there are still many issues to address in order to determine how software can be successfully deployed in clouds: additional non-functional requirements such as fault-tolerance, execution cost or security need to be considered, and improved models that are able to predict applications' performance considering different parameters such as processor, memory, network and disk usage have to be developed.

VII. Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant No. OISE-0730065.

References

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the clouds: A berkeley view of cloud computing," Feb 2009. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html

[2] D. Villegas and S. Sadjadi, "DEVA: Distributed ensembles of virtual appliances in the cloud," in *Proceedings of the 17th Euro-Par Conference (Euro-Par 2011)*, 2011.

[3] L. Youseff, M. Butrico, and D. Da Silva, "Toward a unified ontology of cloud computing," in *Grid Computing Environments Workshop, 2008. GCE '08*, 2008, pp. 1 –10.

[4] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: state-of-the-art and research challenges," *Journal of Internet Services and Applications*, vol. 1, pp. 7–18, 2010, 10.1007/s13174-010-0007-6.

[5] C. Sapuntzakis, D. Brumley, R. Chandra, N. Zeldovich, J. Chow, M. S. Lam, and M. Rosenblum, "Virtual appliances for deploying and maintaining software," in *Proceedings of the 17th USENIX conference on System administration*. Berkeley, CA, USA: USENIX Association, 2003, pp. 181–194.

[6] L. Chung and J. do Prado Leite, "On non-functional requirements in software engineering," in *Conceptual Modeling: Foundations and Applications*, ser. Lecture Notes in Computer Science, A. Borgida, V. Chaudhri, P. Giorgini, and E. Yu, Eds. Springer Berlin / Heidelberg, 2009, vol. 5600, pp. 363–379.

[7] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster, "Virtual infrastructure management in private and hybrid clouds," *IEEE Internet Computing*, vol. 13, pp. 14–22, 2009.

[8] C. Stewart and K. Shen, "Performance modeling and system management for multi-component online services," in *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation - Volume 2*, ser. NSDI'05. Berkeley, CA, USA: USENIX Association, 2005, pp. 71–84.

[9] S. Sadjadi, S. Shimizu, J. Figueroa, R. Rangaswami, J. Delgado, H. Duran, and X. Collazo-Mojica, "A modeling approach for estimating execution time of long-running scientific applications," in *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, 2008, pp. 1 –8.

Computational Reflection in order to support Context-Awareness in a Robotics Framework

Sheila Mendez
R&D Manager
B2B E-Commerce
s.mendez@b2b2000.com

Francisco Ortin
Computer Science Department
University of Oviedo
ortin@lsi.uniovi.es

Miguel Garcia
Computer Science Department
University of Oviedo
be37378@uniovi.es

Vicente García-Díaz
Computer Science Department
University of Oviedo
garciavicente@uniovi.es

***Abstract*—The development of service robots has gained more and more attention over the last years. Advanced robots have to cope with many different emerging at runtime situations, while executing complex tasks. They should be programmed as context-aware systems, capable of adapting themselves to the execution environment, including the computing, user and physical environment. Since computational reflection is a programming language technique that offers a high level of runtime adaptability, we have analyzed the suitability of this language feature to fulfill the dynamism requirements of context-aware robotic systems. In order to evaluate their appropriateness, we have implemented an example scenario in a dynamic reflective language and compared it with Java.**

***Index Terms*—Computational reflection, context-awareness, robotics framework, dynamic languages.**

I. Introduction

Robotic systems which should be able to interact in everyday life have to manage the high dynamism and complexity of real-world environments. An important element to be considered is the environment context, commonly referred to as location, the identification of nearby people and objects, plus changes to those objects [1]. Contexts are defined to be the constantly changing execution environment, including the computing, the user and the physical environment [2]. Therefore, context-aware software is software that must adapt itself according to its location of use, the collection of nearby people and objects, and changes to those objects over time [1]. An advanced robotic platform should take into account the context-awareness requirements of future robotic applications.

Dynamic languages have recently turned out to be suitable for specific scenarios such as Web development, application frameworks, game scripting, interactive programming, rapid prototyping, dynamic aspect-oriented programming and any kind of runtime adaptable or adaptive software. The main benefit of these languages is the simplicity they offer to model the dynamism that is sometimes required to build high context-dependent software. Common features of dynamic languages are meta-programming, reflection, mobility and dynamic reconfiguration and distribution. Computational reflection is one of the most distinguishing features of these languages, defined as the capability of a computational system to reason about and act upon itself, adjusting itself to changing conditions [3]. Due to the strong connection between context-awareness and computational reflection, we present in this paper how computational reflection can be used to facilitate the programming of context-aware services in a robotic platform. In fact, it has been previously stated that in order "to support context-awareness in an open and much larger setting, a reflective, or self-describing, context model is required" [4].

Dynamic languages have been previously used in different robotics scenarios. An example is the SmartTCL language, an extension of Lisp that was used to implement the sequencing layer in a three layer robotic architecture [5]. The sequencing layer is the place to store procedural knowledge on how to configure skills to behaviors. Due to the dynamic features of SmartTCL, the plans stored inside the task coordination module can be modified easily at runtime.

The dynamic language Lua has also been used to implement the behavior engine of the humanoid robot Nao [6]. The formalism of hybrid state machines (HSM) was used to bridge the gap between high-level strategic decision making and low-level actuator control. The model of HSMs was extended with dependencies and sub-skills to call the behaviors or skills hierarchically. Lua turned out to be an expressive language to implement these HSM-skills.

Other examples of existing robotics products that use dynamic languages to obtain a high level of adaptability are UrbiScript, an orchestration language for robotics systems; ROS (Robot Operating System), a set of libraries and tools to help software developers create robot applications in Python; and Pyro (Python Robotics) a programming environment for exploring advanced topics in artificial intelligence and robotics, facilitating the creation of interfaces for accessing and controlling a wide variety of real and simulated robots.

II. The TIC4BOT Project

The work presented in this paper is part of the TIC4BOT Project developed by the Treelogic Company, the Cartif Foundation, and the University of Oviedo. The aim of the project is to provide the necessary infrastructure to develop complex services in the social robotics field, by raising the abstraction level. The implementation was tested over a real robotic platform (SCITOS-G5), though the system was designed with total independence of the hardware. In order to get this independence, the Player/Stage [7] system was used during the development process. The SCITOS-G5 robot is controlled by an embedded PC with an Intel Core 2 Duo processor, and multiple small hardware units that monitor several functions of the robot. The PC operating system is Fedora Core 8.

Figure 1 shows the system architecture, consisting of three layers: primary modules, robotics framework and service modules.

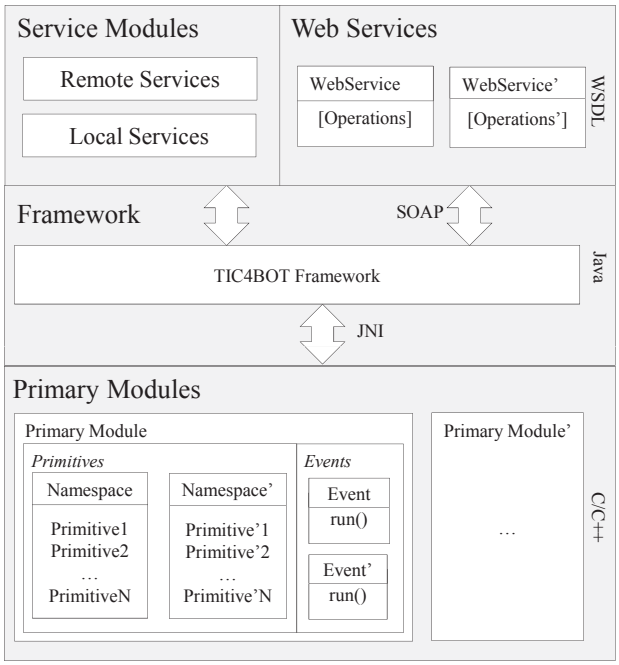


Figure 1. Architecture of the TIC4BOT system.

A. Primary Modules

Primary modules are developed accessing the hardware libraries in C and C++ and they implement primitives and events. A primitive is defined as a low-level function that can perform a simple task or a query over any sensor or actuator of the robotic platform. An event is a notification that something has happened.

In the framework, primitives are classified into *namespaces*. Each *namespace* contains a set of primitives, with a one-to-one correspondence with the primary functions.

B. The Framework

The framework provides a middle layer for the integration of primary modules and service modules [8]. In order to obtain platform independence and a higher level of abstraction, we chose the Java programming language. Java raises the abstraction level, providing automatic memory allocation, garbage collection and multi-threading. However, these benefits come at the expense of lower runtime performance. This implies that the robot hardware should have enough processing capabilities to perform the required tasks. In the case of our project, the SCITOS-G5 processor greatly suffices this requirement. The main functionalities provided by the framework are:

- *Dynamic runtime detection of primary modules*, allowing the addition of new primary modules at runtime. We allow the implementation of these modules in both C and C++, generating a Java proxy class at runtime for each C/C++ module. Proxies are classes providing the access to primary module functions. These Java proxies were developed using the JNI (Java Native Interface) standard through SWIG (Simplified Wrapper and Interface Generator) [9], that provides an automated connection of programs written in C and C++ with a variety of high-level programming languages.

```
// Autentication
Subject subject = null;
subject = Authorization.login("admin", "defaultPassword");
// Retrieve Namespace and Primitive
ApiManager api = FactoryManager.getApiManager(FactoryManagerType.Default);
Namespace namespace = api.getNamespaceByName("Navigation");
Primitive primitive = namespace.getPrimitiveByName("goTo");
// Parameters creation
Object[] parameters = new Object[2];
parameters[0] = x;
parameters[1] = y;
// Invocable element for the framework
InvocationElement invocationElement =
new InvocationElement(subject, namespace, primitive, parameters);
// Create a list of invocation elements
List<InvocationElement> list = new ArrayList<InvocationElement>();
list.add(invocationElement);
// Create the task (Execution Mode)
Task task = new DefaultTask("Sample Task");
task.setInvocationElements(list);
// Create the Runnable (Priority)
ExecutionManager executionManager = FactoryManager.
            getExecutionManager(FactoryManagerType.Default, subject);
Runnable runnable = executionManager.insertTask(task,
            RunnablePriority.MAX, RunnableSyncronism.SYNCHRONOUS);
// Retrieve the result
Object result;
while (true){
    try {
        result = runnable.getResult(0);
        break;
    } catch (NullValueException e) {
        Thread.currentThread().sleep(100);
    }
}
```

Figure 2. Sample code of framework programming.

Figure 2 shows a sample code of the invocation of a primitive. First, authentication over the framework is performed and credentials for executing tasks are obtained. Then, the `Navigation` *namespace* and its `goTo` primitive are obtained. An `Object` array is created containing the values of parameters for the invocation. A list of `InvocationElement` is also created. The `InvocationElement` type is provided by the framework and it could contain a set of primitive elements that will make up a complete task.

- *Transparent publication of all the elements offered by the primary modules*. The framework provides the necessary mechanisms to discover and invoke primitives at runtime. It also provides event subscription management.

  The next step of our sample code in Figure 2 is to create a task with the `InvocationElement` list previously mentioned. In this sample code, the type of the created task is `DefaultTask`. The task type indicates the execution mode of the primitives within the task (a `DefaultTask` indicates non-transactional behavior, while a `TransactionalTask` indicates that the task cannot be interrupted by any other task until it finalizes its execution). Finally, a `Runnable` object is created as the result of introducing the previously created task into the framework execution engine. In this stage, the execution mode is established to `SYNCHRONOUS`. Synchronous tasks cannot be executed in parallel with other synchronous ones, whereas asynchronous tasks can. The task is assigned the maximum priority execution level (`MAX`). The `Runnable` object is then used to retrieve the result of the primitives executed. The loop in the code waits until the result of the primitive is ready.
- *Execution engine for service modules*, providing a prioritization mechanism of tasks and different execution modes (transactional or non-transactional, and syn-

chronous or asynchronous), which can be combined in each specific case.

- *Remote hot reprogramming at runtime*. This service enables remote applications to send their Java source code to the framework at runtime. This was developed through Web Services, using standard technologies. We used the W3C recommended WSDL (Web Services Description Language) for describing Web Services, and the SOAP (Simple Object Access Protocol) protocol to specify the interchange of data with Web Services by means of XML messages.

C. Service Modules

Service modules are complex modules that provide end-user services by means of the composition of primary modules functionalities. The access to primary modules is always performed through the framework. Service modules insert sets of tasks in the framework with a concrete priority and execution mode. These modules can be added at runtime through the hot reprogramming service.

III. Computational Reflection

Reflection is "the capability of a computational system to reason about and act upon itself, adjusting itself to changing conditions" [3]. The computational domain of reflective languages includes their self-representation. Therefore, they can offer their structure and semantics as computable data.

Since context-aware systems should dynamically adapt to runtime changing environments, computational reflection seems to be a suitable technique to face this kind of scenarios. We previously used computational reflection as a suitable technique for adaptive systems such as persistence management [10], dynamic aspect-oriented programming [11] or heterogeneous device support [12].

One classification of reflection considers the observation and modification issues of the system self-representation:

- **Introspection**: Self-representation of programs can be dynamically consulted but not modified. The applications can obtain information about runtime classes, objects, methods, etc. This level of reflection is offered by languages such as Java or C#.
- **Intercession**: The ability of a program to modify its own execution state, interpretation or meaning. Most dynamic languages offer this feature.

Another classification can be established according to *what* can be reflected:

- **Structural Reflection**: System structure can be accessed. In case the system structure is modified, changes will be reflected at runtime. An example of this kind of reflection is the Python feature of adding fields and methods to both objects and classes.
- **Behavioral Reflection**: This level of reflection implies access to system semantics. In case the semantics is modified, it involves a customization of the runtime behavior of programs. As an example, Python offers overriding the semantics of member lookup; if a class has a `__getattr__` method, it will be called whenever a non-existing member is accessed.

Another feature that is commonly used together with computational reflection is runtime generative programming [13]. It consists of the capability of programs to generate new (parts of) programs. This feature is usually offered in conjunction with reflection, because those new parts of generated programs modify the structure or semantics of running applications.

IV. Adaptive Dynamic Programming Layer

The adaptive dynamic programming layer is a new tier over the framework that supports adaptive context-awareness capabilities. This layer is designed to offer runtime adaptation to dynamic environments. Context-aware scenarios often involve the addition of new requirements at runtime, making use of services offered by the framework that are consulted at runtime.

Since dynamic languages offer a high degree of runtime adaptability, we propose their use to provide a runtime adaptive system that additionally provides a simplified way of context-aware tasks. This enhances the framework services with dynamic languages features, obtaining an additional layer with a higher degree of flexibility and adaptability.

Since the framework is developed in Java, an intercommunication mechanism between Java and Python is required. The standard Java Scripting API (JSR 223) [14] allows the use of script engines from Java code. By using it, all the functionality provided by the framework in Java will be accessible at the adaptive dynamic layer using a dynamic language.

We selected the Python programming language to implement the adaptive dynamic programming layer because it is a mature dynamic language that provides runtime structural and behavioral reflection, dynamic generative programming, a simple syntax, a substantial number of powerful libraries and functions.

A. Framework Services

Although the framework provides useful features for many use cases, the code for one single operation over the framework may become verbose (as shown in Figure 2) due to the lack of metaprogramming features in Java. It implies the codification of several lines of code that performs non-functional actions, entailing a less agile development of new services. One of the main goals of the dynamic adaptive layer is to simplify the programming over the framework.

In the adaptive dynamic programming layer, authorization and synchronization data is stored along the entire programmer session, and it is used in all the operations in a transparent way. Furthermore, this data could be changed at any moment by the programmer. Figure 3 shows how the authentication and task priority assignments (and synchronization specification) are performed in the adaptive layer. In addition to this, credentials and synchronization data are stored in the adaptive layer module. When an operation is executed over the framework, this data is retrieved and used.

B. Primitive Management

In the case of primitives, the dynamic adaptive layer allows the final programmers to write their code in a more natural and compact way, facilitating its maintainability and legibility, and without losing any functionality. This layer provides primary module discovery at runtime, making it possible to act over new services discovered at runtime, even if they were not present at design time.

As it was explained in Section 2, primitives in the framework are organized in *namespaces*. At runtime, when a primary module is discovered, its primitives and *namespaces* objects are created. In the adaptive dynamic programming layer, generative programming and structural reflection are used to transparently create classes that wrap the framework services. The resulting code is a Python class that has a one-to-one equivalence for each *namespace* object in the framework. For every object instance representing a `Namespace` with sets of `Primitive` objects in Java, a class with the corresponding methods is transparently created in Python. The purpose of this generation is to provide a simple and natural way to invoke those primitives discovered at runtime. Figure 3 shows the Python source code that invokes the `goTo` primitive at the adaptive programming layer. This code is equivalent to the Java program in Figure 2, being much more compact and legible.

```
from FrameworkLoader import *
# Authorization request
authenticate(USER, PASSWORD)
# Priority, execution mode and name of the task
setTaskType(RunnablePriority.ONE, TRANSACTIONAL_TASK, "HOT-REPROGRAMMING")
# Object corresponding to Namespace Navigation
navigation = Navigation()
# Primitive execution and retrieve of result
res = navigation.goTo(X, Y)
```

Figure 3. Programming over the adaptive layer.

Code generation is implemented using the `exec` Python function. This function receives a character string and evaluates it at runtime. We evaluate strings that generate new classes at runtime. The strings are parameterized with the name of a *namespace* class using the % operator. Figure 4 shows the skeleton string used to generate `namespaces` classes (we do not show the code in the methods for the sake of brevity; it can be consulted in [15]). Generated classes do not contain primitive methods; they provide a mechanism to alter the message passing mechanism through behavioral reflection. Generated classes implement a dynamic lazy search for the invoked method (primitive) using introspection. This search is performed over the framework and it is completely transparent to the final user.

As Figure 3 and its sequence diagram in Figure 5 show, these generated classes make use of the two `__call__` and `__getattr__` built in methods that offer behavioral reflection. The `goTo` message is passed to the `navigation` object. Since this member is not offered by the object, the `__getattr__` method is called. This method stores the name of the requested member (i.e., `goTo`) so it can be later used for searching and invoking the corresponding primitive. Since the returned object

```
classCode = """
# Import sentences
class %(className):
    def __init__(self):
        # Constructor code
    def __getattr__(self, item):
        # Saves the name of the member
    def __call__(self, *args, **kwargs):
        # Invokes the corresponding primitive
"""
```

Figure 4. Sample code to be dynamically evaluated.

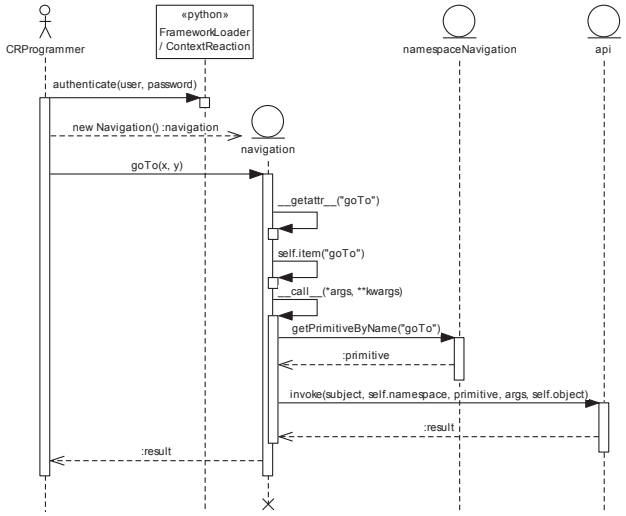


Figure 5. Access to primitive services.

(`navigation`) implements the `__call__` method, it can be called as it was a function. The `__call__` method receives as many parameters as the actual arguments of the call, in a variable-length argument list. This method receives the `x` and `y` parameters, and performs the search of the primitive that fits this signature. Once found, the appropriate primitive of the framework is invoked and the `__call__` method returns the result of calling the primitive.

Computational reflection and generative programming allowed us to fulfill the adaptability requirements without needing to specify all the classes and their methods at design time. This also allows the development of context-aware emerging at runtime scenarios. Furthermore, we can introduce complete service modules on the framework at runtime, using its reprogramming feature. These service modules could introduce new behavior patterns to undertake a specific goal.

We implemented a simple navigation prototype based on fuzzy rules to show the simplicity and adaptiveness provided by the dynamic programming layer. The main `move` function is shown in Figure 6. This function provides the functionality to make the robot advance a step, making use of fuzzy rules.

The rules have the following meaning, expressed with an antecedent and a list of consequents: 1) if there is no obstacle close, go to the target; 2) if the front-left side is freer than the front-right side, turn left and continue until right sonar sensors are free of obstacles; 3) if the front-right side is freer, turn right and continue until left sonar sensors are free of obstacles. An `evaluateFuzzyRules` function evaluates every rule premise and executes the consequents of that rule whose antecedent value is greater.

These three rules provide the logic of decision making for

```
def move(xPos, yPos):
    global x, y
    x = xPos
    y = yPos
    rules = [
        ( fuzzyNot( obstacleClose ),
            [goToTarget] ) ,
        ( fuzzyAnd( obstacleClose, isFrontLeftMoreFree ),
            [turnLeft, moveUntilRightIsFree] ),
        ( fuzzyAnd( obstacleClose, isFrontRightMoreFree ),
          [turnRight, moveUntilLeftIsFree] )
    ]
    evaluateFuzzyRules(rules)
```

Figure 6. Sample fuzzy rules.

the navigation algorithm to avoid obstacles while going toward a specific point. Although this sample logic is very simple, it can be improved by just adding more fuzzy rules describing an optimized behavior of the robot.

Since Python supports first-class functions, fuzzy rules were defined using functions that represent fuzzy operators, predicates and actions. Fuzzy operators (e.g., `fuzzyAnd` and `fuzzyNot`) can be implemented in Python as higher-order functions, i.e., functions that take functions as parameters. The source code is freely available at [15].

C. Event Management

The adaptive dynamic layer enables any single function to subscribe to a concrete event, complementing the framework services and primitive management provided by this layer. In addition, new events can be discovered at runtime, so that programmers can dynamically access new events published dynamically.

Following with our example, let's suppose that we have an artificial vision primary module capable of detecting a person's face. At the moment the face is recognized, the artificial vision primary module triggers an event with the data of the face (e.g., the name of the recognized person). Figure 7 shows an example subscription to a concrete event in the adaptive dynamic programming layer:

- Authentication is performed in the first place. Afterwards, prioritization is established to the maximum, and the asynchronous execution mode is chosen. With these settings, the *Greet* task is executed in parallel with the navigation task that is running in the framework. If the robot is performing any other task at the moment the event is triggered, like for example moving toward any point, it will be able to look to the face detected and greet at the same time, without having to stop its navigation.
- After this, the function that handles the event is defined. This function instantiates a *namespace* of primitives called `Speech`. This *namespace* performs tasks of voice speech synthesis through the `say` primitive, which receives a text and reproduces it by simulating human voice, greeting the recognized person.
- The last statement subscribes the previous function to the face detection event. Therefore, when the event is thrown, the `handlerOfFaceDetectionEvent` function will be automatically called.

By applying generative programming and structural reflection, each event in the framework corresponds to a generated

```
# Authentication and Priorization
authenticate("admin", "defaultPassword")
setTaskType(RunnablePriority.MAX, ASYNCHRONOUS_TASK, "Greet")
# Event Handler
def handlerOfFaceDetectionEvent(data):
    speech = Speech()
    speech.say("How are you " + data.name + "?")
# Event Subscription
event = FaceDetectionEvent(handlerOfFaceDetectionEvent)
```

Figure 7. Face detection event.

Python class. These classes implement the `Observer` interface (from the *Observer* design pattern) provided by the framework. When an instance of any of these event classes is created, a subscriber for that event is also registered. The constructor of this class receives a Python function as parameter. When the framework notifies the occurrence of an event, the subscriber delegates its management to that function.

Because of the dynamic discovery and event generation code, it is possible to add new event types at runtime without having to specify at design time all the events offered by the framework. Furthermore, it allows performing simultaneous tasks, making the most of the prioritization and synchronization features provided by the framework.

D. Remote Reprogramming

The framework execution engine can be remotely reprogrammed at runtime. This remote hot reprogramming allows the introduction of new behavior guidelines at runtime, without having to foresee them at design time, even parallelizing them with other tasks that might be running.

We are interested in allowing the robot to react to the environment context. As an example, we have included in our example the scenario proposed by Pineau *et al.* [16], where the robotic platform has the capability of assisting elderly individuals with an automated reminder system. This system can, for instance, remind to the patient that it is time to have her medicine.

The framework exposes one Web Service for the addition of new service modules from a remote system, which, after authorization, can send the source code to the framework execution engine. The code received is added to the tasks queue, being executed according to its priority and synchronization settings. Making use of the remote reprogramming feature offered by the framework, the *Reminder* module can be added at runtime.

Figure 8 shows the source code we used to develop the reminder scenario. The `Navigation` module shown in Section IV-B is imported, authentication is performed, and the task is set as transactional. This execution mode (transactional) allows obtaining the execution control and avoids other tasks to interrupt it, notwithstanding the parallel execution of other asynchronous tasks (e.g., when the *Reminder* module is running, other asynchronous modules like the *Greeting* one could be executed in parallel). The `patientLocatorWS` reference points to a Web Service that provides the coordinates of patients. Then, a `getPosition` function that retrieves the coordinates of a patient passing her identification it is defined. The main program contains a loop that iterates while the

```
from Navigation import *
# Framework Initialization
authenticate("admin", "defaultPassword")
setTaskType(RunnablePriority.ONE, TRANSACTIONAL_TASK,
            "HOT-REPROGRAMMING")
# Global variables
speech = Speech()
patientLocatorWS = WebService(WSDL_URL)
PATIENT_ID = "1"
navigation = Navigation()
def getPosition(id):
    position = patientLocatorWS.getPosition(PatientId = id)
    return position.getX(), position.getY()
# Main program
x, y = getPosition(PATIENT_ID)
while (not isTargetReached(x, y)):
    navigation.move(x, y)
    x, y = getPosition(PATIENT_ID)
speech.say("It is time to have your medicine, Sir")
end()
```

Figure 8. Reminder module.

target is not reached, performing one more step through the `Navigation` module.

The reprogramming process is initialized by the remote system, which requests the framework for an authorization credential. Then, the remote system would include its own source code like the one in Figure 8. Once this code is included in the execution engine, the *Greet* asynchronous module shown in Section IV-C will be executed in parallel (asynchronously). Thus, the robot would head the patient and, if in its path it detects another person, it would say hello to her, without having to interrupt its navigation—the *Greet* module is asynchronous.

This hot reprogramming system capable of managing runtime emerging primitives and events allows fulfilling the dynamic adaptability requirements of context-aware systems, not only discovering services at runtime, but also adding new programs dynamically. Both computational reflection and generative programming made it possible.

V. Conclusions

Computational reflection seems to be a suitable technique to facilitate the implementation of context-aware scenarios in a robotics framework. The use of a dynamic reflective language allowed us to develop an adaptive dynamic programming layer, which is aware of context environment changes at runtime. This layer offers dynamic runtime discovery of new services and provides transparent programmatic access.

The use of different levels of computational reflection allowed us to achieve the desired degree of runtime flexibility. First, introspection offered the dynamic discovery of services published by the framework and even external devices. We applied generative programming to generate code that wraps these services in Python classes. Generated Python classes were modified by means of structural reflection, adding new attributes to classes and objects when needed. We modified the semantics of the message passing mechanism (by means of behavioral reflection) in order to implement a lazy method search in the generated classes. This allowed us to implement a dynamic adaptive layer, offering the existing services in the current context without the need of specifying them at design time. Additionally, the use of dynamic languages and computational reflection involved a significant simplification of the code to program the dynamic adaptive layer.

The source code of the whole example presented in this paper is freely available at http://www.reflection.uniovi.es/tic4bot. Although the robotics framework is not freely downloadable because it belongs to the TreeLogic Company, the URL above provides a demonstrating video showing the example presented in this paper running on the robotics framework.

Acknowledgments

This work has been funded by the Spanish Ministry of Industry, Tourism and Commerce (TSI-020100-2008-235). It has also been funded by the Department of Science and Technology (Spain) under the National Program for Research, Development and Innovation; project TIN2008-00276.

References

[1] B. N. Schilit and M. M. Theimer, "Disseminating active map information to mobile hosts," *IEEE Network*, vol. 8, no. 5, pp. 22–32, 1994.

[2] B. N. Schilit, N. I. Adams, and R. Want, "Context-aware computing applications," 1994, pp. 85–90.

[3] P. Maes, "Computational reflection," Ph.D. dissertation, Vrije Universiteit Brussel, 1987.

[4] H. Lei, "Context awareness: a practitioner's perspective," in *UDM '05: Proceedings of the International Workshop on Ubiquitous Data Management*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 43–52.

[5] A. Steck and C. Schlegel, "SmartTCL: An execution language for conditional reactive task execution in a three layer architecture for service robots," in *Proceedings of the SIMPAR Workshop on Dynamic Languages for Robotic Sensor Systems*, Darmstadt, Germany, 2010.

[6] T. Niemueller, A. Ferrein, and G. Lakemeyer, "A Lua-based Behavior Engine for Controlling the Humanoid Robot Nao," in *Proceedings of RoboCup Symposium*, Graz, Austria, 2009.

[7] B. P. Gerkey, R. T. Vaughan, and A. Howard, "The player/stage project: Tools for multi-robot and distributed sensor systems," in *Proceedings of the 11th International Conference on Advanced Robotics*, 2003, pp. 317–323.

[8] V. Garcia-Diaz, S. Mendez, J. Barranquero, I. Gonzalez, M. A. Garcia, and J. M. Cueva, "RIF: A reflective integrator framework," in *Proceedings of International Conference on Artificial Intelligence (IC-AI'09)*, 2009.

[9] D. M. Beazley, "Swig: an easy to use tool for integrating scripting languages with C and C++," in *TCLTK'96: Proceedings of the 4th conference on USENIX Tcl/Tk Workshop, 1996*. Berkeley, CA, USA: USENIX Association, 1996, pp. 15–15.

[10] F. Ortin, B. Lopez, and J. Perez-Schofield, "Separating adaptable persistence attributes through computational reflection," *Software, IEEE*, vol. 21, no. 6, pp. 41–49, 2004.

[11] F. Ortin and J. M. Cueva, "Dynamic adaptation of application aspects," *Journal of Systems and Software*, vol. 71, no. 3, pp. 229–243, May 2004.

[12] F. Ortin and D. Diez, "Designing an adaptable heterogeneous abstract machine by means of reflection," *Information and Software Technology*, vol. 47, no. 2, pp. 81–94, Feb. 2005.

[13] C. Krzysztof and U. Eisenecker, *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley, 2000.

[14] M. Grogan. (2006, December) JSR 223. scripting for the Java Platform. Sun Microsystems. [Online]. Available: http://www.jcp.org/en/jsr/detail?id=223

[15] S. Mendez and F. Ortin, "Tic4bot, context-aware robotic platform," 2011. [Online]. Available: http://www.reflection.uniovi.es/tic4bot/

[16] J. Pineau, M. Montemerlo, M. Pollack, and N. Roy, "Towards robotic assistants in nursing homes: Challenges and results," *Robotics and Autonomous Systems*, vol. 42, pp. 271–281, 2003.

A Survey of Software Engineering for Self-Organization Systems

Yi Guo, Xinjun Mao, Cuiyun Hu, Junwen Yin, Jiang Cao
Dep. of Computer Science and Technology
National University of Defense Technology
Changsha, China
atsm@tom.com, mao.xinjun@gmail.com, hcy56316@163.com

***Abstract*—Self-organization systems are rapidly emerging as a powerful paradigm for tacking complexity, improving reliability and optimizing performance in several literatures and are widely applied in a number of domains. Obviously, to develop such complex systems need specified technologies as well as systematic approaches, and they should be developed in an engineering way. We argue software engineering for self-organization systems should consider the specific complexity properties of self-organization and should seek effective approaches to model, design, and deploy such systems. This paper aims to survey the state-of-the-art of the researches on software engineering for self-organization systems from theory, technology and tool viewpoints, investigate the potential issues and challenges in current researches, and suggest future research directions.**

Keywords- Self-Organization systems, Software Engineering

I. INTRODUCTION

As self-organization is a powerful approach to dealing with complexity, it is widely used in several domains to improve the reliability and performance of systems. However, developing self-organization systems in a repeated and effective way is still a great challenge in the literature of software engineering and knowledge engineering. Recently, many researchers have paid attentions to the software engineering for self-organization systems and a lot of relevant research works have been conducted. This paper aims to survey the state-of-the-art of the researches on software engineering for self-organization systems, investigate the potential issues and challenges in current researches, and suggest future research directions. Section 2 gives a survey of the researches on self-organization software engineering. Section 3 discusses the potential issues and challenges in existing researches and suggests some future directions. Finally, conclusions are made in section 4.

II. SURVEYS OF THE STATE-OF-THE-ART OF SOFTWARE ENGINEERING FOR SELF-ORGANIZATION SYSTEMS

We catalogue current works as three dimensions: theory, technology and tool.

A. Theory Aspect

In theory aspect, researchers concern the questions like how to abstract and represent the system in accurate? What mechanisms can be identified and should be utilized to implement self-organization systems?

1) Abstraction and Model

There are three kinds of self-organization abstraction methods for investigating self-organization: differential equations, cellular automata, and agent. These methods are typical based on different metaphors and abstractions.

Differential equation depicts self-organization from dynamic property aspect. Some researchers make use of systems of differential equations to model the self-organized emergence of synchronization with coupled oscillators ranging from fireflies to neurons [1][2]. Although differential equation has range of applicability, it is limited to describe pattern formation, oscillations and other simple collective phenomena. It does not allow enough flexibility for diversity of components and assumes a uniform spatial interaction, neglecting more realistic types of connectivity in real-world systems.

Cellular automata abstraction is a simulation method and has been successfully used for model many real physical self-organization systems from predator-prey systems to chemical spiral waves, to hydrodynamics [3][4]. Such abstraction assumes the world is a discrete grid and each site in the grid can be found in one of a set of possible discrete states. Cellular automata allows for locality, i.e. sites in the grid interact directly only with neighboring sites, thus there is no action at a distance and all signals must propagate along path of connected neighbors. On the other hand, cellular automata is readily exhibit interesting behaviors such as forming a range of patterns similar to differential equations and also far more complex dynamics such as moving patterns[29]. A fundamental requirement for cellular automata to exhibit interesting behaviors is synchronous updating. There are simple algorithms which do allow collections of objects to synchronize, so it may be possible to engineer a system to fit the cellular automata.

Agent-based abstraction is a natural metaphor if components in a self-organization system can vary dramatic from one another and display a range of behaviors and strategies like decision making. In contrast with Cellular Automata, agents do not need to assume an underlying topology, specifying or restricting which components can interact with, and they can come together and interact with each other in complex and dynamic ways. These characteristics are important if we want to model interactions among humans with complex decision making abilities. Many researchers use Multi-Agent Systems (MAS) as a basic abstraction to model

self-organization systems. [5]-[11]show different models based on MAS. Agent-based approaches are extremely flexible to model the interactions among collection of autonomous agents. They can model phenomena in social systems as well as bio-systems. However, a major drawback of agent based modeling is the lack of rigorousness. Because of the complexity of the model specifications, it is difficult to assess the robustness of observed phenomena to change in the specifications and the accuracy by which the model describes the real systems.

2) Self-Organization Mechanism

Self-organization mechanism guides the behaviors of individuals, it determines how the individuals in the system act and interact, as well as restrict the emergent result of the self-organization system. As bio-systems and human societies exhibit self-organization properties, current mechanisms are almost inspired by bio-system or human society. [30] discusses mechanisms which are inspired from bio-systems and gives some application examples.[12]surveys mechanisms from human societies. Besides, some researchers discuss the engineering principles for special mechanisms, for example [31] discusses the engineering principles from stigmergy. These mechanisms provide good inspirations for implementing of self-organization.

B. Technology Aspect

In most of researches on self-organization systems, agents often as the individual metaphor are represented and designed. However, as argued in [13], most current approaches disregard the macro-scale issues and focus on development of micro-scale MAS. Researchers need to find new technologies and methodologies for engineering self-organization systems.

1) Methodology

The methodologies for self-organization systems often take agent as the entity of system. These methodologies care about how to construct a self-organization system? What principles and processes are needed in the development process? From life cycle aspect, these researches concern about different phases of software life cycle, e.g. [17][19][15]concern all of the phases in life cycle from requirement analyzing phase to testing phase. [19]proposes a domain-independent methodology for designing and controlling self-organization systems. The iterative and incremental methodology includes five steps: Representation, Modeling, Simulation, Application and Evaluation, which are interrelated, and in each step, corresponding development actions are also proposed. [18] And [20]are engaged in design phased and [20]also considers implementation phase. At the design phase, [18] proposes a design guide for swarming systems engineering consisting of ten design principles, in [20] the requirements related to the properties of components as well as the rules that guide their behavior and how the development process has to be carried out are identified. During the implementation phase, the run-time infrastructure, components, policies, and metadata are developed.

In modeling languages aspect, there are still no effective modeling languages to support the modeling of self-organization systems. The reason is that the macro properties of self-organization systems are often novel for the individuals, so representing of the relationship between macro properties and individual behaviors is difficult. Some researchers try to solve this problem, for example [14] [16], however the results are limited.

2) Implementation Technology

Autonomy of entities and emergence also bring challenges to the implementation of self-organization systems. How to realize this kind of agents is an important problem.

- Middleware

Middleware in self-organization systems is used for supporting implementation of the agent which is adaptive and uncoupled interaction mechanisms as well as context awareness. Many self-organization mechanisms are such mechanism e.g. stigmergy, schooling. The representative work is TOTA [26]. It is a spatially distributed tuples. Agents can inject these tuples in the network, to make available some kind of contextual information and to interact with other agents. Tuples are propagated by the middleware, on the basis of application specific patterns. Agents can locally sense these fields and rely on them for both acquiring contextual information and carrying on distributed self-organization coordination activities. The similar works include Anthill [25]and MMASS [33].

- Architecture

In agent architecture aspect, researchers referred to some theories that make agents easy to realize self-organization. For example [34] refers to AMAS theory (Adaptive Multi-Agent Systems) proposes a three level architecture. When agent aware the changes of its environment, higher level can choose the best state from self-organization of lower level. [35] proposes an observer/controller architecture which refers to Cybernetics. In system architecture aspect, researchers provide conceptual framework which can guides the interplay of application dependent component functionalities and their coordination and therefore prepares a constructive approach to the utilization of self-organization, for example SodekoVS project [22]. [36] considers either agent architecture or system architecture.

- Design pattern

Software reuse in self-organization applications is also attracted the interest of some researchers, who want to design some general self-organization design patterns from some self-organization mechanisms and expect to apply them in different applications. For example [28] distinguishes five patterns from a lot of self-organization applications. These patterns are described in a format description and a catalogue of these patterns by the name of "Problem/Solution" is proposed with the aim of helping engineers find the applicable patterns or mechanisms for their developing systems. Such researches offer some design patterns for self-organization applications, but they lack a strict representation, all of these patterns were represented by nature language.

3) Testing technology

Self-organization systems will only be acceptable in large scale applications if they can guarantee that they will accomplish exactly what they were designed for. Therefore, we need a practical and efficient method to anticipate and verify this emergent behavior. However, because of the autonomy of entities and macroscopic emergence, self-organization-based testing technology is still very few. [37]presents a method to verify self-organizing multi-agent systems using an autonomic computing approach and based on online planners and self-configuration.

C. Tool Aspect

There are only a few works in tools aspect. They can be classified in three aspects: 1) Self-organization MAS development tools. The tools have a database which stores the behaviors of agents, and these behaviors are refined from self-organization mechanisms. Behaviors are chosen by policy and are loaded into the behaviors of agents when the system is running. 2) Self-organization MAS simulation platform. Designers can model their designed mechanism by such tools to simulate the real running of systems and observe the running result.3) Self-organization system running environment. These environments are distributed implementation of self-organization systems that provide low-level functions such as communication, security, resource management and mechanism scheduling. Table 1 illustrates these tools.

TABLE I. SELF-ORGANIZATION DEVELOPMENT TOOLS

Tools	Modeling	Implemen-tation	Simulation	Running environment
VAstido[21]	√	√		
Netlog[23]			√	
MASON[24]			√	
Anthill[25]		√		√
DIET[27]		√		√

III. CHALLENGES AND DIRECTIONS

C1: Effective and enabling mechanism of self-organization

An important challenge in engineering is to research how we can translate the insights of self-organization we have got into decentralized software mechanisms that allow controlling distributed complex system efficiently. The development of such mechanisms, for example reusable frameworks can be a huge support for software engineers to construct self-organization systems. Designers can select the mechanisms which can achieve the expected global coherent behavior, and realize these mechanisms into design of systems easily. We need to propose a framework of self-organization mechanisms which needs to meet two requirements. Firstly, the framework needs to have a standardized format and sufficient capability to express different mechanisms. Secondly, the mechanisms represented by this framework are easy to be translated to the decentralized algorithm for different applications.

C2: Breakthrough of modeling technology

A great challenge for modeling this kind of system during the development process is how to model the macro properties of system as well as the behaviors of individuals in the system. Current modeling technologies just care about the microscope individuals. We first need a general modeling language with formal or semi-formal semantics to either represent the behaviors of individual or the system macro properties. The modeling language should have a multi-level expressional capability to represent the self-organization system from different aspects, and effectively support represent the characteristics of this kind of systems like dynamic, autonomous, and local interactions etc. In addition, a challenge consists in current research works is that models of the self-organization systems are always from different abstract levels and different viewpoints, as a result, these models are inter-correlative. Some technical methods like strategy, tools which can support the consistency checking for these multi levels models are needed.

C3: Change in the development process

Current methodologies for engineering of self-organization systems often adopt the traditional development process like RUP [17]. However, Self-organization system is a kind of complex systems, the autonomy of entities and emergence of system make the traditional process models can not effectively support the development of such system, traditional top-down approach is hard to analyze the emergence of system macro properties which can not be refined to the behaviors of individuals. We need to find some innovative and effective process models to construct self-organization systems, for example bottom-up process. In this process, emergence of system macro properties is not acquired from refining of requirements, but from the designing of individual agents and their interactions. This approach is an Ad hoc manner which always needs a lot of experiments and experience of designers.

C4: Software quality assurance technology

Software quality is very important for software engineering. An effective means of achieving quality assurance is a major pre-condition for large-scale application of self-organization systems. However, software quality assurance technologies for self-organization are still lacking. We need some methods to assure the software quality of self-organization systems. For example, testing technologies for individual of self-organization system, it is need to study the agent software testing technologies in different phrase of life cycle and overcome the challenges of complexity of self-organization.

C5: Effective Tools and Platforms support

Although some tools for self-organization systems have been proposed [32], they still have some drawbacks. E.g., some important development processes like software testing are also short of supported by tools. Effective tools can great improve the development efficiency and quality. In self-organization system field, future work should encourage the development and refinement of development tools from two aspects. 1) With respect to the software life cycle, development technologies for different phrases of life cycle need to be implemented, which is the foundations of effective tools. The appearance of tools depends on the maturity of corresponding technologies. 2) Current tools need to be improved from many aspects: reliability, friendly, simplicity, and integration.

IV. CONCLUSION

In this paper, we analyzed the complexities of self-organization systems and the challenges that the complexities bring to software engineering. We also have surveyed the state-of-the-art in software engineering for self-organization systems, focusing on three core areas of theory, technology and tool. Based on this we have identified some key areas where we feel the state-of-the-art is lacking, and could be improved. For each of these key topics we have discussed ways for meeting the challenges, and suggested some possible directions for the future research.

ACKNOWLEDGMENTS

This work is supported by the Natural Science Foundation of China under Grant No. 61070034 and 90818028, 973 Programme of China under granted No. 2011CB302601.

REFERNCES

[1] R.E.Mirollo and S.H. Strogatz. Synchronzation of Pulse-Coupled Biological Oscillators. SIAM Journal on Applied Mathematics, 50(6), pp.1645-1662, 1990.

[2] A. Pikovsy, M. Rosenblum, and J. Kurths. Synchronization: A Universal Concept in Nonlinear Sciences. Cambridge University Press, Cambridge, England, 2003.

[3] B.Chopard and M.Droz. Cellular Automata Modeling of Physical Systems. Cambridge University Press, Cambridge, England, 1998.

[4] L.B.Kier, P.G.Seybold, and C-K. Cheng. Modeling Chemical Systems Using Cellular Automata. Springer Netherlands, 2005.

[5] Candelaria S, Juan P. An Adaptive Agent Model for Self-Organizing MAS. Proc. The 7th International joint Conference on Autonomous Agents and Multiagent Systems. pp.1639-1642, 2008.

[6] Vincent, H., Abder, K., Sebastian, R., An adaptative agent Architecture for Holonic Multi-Agent Systems. ACM Transactions on Autonomous and Adaptive Systems. Vol.3(1),article 2, 2008.

[7] Schillo, M., Fischer, K., Fley, B., Florian, M., Hillebrandt, F., Spresny, D.: FORM-A Sociologically Founded Framework for Designing Self-Organization of Multiagent Systems. LNCS ,vol.2934 ,pp.156-175, 2004.

[8] Sansores, C., Pavon, J.: An Adaptive Agent Model for Self-Organizing MAS. International.Proceedings of te 7th International Join Conference on Autonomous Agents and Multi Agent Systems, pp.1639-1642. 2008.

[9] Lieru, M., Este R.: The Holonic enterprise and Theory of Emergence. International Journal Cybernetics and Human Knowing, vol.11,pp. 79-99 , 2004.

[10] Perozo, N., Aguilar, J., Teran, O.: Proposal for a Multiagent Architecture for Self-Organizing systems. LNCS, Vol. 5075, pp. 434-439, 2008.

[11] Parunak, H.V.D., Brueckner, S.A., Matthews, R.: Pheromone Learning for Self-Organizing Agents. IEEE Transactions on Systems Man and Cybernetics Part A Systems and Humans,Vol, 35,pp.316-326, 2005.

[12] Hassas, S., Giovanna, D.M., Karageorgos, A., Castelfranchi, C.: On Self-Organising Mechanisms from Social, Business and Economic Domains. Informatica, Vol30(5),pp. 63-71, 2006.

[13] F. Zambonelli and A. Omicini. Challenges and research directions in Agent-Oriented Software Engineering. Autonomous Agents and Multi-Agent Systems, Vol. 9(3). pp. 253-283, 2004.

[14] M.A.de C. Gatti, C.J.P. de Lucena, P.S. C. Alencar, and D. Cowan. Self-Organization and Emergent Behavior in Multi Agents Systems: A Bio-inspired Method and Representation Model. unpublished

[15] M Morandini, F Migeon, M P Gleizes, C Maurel, L Penserini, and A Perini. A Goal-Oriented Approach for Medelling Self-Organising MAS.LNCS 5881, pp.33-48, 2009.

[16] L Gardelli, M Virol, A Omicini. On the Role of Simulations in Engineering Self-Organizing MAS: The Case of na Intrusion Detction System in TuCSoN. LNCS 3910. pp.156-166, 2006.

[17] T D wolf, T Holvoet. Towards a Full Life-cycle Methodology for Engineering Decentralised Multi-Agent Systems. Proc. The fourth International Workshop on Agent-Oriented Methdologies, 2005.

[18] H V D Parunak, S Brueckner. Engineering Swarming Systems. Methodologies and Software Engineering for Agent systems. Kluwer ,pp.341-376, 2004.

[19] C Gershenson. Design and Control of Self-Organizing Systems. PhD thesis, Faculty of Science and Center Leo Apostel for Interdisciplinary Studies, vrije Univ., Brussels, Belgium, 2007.

[20] G.D.M.Serugendo, J S Fitzgerald, A Romanovsky. MetaSelf: An Architecture and a Development Method for Dependable Self-* Systems. In Proceedings of SAC' 2010. pp. 457-461. 2010.

[21] M. Wang, L. Qiu, F. Lin, Z. Shi. A Multi-agent System Development Tool Support for Self-Organization. International Journal of Computer Science and Network Security, Vol. 6(9), pp.32-40,2006.

[22] J.Sudeikat, L.Braubach,A.Pokahr, W. renz and W.Lamersdorf. Systematically Engineering Self-Organizing Systems. The SodekoVS Approach. Electronic Communications of the EAAST Vol 17, 2009.

[23] U Wilensy, M Resnick. Thinking in Levels: A Dynamic Systems Approach to Making Sense of the World. Journal of Science Education and Technology. Vol. 8(1), pp. 3-18, 1999.

[24] S.Luke, C.C.Revilla, L.Panait, and K.Sullivan. MASON: A New Multi-Agent Simulation Toolkit. Proceedings of the 2004 Swarmest Workshop, 2004.

[25] O Babaoglu, H Meling, and A Montresor. Anthill: A Framework for the Development of Agent-Based Peer-to-Peer Systems.In Proceedings of the 22nd International Conference on Distributed Computing Systems. pp.15-20,2002.

[26] M Mamei, F Zambonelli. Self-Organization in Multi Agent Systems: A Middleware Approach. Engineering Self-organising Systems. LNCS 2997, pp.233-248,2004.

[27] P Marrow, M Koubarakis. Self-Organising Applications Using Lightweight Agents. LNCS 3910,pp. 120-129, 2006,

[28] Wolf, T. D.. A Catalogue of Decentralized Coordination Mechanisms for Designing Self-Organizing Emergent Applications. Department of Computer Science.K.U.Leuver, Belgium. Report. 2006.

[29] M.Gardner. Mathematical Games: The Fantastic Combinations of John Conway's New Solitaire Game "life". Scientific American, vol,223,pp. 120-123, 1970.

[30] Camazine, S., Deneubourg, J., Frank, R.N., Sneyd, J., Theraulaz, G., Bonabeau, E.. Self-Organization in Biological Systems. Princeton University Press, USA.2001.

[31] H.V.D.Oarunak. "Go to the Ant": Engineering Principles From Natural Multi-Agent Systems. ANNALS OF OPERATIONS RESEARCH. Vol.75(0), pp. 69-101, 1997

[32] C.Bernon, V.Camps, M.P.Gleizes, and G.Picard. Tools for Self-Organizing Applications Engineering. Engineering Self-organising Systems, LNCS 2997, pp. 283-298, 2004.

[33] S. Bandini, S. Manzoni, C. Simone. Space Abstractions for Situated Multiagent Systems. 1st International Joint Conference on Autonomous Agents and Multiagent Systems, ACM Press, pp. 1183-1190, 2002.

[34] G. Picard and M.P.Gleizes. An Agent Architecture to Design Self-Organizing Collectives: Principles and Application. LNAI 2623. Adaptive Agents and MAS. pp.141-158, 2003.

[35] H. Schmeck, C.M.S.Cakar, M.Mnif, U.Richter. Adaptivity and Self-Organization in Organic Computing Systems. ACM Trans. On Autonomous and Adaptive Systems. Vol 5(3), 2010.

[36] N.Perozo, J Aguilar, and O. Teran. Proposal for a Multiagent Architecture for Self-Organizing Systems LNCS 5075, ISI workshop,pp.434-439, 2008.

[37] Bruno de C.B.A., Maira A. De C.Gatti, Carlos J.P. de Lucena. Towards Verifying and Optimizing Self-Organizing Systems through an Autonomic Convergence Method. Unpublished.

Self-Management of External Device Failures in Embedded Software Systems

Michael E. Shin
Department of Computer Science
Texas Tech University
Box 43104, Lubbock, TX 79409
1-806-742-3527
Michael.Shin@ttu.edu

Poonam Mane
Department of Computer Science
Texas Tech University
Box 43104, Lubbock, TX 79409
1-806-742-3527
Poonam.Mane@gmail.edu

Abstract

Embedded software systems interact with several external devices that may come across faults or failures. Most of the approaches have focused on software and hardware systems, but relatively less attention has been given to self-management of faults or failures of external devices in embedded software systems. It is necessary to develop an approach such that faults or failures of external devices in embedded software systems are detected and self-managed.

This paper describes an approach to establishing a framework for detecting and self-managing the failures of external devices for embedded software systems. Failed external devices are identified based on external device types, being self-managed in different ways. The proposed approach in this paper is applied to the use case modeling of an elevator system case study.

1. Introduction

Embedded software systems need to be more reliable against faults or unanticipated events so that the systems become more resilient to the system failures. Embedded software systems may still contain software faults in terms of specifications, semantics, logics, resources, and synchronizations even though the systems are verified and validated during development. Embedded software systems are running on hardware, which may fail due to hardware failures like aging, on-board circuit issues, peripheral or accessory maintenance. Previous approaches [Cheng06, IBM06, Roy07] have focused on failures in software and computer hardware [Coyle04, Pinheiro07, Wikipedia10].

However, not much attention has been given to faults or failures of external devices in embedded software systems, such as elevator systems and robotic systems. An elevator system should control several external devices such as direction lamps, elevator buttons, door, and motor. These external devices can run into some faults or malfunctions, which may lead the systems to failures. The software for embedded systems may not work correctly even though these external devices have minor faults or malfunctions.

This paper describes an approach to establishing a framework for detecting failures in the external devices of embedded software systems and for self-managing the detected failures. This paper develops the detection framework based on the external device types in embedded systems. There are unique communication patterns between embedded software systems and their external devices. These patterns are used to devise failure or fault detection mechanisms. Further, the self-managing framework is developed for handling external device failures from the software perspectives. The self-management is carried out by means of self-healing, self-configuration or adaptation, and self-reporting. The proposed approach in this paper has been applied to the use case modeling of an elevator system.

This paper is organized as follows. Section 2 describes classification of external devices. Section 3 describes the devised framework for fault detection, followed by self-management of external device failures in section 4. Section 5 describes the use case modeling for detecting and self-managing failed external devices. This paper is concluded in section 6.

2. Classification of External Devices

There are different external devices in software systems, which can be classified into several types [Gomaa00]. A type of external devices has the unique communication pattern between the devices and the system. This uniqueness of communication styles is used to develop a framework for detecting faults or failures in external devices. Each device type is categorized by means of device characteristics, such as input/output, periodic/non-periodic, asynchronous, or active/passive.

- Input/output devices: An input or output device can provide input to or receive output from the system respectively. Some devices have only one role of either input or output, or both input and output. For example, an elevator button is an input device and an elevator lamp is an output device in an elevator system.
- Periodic/non-periodic devices: A periodic device interacts with the system periodically, whereas a non-periodic device interacts with the system as needed. For example, a brake is a non-periodic device, and an engine is a periodic device in a cruise control system.
- Asynchronous devices: An asynchronous device generates an interrupt to send a message to the system, which then reads a message from the device. Examples of this type are elevator buttons in an elevator system and cruise control buttons in an cruise control system.
- Active/Passive devices: An active device cannot generate an interrupt to be sent to the system, but it may send directly a message to the system. A passive device cannot generate an interrupt for communication with the system and cannot send a message to the system. The motor and the elevator door in an elevator system are passive.

3. Detection for External Device Failures

3.1 Failure Detection of Asynchronous Input Devices

An asynchronous input device generates an interrupt and delivers it to the device interface whenever it meets some event interesting to the interface. When the interface receives an interrupt from an asynchronous input device, it may need to read a message from the device. An asynchronous input device is characterized as input, asynchronous, and non-periodic. An asynchronous device may not generate any interrupt if it fails. In that case, the interface cannot detect the failures of the asynchronous input device. Examples of this type of asynchronous external devices are floor buttons and elevator buttons in the elevator system.

However, there are some exceptional asynchronous input devices whose failures may be detected under some conditions. An arrival sensor in the elevator system is an example of the asynchronous input device whose failures are detected using other devices in the system. Figure 1 shows the communication between the arrival sensor and the elevator control. An arrival sensor is installed at each floor, sensing the arrival of the elevator at the floor. As an elevator reaches a specific floor, the arrival sensor sends an interrupt to the arrival sensor interface, which notifies the elevator controller of the message arrival. The elevator controller checks if the elevator should stop at the floor or not. When an elevator is going up, the i-th floor arrival sensor may be presumed that it fails if the elevator controller receives an arrival message from the (i+1)th arrival sensor without a message from i-th arrival sensor. Similarly, this can be applied when the elevator is going down.

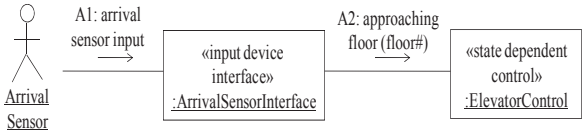


Fig. 1 Communication Diagram for Arrival sensor and Elevator Control Objects

3.2 Failure Detection of Periodic Input Device

A periodic input device delivers inputs to the system periodically, but it is not an active object. A periodic input device is characterized as input, periodic, and passive. A system polls a periodic input device to get an input from the device on a regular basis. When the system polls the device, the device is presumed that it fails if the system cannot read an input from the device upon each poll.

The engine in a cruise control system is the example of this type of device. The engine is polled by the engine interface to read an input at periodic intervals. When the engine does not return its input to the interface, it may meet a failure. Figure 2 shows the engine failure detection in the cruise control system. Also failures of an engine can be detected in another way. If the system does not receive the normal heat temperature values from the heat sensor for a certain period of time, the system detects the failure of engine.

3.3 Failure Detection of Periodic Output Device

A periodic output device receives output from the system periodically and handles the output. A periodic output device is passive, thus it just waits for an output from the system. Failures of a periodic output device may not be detected by the system. This is because the device may not respond to the system when an output is arrived. But failures of periodic output devices may be detected by users. Examples of such devices are a mileage display, a trip average display and a maintenance display in a cruise control system, and a microwave display in a microwave system (Fig. 3). These displays show some information to users in which the information is changing. Users can detect the failures of these displays when the information is not changing.

Some periodic output devices can be monitored by polling the devices. Such an example is the throttle in the cruise control system. The throttle is classified as a periodic output device whose failures may not be detected. But a throttle is a polled device whose failure is detected when the throttle position sensor does not return any value to the system within a defined period of time.

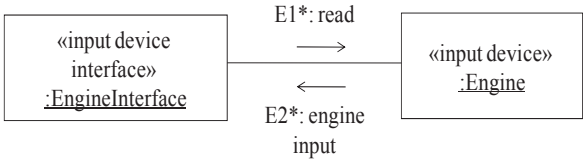


Fig. 2 Engine sensor failure detection

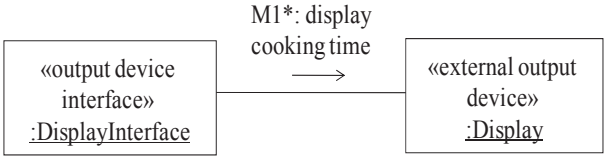


Fig. 3 Microwave display failure detection

3.4 Failure Detection of Passive Output Device

The system sends an output to a passive output device as needed, which handles the output. A passive output device is checked for the status by the system after an output is sent to the device. A passive output device is characterized as output, passive, and non-periodic. The motor, elevator lamp, and elevator door in the elevator system are examples of passive output devices.

The status of the motor (Fig. 4) in the elevator system is determined by checking the rotation/motion sensor after an output, such as motor start or stop, is being delivered to the motor. When the rotation/motion sensor does not return a valid response to the elevator system within a defined period of time, it leads to a conclusion that the motor has failed. Similarly, the elevator direction lamp is monitored by the voltage sensor, and the elevator door's failures are detected by the rotation sensor status.

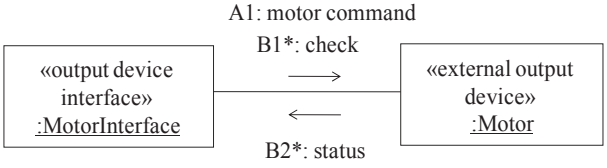


Fig. 4 Elevator motor failure detection

4. Self-management of External Device Failures

4.1 Self-healing

Failures detected in external devices may be resolved by retrying the communication with the devices. Similar to software, an external device may meet a failure attributed to some minor mechanical error or instant error. For failures of a periodic input device, the system reads an input from the devices again. For instance, an engine interface retries to read an input from an engine if there is no response from the engine. A passive output device interface retries to send an output to the device, and polls the status of the device again. For example, the motor in the elevator system may be retried if it meets some failures.

Failed external devices can be reinitialized by calling the initialize() operation, which initializes the devices and its related variables. A device interface object has the initialize() operation that is called at initialization time. However, the system calls the initialize() operation if it meets failures in the devices, and the system retries to communicate with the failed devices.

The system controller switches failed external device to another device if a secondary device is available. The secondary device has the same functionality as the primary device. The switch between a primary device and a secondary device is carried by the device interface, but the system does not require reconfiguration of software components constituting the system. An external device may not have its secondary device if the device is not critical to the system or if the secondary device is not cost-effective.

The self-healing approach can be applied to the external devices that are non-time critical. A failure of external devices can be categorized with response time, such as time critical or non-time critical. A time critical device's failure is intolerant if the device does not work on time. A time critical device does not have enough time to self-repair the failures. A failure of time critical device may shut down the entire system. The brake or engine in a cruise control system is an example of the time critical device. A non-time critical device is not a time bound device whose failure may be fixed by self-repair. A floor button in an elevator system is an example of non-time critical device.

The self-healing approach for external devices has some limitations that may not fix failures attributed to real mechanical problems. An external device with a mechanical problem needs to be replaced physically by hardware engineers if the secondary device is not installed with the primary device. For example, an elevator direction lamp may be burn out due to the life time, and the motor of an elevator may meet a mechanical problem that cannot be fixed by retrying or initializing it.

4.2 Self-configuration or adaptation

Failure analysis enables us to analyze the impact and the consequences of the failed device in the embedded system. The failures can be classified as tolerable, serious and catastrophic. A failure of external devices that does not disrupt the normal functioning of the system is termed as a tolerable failure. An example is a failure of a direction lamp of an elevator. Even though a direction lamp fails, the elevator works fully without degrading its services to passengers. A serious failure in the external devices degrades the functions of a running system, but it may not necessarily bring the system to a total failure. For example, an elevator button's failure makes some passengers not reach to a specific floor, but the elevator still works. Another example is a failure of arrival sensor at a specific floor in an elevator. A catastrophic failure is irrecoverable, and it brings the entire system to a sudden halt. The examples are failures of the motor in an elevator system, and failures of the brake in a cruise control system.

The embedded software may need to be reconfigured or adapted against tolerable or serious failures of external devices so that the system prevents from the ripple effect in terms of the failures – worse performance or additional failures. When a device meets either a tolerable or serious failure that cannot be self-repaired, the system should not communicate with the device in order to send an output to or read some input from the device. This communication can make the device interface meet the same failure, and try to self-repair it using retry or re-initialization. Even the same failure may be notified to the system controller, which already knows the device status. For example, an elevator system should not send an "on or off" message to an elevator direction lamp if this lamp is already detected to be failed. Otherwise, the elevator direction lamp will try to self-repair the same failure again and the system's performance will be getting worse.

A failure of external device may bring another failure to the system if the system is not adapted against the failed device. Suppose that multiple elevators are operating in a building and an arrival sensor for a specific floor of an elevator fails. The scheduler of the elevator system should not patch the elevator to the floor with a failed arrival sensor. The elevator cannot stop at the floor due to the failure of arrival sensor. If the elevator is patched to the floor, the elevator may continue to go up or down repeatedly to reach the floor.

4.3 Self-reporting

The embedded software systems can be divided as unmanned control or manned control systems. The elevator system is an example of unmanned control system, whereas the microwave and the cruise control systems are manned control systems. This criteria is made based on to whom the non-self managed failures should be reported. In the case of the unmanned control systems, the maintenance centers are notified while the users are notified in the manned control systems. When a device failure is detected, the failure can be either detected by the device interfaces, or users. If a catastrophic device in the unmanned control system fails and there is no way to self-manage, the system should stop and report to appropriate person or maintenance center.

5. Detection and Self-management in Use case Modeling

The proposed approach for detection and self-management of failed external devices is applied to the use case modeling for the elevator system, cruise control system and microwave system [Mane10]. The following is the Stop Elevator at Floor use case [Gomaa00] in the elevator system, which is extended to

detect and self-manage failed external devices – arrival sensor, motor, floor direction lamp, and elevator door.

Name: Stop Elevator at Floor Use Case
Actors: Arrival sensor
Precondition: Elevator is moving.
Description:
As the elevator moves between floors, the *arrival sensor* detects that the elevator is approaching a floor and notifies the system. The system checks whether the elevator should stop at this floor. If so, the system commands the *motor* to stop and turns on the *floor direction lamp*. When the elevator has stopped, the system commands the *elevator door* to open.
Alternative for Application:

- The elevator is not required to stop at this floor and so continues past the floor.

Postcondition: Elevator has stopped at floor with door open.
Detection and Self-management of External Device Failures:

- Arrival sensor (asynchronous input device). **Detection -** the system may not detect the failure, but it may detect it using neighbor sensor input.
  Self-management - if an arrival sensor fails and the elevator needs to stop at the floor, the system stops the elevator at the next (up or down) floor to drop passengers. The system self-reports this failure to the maintenance center if the arrival sensor does not send the signal to the system continuously.
- Motor (passive output device). **Detection** - if the motor does not return the status within a defined time or return an abnormal door sensor value, the system detects the failure of motor. **Self-management** – the system reinitializes the motor and commands it to start moving the elevator. If the motor is still not moving the elevator and the elevator is located at a floor, the system commands the elevator door to open so that passengers can go out of the elevator. The motor is a critical device, so the system self-reports this failure to the maintenance center. The system does not allocate this failed elevator to user requests.
- Floor direction lamp and Elevator door (passive output devices). **Detection** - if the floor direction lamp (elevator door) does not return the normal value, the system detects the failure of the floor direction lamp. **Self-management** – the system reinitializes the floor direction lamp (elevator door) and goes to the next use case description step because it is not critical device. Since it is a non-critical, tolerable failure, the system reconfigures the failed floor direction (elevator door) lamp interface so that the system does not send messages to the failed device. The system self-reports this failure to the maintenance center.

The use cases for the elevator system with detection and self-management of failed external devices are implemented to validate the proposed detection and self-managing framework. This implementation is to develop a simulator for the elevator system in which the external devices are implemented as software objects rather real devices. The software architecture for self-managing external devices of the elevator system is designed as a layered architecture structured with the service layer and self-management layers [Mane10]. The service layer has application objects for the elevator system, whereas the self-management layer has objects – detection, planning, and execution modules - related to detection and self-management of device failures.

6. Conclusions

This paper has described an approach to establishing a framework for detection and self-management of failed external devices in embedded software systems. This paper has developed the detection framework based on the external device types in embedded software systems. The communication patterns between external devices and the system are used to detect failure or fault of external devices. The self-managing framework handles external device failures from the software perspectives. The proposed approach in this paper has been applied to the use case modeling of the elevator system.

This paper can have further research. This research has been applied to three embedded software systems - elevator system, cruise control system and microwave system. This research may need to other embedded software systems so that we enable to identify more device types. Also this paper does not include the relationship between an external device and its sensor. An external device may have its sensor that is used to monitor the device. When an external device fails, the failure may be in the device or its sensor. This relationship needs to be addressed in our future research.

References

[Cheng06] Cheng, S. W., Garlan, D., and Schmerl, B., "Architecture-based self-adaptation in the presence of multiple objectives," 2006 international workshop on Self-adaptation and self-managing systems, Shanghai, China, 2006.

[Coyle04] Coyle, E. A., Maguire, L. P., and McGinnity, T. M., "Self-repair of embedded systems," Engineering Applications of Artificial Intelligence, Vol. 17, Issue 1, 2004, pp 1-9.

[Finne08] Finne, N., Eriksson, J., Dunkels, A., and Voigt, T., "Experiences from two sensor network deployments: self-monitoring and self-configuration keys to success," 6th international conference on Wired/wireless internet communications, Tampere, Finland, 2008.

[Gomaa00] Gomaa, H., "Designing Concurrent, Distributed, and Real-Time Applications with UML," Addison-Wesley, 2000.

[IBM06] IBM, "An architectural blueprint for autonomic computing," White Paper. June 2006.

[Mane10] Mane P., "Self-Management of External Device Failures in Embedded Software Systems," Master Thesis, CS/Texas Tech University, 2010.

[Pinheiro07] Pinheiro, E., Weber, W.-D., and Barroso, L. A., "Failure trends in a large disk drive population," 5th USENIX conference on File and Storage Technologies, San Jose, CA, 2007.

[Roy07] Roy, P. V., "Self Management and the Future of Software Design," Electronic Notes in Theoretical Computer Science, 1822007, June, 2007, pp 201-217.

[Wikipedia10] Wikipedia, "Self-Monitoring, Analysis, and Reporting Technology (S.M.A.R.T)," Free encyclopedia, 2010.

Towards Modeling and Validating Analysis Processes for Software Adaptation

Xiangping Chen
National Engineering Research Center of Digital Life,
Institute of Advanced Technology, Sun Yat-sen University,
Guangzhou, 510006, China
Key Laboratory of High Confidence Software Technologies (Peking University), Ministry of Education,
Beijing, 100871, China
Chenxp8@mail.sysu.edu.cn

Gang Huang
Key Laboratory of High Confidence Software Technologies, Ministry of Education,
School of Electronics Engineering and Computer Science,
Peking University,
Beijing, 100871, China
Huanggang@sei.pku.edu.cn

Lingshuang Shao
Computer School of Wuhan University,
Wuhan, 430072, China
shaolsh@gmail.com

***Abstract*— With the complexity and the dynamic analysis requirements during software adaptation for software in the open and dynamic environment, the software analysis requires integrating multiple analysis methods and controlling the execution of analysis methods in specific sequence. As a result, the analysis process is hard to be communicated to its designers, implementers and users without an explicit model. In addition, if dependent methods are not included or encounter runtime failure, the data dependency and control dependency between analysis methods may cause correctness problem. In this paper, we proposed a domain-specific language for modeling the analysis process using subset of notations from BPMN. In order to cope with the dependency problem in model and implementation level, the dependency relationships are divided into static dependency and dynamic dependency. Dependency identification and matching algorithm is provided to assure the static dependency in the analysis process model. For the dynamic dependency, we provide a service to detect the analysis failure caused by dynamic dependency. We model and validate two analysis processes supporting software adaptation with performance and reliability concerns, and apply the analysis processes in analyzing a running system ECPerf.**

Keywords- software analysis; analysis process, process validation, analysis process modeling

I. INTRODUCTION

Analysis is the process of breaking a complex topic or substance into smaller parts to gain a better understanding of it. Software analysis technology is widely used during software lifecycle, and generates descriptive, predictive or prescriptive analysis results regarding to the system consistency, correctness, quality attributes and so on.

More and more software systems are running in the internet which is open and dynamic. In order to provide continues and high quality services in the internet, software may change the structure or behavior according to the changing user requirements and environment [1]. The increasing complexity of the software system brings challenges to the software analysis.

Software analyses play an important role during software adaptation. A general adaptation process contains four stages: monitoring, analysis, planning and execution. In the monitoring stage, the system model generated by monitoring system is usually low-level model containing raw data of different system aspects. These models are hard to be used without a transformation to high-level model or domain-specific model. In the analysis and planning stages, the system designer usually employs analysis methods to acquire status of the system, possible adaptation strategies and change impact, so as to gain a better understanding for decision making. For the information of different kinds: descriptive, predictive or prescriptive, usually more than one method are needed. Even in the execution stage, analyses of platform specific features and execution strategies are usually required for assurance of adaptation correctness.

During the software adaptation, multiple analysis methods are used and carried out in specific sequence. For example, an analysis method for adaptation planning may be used after the analysis methods for system quality attributes. In addition, the analysis results may affect the choice of analysis methods afterward. For example, the quality attributes acquired in the analysis stage may affect the choice of adaptation planning methods. If there is security problem, the designer may choose to analyze the possibility of adding fault-tolerant ability. If the system's performance is too low to fulfill stakeholders' requirement, the designer may choose to see possible reconfiguration with higher performance.

The analysis requirements for software adaptation bring new challenges. First, the analysis process is complex that it contains several related analysis methods. The set of analysis methods is chosen according to the system status and these methods should be carried out in specific sequence. The analysis process is repeatedly executed by hand. As a result, the execution process is error prone and hard for reuse. Second, the assurance of the analysis process correctness is hard because of complex relationships between analysis methods. There are two kinds of dependency relationship: the data dependency and control dependency. Data dependency means that an analysis method's input relies on the output of another analysis method. Control dependency happens when an analysis method's output decides whether another analysis method is executed. Thus, if an analysis method relies on the analysis result produced by an analysis method which is not included in the process, or the relied analysis method encounters runtime failure and can not

provide proper result for other analysis methods, the analysis process could not generate correct results or even could not finish execution.

Recent works on the integration of analysis methods have proposed methods and framework to ease the integration of analysis methods. These works focus on extracting proper model for analysis, controlling external analysis tool to carry out analysis task, and synthesizing analysis results back to the core model. Among these works, ADD [7] provides guidelines and standard interface for integration of analysis input, execution and output; XTEAM [5] and KAMI [3] focus on providing facilities for input adaptation and tool execution; our ongoing work focus on the integration on the synthesis of analysis results. However, these works assume that the analysis methods can be integrated independently and used together to finish an analysis process. The relationships between analysis methods during an analysis process are not considered.

In this paper, we propose a process based method to model and validate analysis process. We propose a domain-specific language for modeling the analysis process based on notations from BPMN. The modeling language considers the sequence relationship and dependency relationship between analysis methods in the process. In order to cope with the dependency problem in model and implementation level, the dependency relationships are divided into static dependency and dynamic dependency. Based on the analysis process model, dependency identification and matching are provided to assure the correctness of analysis execution. To ensure the static dependency could be guaranteed, the require input of each analysis method is evaluated. For the dynamic dependency, we provide a service to detect the analysis failure caused by dynamic dependency which is expressed by the elements with runtime data dependency definition, and this detection service is added to the process.

The remainder of the paper is organized as follows: Section 2 illustrates the modeling and validation requirement of analysis process. Section 3 and 4 introduce the modeling and validation method for analysis process. Going back to our example, Section 5 presents the use of our method to model and validate analysis process for software adaptation. Finally Section 6 discusses some related works before Section 7 concludes our work.

II. ILLUSTRATIVE EXAMPLE

In this section, we will introduce an analysis process including three analysis methods to show the modeling and validation requirement.

This example is a software adaptation analysis process, which first appeared in the work on performance adaptation [9]. The process includes three analysis methods for monitoring, analyzing and planning respectively: behavior model generation method COMPAS (Component Performance Assurance Solutions) [2], performance related pattern detection method [9], pattern-based reconfiguration method [9]. The user first uses the COMPAS analysis method to generate the behavior model based on runtime information contained in a software model from the monitoring stage. Then, the user uses an analysis method to detect Fine-grained Remote Calls structure in the software model. If instance of the bad structure pattern is found, the user will execute pattern based reconfiguration method to generate an adaptation plan. If bad structure does not exist in the software model, the analysis process is finished without planning.

In the example, the pattern detection method relies on the behavior model generated from the COMPAS method. Whether the planning method will be executed is decided by the result of pattern based detection method. The data dependency and control dependency may cause problem in design time and runtime. At design time, if the user first chooses the pattern based detection method instead of the COMPAS method, the detection method could not be executed without behavior model. We add the COMPAS method in the analysis process to solve the static dependency problem. However, dynamic dependencies may happen. For example, the COMPAS analyzer will not generate any result when it encounters runtime error. In this situation, all the analysis methods and control switchers rely on it will be affected. The analysis process will stop executing, or generates fault results or useless results.

Besides modeling the analysis process, we need to validate the process in order to correctly analyze the system.

III. MODELING ANALYSIS PROCESS

Our approach starts from the modeling of analysis process. The activities in the process are all of the type analysis method. An analysis process can be viewed as a group of analysis methods.

In order to simplify the analysis process modeling, we build our modeling method based on existing works on the integration of analysis methods. The integration of the analysis methods can be divided into three aspects: the integration of the input, the integration of the execution and the integration of the output. We assume that the analysis methods are already integrated that its input is provided by the software model through input adaptation; its tool is integrated and can be invocated; the analysis results are synthesized back into the software model.

Data proceeding and control logic are two important consideration in the molding of analysis process.

From the data level, all the analysis methods in an analysis process have the same analysis object: the software system. The analyzed system is usually abstracted and saved as model. The analysis results are the features of software system and can be modeled as elements or attributes of elements in the software model. As a result, the data flow in the analysis process includes the core software model and the analysis results. The analysis methods are organized as a star configuration.

In order to simplify the data processing modeling, we use our former work [11] on the integration of analysis results to update the core software model. Based on the framework, the software model is updating during the analysis execution. After an analysis method is executed, the software model will be extended with its results. For each analysis method, its required input can be provided by the software model and the analysis results produced of the analysis method executed

before. As a result, the analysis result integration framework maintains the models and manages the data processing in the analysis process. Based on the integration framework, modelers do not need to model the data processing in the analysis process; they can concentrate on modeling the control logic of analysis process.

The complex control logic of a process can be described using three kinds of structure: sequence structure, decision structure and repetition structure. However, an analysis method is seldom repeatedly executed during an analysis process. Because analysis is a way to generate software features from the software model, the analysis behavior does not affect the model. For a specific analysis method and a software model, the analysis result is the same if the analysis method is executed repeatedly. Thus, repetition structure is not needed in analysis process. It should be noted that an analysis method may be carried out more than once during an analysis process. This is because an analysis method can be applied to analyze different part of software model especially analysis results generated by other analysis methods.

Besides the sequence structure and decision structure, parallel structure is supported in the analysis process modeling for performance consideration. Because analysis methods are usually provided and implemented by different vendors, their analyzers may be running in different nodes. If some of the analysis tasks are time consuming and can be carried out by the analyzers in different node, the designer can model these analysis methods in a parallel structure.

We use a subset of notation of BPMN to model the analysis process. The modeling element types are listed in table 1. The start event and end event are used in the beginning and ending of the analysis process. Inter Event is only used to show possible exception in the process. A Task is mapped to an integrated analysis method which is the only kind of atomic activity in the analysis process. AND Gateway is used to describe the set of analysis methods which should be carried out in parallel. XOR Gateway and Sequence Flow are used to describe the sequence structure and decision structure.

Table 1 modeling elements

Element	Description
Start Event	The Start Event indicates where a particular process will start.
Inter Event	Intermediate Events occur between a Start Event and an End Event and will affect the flow of the process.
End Event	The End Event indicates where a process will end.
Task	A task is an atomic activity within the process. It represents work to be performed. A task will be activated when one of its incoming sequence flows is triggered and will trigger all its outgoing sequence flows when it is finished.
AND Gateway	An AND Gateway will trigger all outgoing flows, when all incoming flows are triggered.
XOR Gateway	An XOR Gateway will trigger one of its outgoing flows when one of its incoming flows is triggered.
Sub-process	Sub-processes are used to support hierarchy. A Sub-process contains its own Start and End Event.
Sequence Flow	The Sequence Flow shows the order in which activities will be performed in a Process.

Our analysis process modeling language uses the notation of BPMN. Designer can use the BPMN visualization modeling tool and implementation generation tool to model the analysis process and get the executable process described using BPEL regarding semantic mapping between BPMN and BPEL.

IV. VALIDATING ANALYSIS PROCESS

There are two kinds of dependency: data dependency and control dependency. To ensure the static dependency could be guaranteed, the dependency occurs during the analysis process is detected and the depended analysis method is automatically added to the process. For the dynamic dependency, we provide a service to detect the analysis failure caused by dynamic dependency which is expressed by the elements with runtime data dependency definition, and this detection service is added to the process afterwards

A. Dependency Definition

In order to validate the dependency relationship, we extend the integration definition in the analysis result integration framework. Fig 1 shows the extended meta-model, the integration model is extended with the element Relied Element. Analysis method integrator can specify the relied elements and their validation rules. The default validation rule for a relied element is that it should not be null. For the Gateway element in the analysis process, the switcher usually relies on some elements for decision making. The process modeler can define an integration model for a Gateway element. The integration model only includes a list of relied elements. We detect static dependency problem and validate the required input model at runtime based on the integration models of analysis methods and Gateway elements in the process.

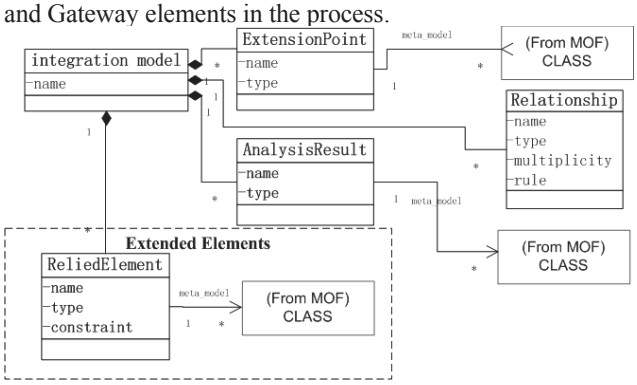


Fig 1. Extended Meta-model of Integration Definition

B. Static Dependency Detection

There are two kinds of dependency relationship between analysis methods: data dependency and control dependency. An analysis process is incorrect if there is any analysis method or control switcher who relies on one or more analysis methods which could not generate needed analysis

result for this method or control switcher. The dependency problem may occur when (1) the relied analysis method is not included in the analysis process; (2) the relied analysis method is included in the analysis process, but could be executed after the analysis method or control switcher in some situation; (3) the relied analysis method is included in the analysis process and could be executed before the analysis method or control switcher in any situation, but the analysis method has dependency problem.

For an analysis method am_0 in the process model, it does not have dependency problem if

(1) All the analysis methods which am_0 relies on are included in the process model.

(2) All the process instances will carry out am_0 's dependent analysis methods before am_0 。

Because analysis methods do not affect each other directly even if they are in the same analysis process. Their dependency relationships are built based on the relationship between their input and output. $\mathrm{AR}_{\mathrm{am0\_dependent}}$ is the set of analysis results which analysis method am_0 relies on; $\mathrm{AM}_{\mathrm{am0\_dependable}}$ is the set of analysis methods whose results can be used by analysis method am_0 ; $\mathrm{AR}_{\mathrm{am0\_dependable}}$ is the set of analysis results produced by all the analysis methods in $\mathrm{AM}_{\mathrm{am0\_depedable}}$. An analysis process do not have static dependency problem when all the analysis methods in the process rely on the analysis results which can be generated before its execution in any process instances. The constraint can be expressed in OCL as below:

$$P.task \rightarrow forall\{\mathrm{am}_0 \mid (\mathrm{AR}_{\mathrm{am0\_depend\ able}} \cup CoreModel) \supseteq \mathrm{AR}_{\mathrm{am0\_depend\ ent}}\} = P.task$$

For an analysis method am_0 in the analysis process, its dependent analysis results are defined by the integrator. We developed an algorithm to find out the dependable analysis methods, the union of all the results produced by the dependable analysis methods is $\mathrm{AR}_{\mathrm{am0\_dependable}}$.

Input: process model P, analysis method am_0 , the starting point and ending point in the process model P $point_{start}$ and $point_{end}$;

Variables: $\mathrm{AM}_{\mathrm{dependable}}$, $\mathrm{AM}_{\mathrm{dep\_temp}}$ and $\mathrm{AM}_{\mathrm{temp}}$ are sets of analysis methods, SF_{temp} is a set of sequence flow, and $checkpoint$ is a reference to an element in the process, e_{start} and e_{end} are the start event and the end event of process P;

FindAMDepedent (P, am_0 , $point_{start}$, $point_{end}$)

Initialization: $\mathrm{AM}_{\mathrm{dependable}} = \phi$, $checkpoint = po\mathrm{int}_{start}$, $\mathrm{SF}_{\mathrm{temp}} = checkpoint.outgoing$;

1. if $checkpoint = po\mathrm{int}_{start}$ and $AM_{temp}.size = 1$, then $checkpoint = AM_{temp} \rightarrow first()$
   $\mathrm{AM}_{\mathrm{temp}} = checkpoint.outgoing$
2. if $checkpoint = po\mathrm{int}_{end}$, goto (9)
3. if $checkpoint = am_0$, goto (8)
4. if $checkpoint$ is of the type Task and $AM_{temp}.size = 1$, then
   $\mathrm{AM}_{\mathrm{dependable}} = \mathrm{AM}_{\mathrm{dependable}} \cup \{checkpoint\}$,
   $checkpoint = AM_{temp} \rightarrow first()$,
   $\mathrm{SF}_{\mathrm{temp}} = checkpo\ \mathrm{int}.outgoing$;
5. if $checkpoint$ is of the type Gateway, then
   for *am* in AM_{temp}
   $\mathrm{AM}_{\mathrm{dep\_temp}}$ = FindAMDepedent(P, am_0 , *am*, $checkpoint.jointpoint$)
   If $\mathrm{AM}_{\mathrm{dep\_temp}} \neq \phi$, then
   $\mathrm{AM}_{\mathrm{dependable}} = \mathrm{AM}_{\mathrm{dependable}} \cup \mathrm{AM}_{\mathrm{dep\_temp}}$
   Goto (9);
   endfor
   $checkpo\mathrm{int} = checkpoint.jointpoint$
   if $checkpoint$ is of the type AND Gateway and $\mathrm{AM}_{\mathrm{temp}} = \phi$, then
   goto(6)
6. $\mathrm{AM}_{\mathrm{dep\_temp}} = \phi$
   for *am* in AM_{temp}
   if *am* is of the type XOR Gateway, then
   $\mathrm{AM}_{\mathrm{dep\_temp}} = \mathrm{AM}_{\mathrm{dep\_temp}} \cup am.jointpoint$
   if $am \neq checkpoint.jointpoint$ and am is of the type Task
   $\mathrm{AM}_{\mathrm{dep\_temp}} = \mathrm{AM}_{\mathrm{dep\_temp}} \cup am.outgoing$
   $\mathrm{AM}_{\mathrm{dependable}} = \mathrm{AM}_{\mathrm{dependable}} \cup \{am\}$
   endfor
   if $\mathrm{AM}_{\mathrm{dep\_temp}} \neq \phi$, then
   $AM_{temp} = \mathrm{AM}_{\mathrm{dep\_temp}}$
   goto (6)
7. goto (1)
8. return $\mathrm{AM}_{\mathrm{dependable}}$
9. return ϕ

Using this algorithm, a process can be validated. If for each analysis method in the process, its relied input can be provided by the core model and the analysis results of its dependable analysis methods, this analysis process do not have static dependency problem.

C. Runtime Validation of Required Input Model

During the execution of an analysis process, runtime exception may affect the analysis execution and produce meaningless analysis results. The runtime exception may be caused by missing input from the core software model, exceptions from analysis tool execution, and so on. The fault or meaningless analysis results may affect other analysis methods which rely on these results. Thus, if this kind of fault can be detected at runtime, the process can be stopped so as to save time and avoid the misleading analysis results.

The runtime validation is the process to make sure that the input constraints are not violated. The validation is based on the dependency definition. The validation objects are the elements specified as relied elements in the integration model. There is a default rule that the required input element should not be null. This rule can be described using OCL:

$$integrationmodel.reliedElement \rightarrow oclIsUndefined() = \text{false}$$

Besides this rule, analysis method integrator can define rules as constraints for the *reliedElement*.

The runtime validation of required input model can be abstracted as a special analysis method in the analysis process. Its analysis result is used to decide whether the process will continue or not. For each analysis methods or switchers with relied elements, we add a runtime validation method before it. We develop and integrate the validate tool in the integration framework. Then, the analysis process can detect validation of constraints at runtime.

V. Case Study

In this chapter, we use our approach to model and validate the analysis process for the adaptation of a component-based system, the ECPerf system [6]. The ECPerf system is provided by Sun Microsystems, which has been adopted as a standard benchmark application for JEE systems. It is an online "Just-in-Time" manufacturing system. We used the SM@RT [10] monitoring platform to extract data from the running ECPerf system and to visualize the corresponding software model. We used the analysis method integration framework [11] to integrate the related analysis methods. For space limit, we focus on the modeling and validation of analysis process, detail of analysis methods integration is not included in this paper.

A. Peformance Adaption

In section2, we introduce an analysis process for performance adaptation. For a maintainer who is not familiar with the analysis methods, he does not know that the pattern detection method relies on the behavior model which is not provided by the monitoring system. He my model an analysis process containing two steps: detect performance related bad pattern, and reconfigure the software if bad structure exists, as shown in Fig 2.

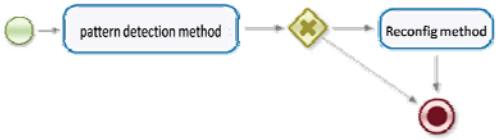


Fig 2 Analysis Process for Performance Adaptation

We use our static dependency detection algorithm to validate this analysis process. Because pattern detection method is the first method in the process, it can only rely on the core software model. In the integration definition of pattern detection method, two relied elements are defined: software configuration element *Config* and the behavior model element *SequenceDialog*. The configuration model is included in the software model. However, the software model only contains a list of invocation records. The software model does not contain any element of the type *SequenceDialog*. There is static dependency problem in the process. If an analysis method COMPAS is added before the pattern detection method, as shown in Fig 3, it can generate behavior model based on the invocation record in the core model. This analysis process does not have dependency problem and can be used for performance adaptation.

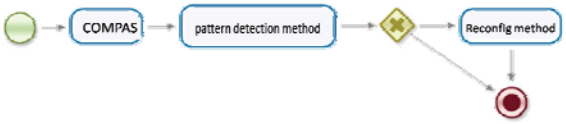


Fig 3 Analysis Process for Performance Adaptation

B. Fault-torelant Style based Software Reconfiguration

Reliability is one of the most important quality attributes for online services. Because reliability is important, the maintainer chooses two reliability analysis methods SBAR [12] and ABRAM [14] to analyze the runtime software model. If the value of any reliability result is less than 90%, the maintainer will choose a planning method ASPIRE [13] to generate adaptation plan, and then use the SBAR and ABRAM method again to analyze the planning result to make sure that the reliability will be improved. During the analysis process, the maintainer may stop analyzing if the system reliability is acceptable. We use our modeling language to describe this analysis process, as shown in Fig 4.

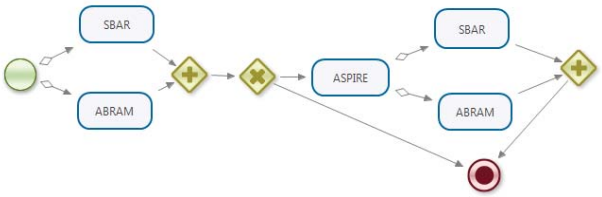


Fig 4 Analysis Process of Style-based Software Adaptation

In this example, the SBAR method and ABRAM method is used twice with different input. The XOR Gateway before

ASPIRE method relies on the reliability analysis results: *SBARresult* and *ABRAMresult*. The constraints for these two elements are the same that the value of the element should be in the range of 0 to 1. A validation analyzer is added before the gateway in the analysis process. This analyzer will check both the *SBARresult* and *ABRAMresult* when the SBAR and ABRAM method finishes execution.

When we apply this analysis process in analyzing the software model of ECPerf system, the process always stopped and through an exception before the XOR gateway. Because we re-implement ABRAM analyzer using the model transformation language ATL with too many nested statements, the analyzer will be short of memory when analyzing software models with more than 10 components. When analyzing the ECPerf software model, the ABRAM analyzer cannot return the result *ABRAMresult* until timeout. As a result, the runtime validation carried out before the XOR gateway will find that the required input *ABRAMresult* violates its constraint. The runtime validation is also helpful for us to detect improper usage of analysis method.

VI. Related Work

The works on the Enterprise Application Integration (EAI) can be applied to integrate analysis tools. However, EAI most focuses on the translation the outputs of some application into inputs for other application, and the invocation of application interfaces [15]. The relationships between integrated analysis methods are not considered.

Many approaches and frameworks have been developed for complex analysis of software models. Some approaches are proposed to facilitate the use of multiple analysis methods during software design. ADD [7] and FADF [6] provide standard interfaces for integration of analysis methods. These works notice the requirement for the analysis process, but do not provide support for process modeling. In addition, these works simplify the relationship between analysis methods as data exchanging and leave the user to guarantee the process correctness.

Several research projects propose frameworks [3, 4, 5, 6] to simplify the integration of existing model-based analysis methods. XTEAM [5] is a framework focusing on solving the mismatch between SA model and required input of analysis methods. KAMI [3] is a framework for run-time model adaptation. These frameworks provide facilities to simplify the development of model interpreters and model updaters, respectively. DUALLY [4] is an automated framework that allows architectural languages and tools interoperability. It provides the infrastructure for (semi) automatic generation of the weaving model to integrate analysis abilities provided by ADLs. However, these frameworks assume that analysis methods are independent and do not consider dependencies of the analysis methods. Our framework focuses on the modeling and validation of analysis process, and is build based on some of these works.

VII. Conclusion

This paper introduces an approach to model and validate the analysis process for software adaptation. We believe that an explicit model for analysis process can enhance the reuse of analysis process and save effect for analysis execution. Considering the dependency relationship in analysis methods, our method provide an algorithm to detect static dependency problem in the analysis process model and an analysis method to validate required input model at runtime, so as to avoid correctness problem caused by dependency .

VIII. ACKNOWLEDGMENTS

This effort is sponsored by the Joint Funds of NSFC-Guangdong under Grant No. U0835004, the National Natural Science Foundation of China under Grant No. 60873060, 60933003, and 61003072.

References

[1] Oreizy P, Gorlick MM, Taylor RN. An Architecture-based Approach to Self-Adaptive Software. IEEE Intelligent Systems.1999, 14(3): 54-62.

[2] Parsons T. Automatic Detection of Performance Design and Deployment Anti-patterns in Component Based Enterprise Systems. Ph.D. Thesis. 2007: University College Dublin.

[3] Epifani I, Ghezzi C, Mirandola R and Tamburrelli G. Model evolution by run-time parameter adaptation. Proceedings of the 2009 IEEE 31st International Conference on Software Engineering. IEEE Computer Society Washington, DC, USA, 2009:111-121

[4] Malavolta I, Muccini H, Pelliccione P, Tamburri DA. Providing architectural languages and tools interoperability through model transformation technologies. IEEE Transactions on Software Engineering. 2009, 36(1): 119-140.

[5] Edwards G. and Medvidovic N. A methodology and framework for creating domain-specific development infrastructures. Proceedings of the 23rd IEEE ACM International Conference on Automated Software Engineering. 2008: 168-177

[6] Dai L and Cooper K. Modeling and Analysis of Non-functional Requirements as Aspects in a UML Based Software Architecture Design. Proc. 6th Int'l Conf. on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing and 1st ACIS Int'l Workshop on Self-Assembling Wireless Network (SNPD/SAWN'05), 2005.

[7] L. Bass, P. Clements, and R. Kazman, Software Architecture in Practice, 2nd ed. Addison-Wesley, 2003.

[8] ECperf Home Page. ECperf http://www.spec.org/jAppServer2001/docs/DesignDocument.html#S1

[9] Ling Lan, Gang Huang, Weihu Wang, Hong Mei. A Middleware-based Approach to Model Refactoring at Runtime. 14th Asia-Pacific Software Engineering Conference (APSEC'07), 2007: pp.246-253,

[10] Gang Huang, Hui Song, Hong Mei. SM@RT: Applying Architecture-Based Runtime Management into Internetware Systems. International Journal of Software and Informatics, 2009, 3(4):439-464

[11] Xiangping Chen,Gang Huang,Franck Chauvel,Yanchun Sun,Hong Mei. Integrating MOF-Compliant Analysis Results. International Journal of Software and Informatics, 2010,4(4):383~400

[12] Yacoub S, Cukic B, Ammar H. Scenario-Based Reliability Analysis of Component-Based Software. Proceedings of the 10th International Symposium on Software Reliability Engineering. 1999.

[13] Li JG, Chen XP, Huang G, Mei, H and Chauvel, F. Selecting Fault Tolerant Styles for Third-Party Components with Model Checking Support. Proceedings of the 12th International Symposium on Component-Based Software Engineering. 2009: 69-86

[14] Wang WL, Wu Y, Chen MH. An Architecture-based Software Reliability Model. Proceedings of the 1999 Pacific Rim International Symposium on Dependable Computing. 1999, 143-150.

[15] Mickael Clavreul, Olivier Barais, Jean M. Jézéquel. Integrating Legacy Systems with MDE.Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering (2010)： 69-78.

A Reflective Model for Architecting Feedback Control Systems

Filip Křikava
Université Nice Sophia Antipolis
I3S - CNRS UMR 6070, Sophia Antipolis, France
filip.krikava@i3s.unice.fr

Philippe Collet
Université Nice Sophia Antipolis
I3S - CNRS UMR 6070, Sophia Antipolis, France
philippe.collet@unice.fr

***Abstract*—Autonomic Computing aims at realizing computing systems that are able to adapt themselves, but the engineering of such systems in the large is rather a challenging task. It is hard to find an appropriate model that controls the adaptation itself and several loops are likely to be coordinated to avoid unexpected and harmful behaviors. This paper presents an approach and a runtime support to architecture self-adaptive systems, in which each part of the feedback control loop is uniformly and explicitly designed as a first-class adaptive element. Making these elements explicit allows the architect to reason about system modeling and to reuse them. Code generation from the architecture model avoids painful details of low-level system implementation.**

***Keywords*-Autonomic Computing, Model-Driven Engineering, Software Architecture**

I. Introduction

The 24/7 deployment of distributed systems is dramatically increasing the complexity and maintenance costs of software systems. The ability to adapt then becomes crucial for such systems. Autonomic computing is a promising way to organize self-adaptive systems. This approach aims at realizing computing applications that can dynamically adjust themselves to accommodate changing environments and user needs with minimal or no human intervention [1]. The engineering of such systems in the large is rather a challenging task. It is hard to find an appropriate model that controls the adaptation itself [2]. Its implementation being generally integrated in a feedback control loop, developing several adaptive behaviors necessitates to reason on them, at least to obtain an appropriate coordination between them. Moreover, engineering in the large obviously needs some reuse capabilities on loops or more likely, on elements composing the loops [3].

Following the general principle that control loops should be made explicit [4], [5], we present in this paper an approach for engineering self-adaptive systems with an explicit architectural model and its runtime support. In the proposed architecture each part of the feedback control loop is uniformly and explicitly designed as a first-class adaptive element. Making these elements explicit allows the architect to reason about system modeling, while capturing different patterns of interactions and controls (coordination of loops, adaptive monitoring, etc.). By applying a model-driven approach with a framework based on standardized interfaces, loops and loop elements are then likely to be more easily reused. Moreover code generation from the architecture avoids painful details of low-level system implementation.

The remainder of this paper is organized as follows. Section II introduces some background on autonomic control loops and discusses requirements. Our approach is described in Section III, giving details of its principles and architectural model. Section IV illustrates our proposal on an overload scenario of an extensively used distributed batch computing system. The runtime support is also presented, and first experimental results are evaluated. Some related work are discussed in Section V. Finally, we conclude and briefly present future work in Section VI.

II. Engineering Self-Adaptive Systems

Self-adaptive systems are characterized by runtime decisions to control their structure or their behavior and making these decisions autonomously (i.e. without or with minimal interference) while reasoning about their contexts and environments [5]. Self-adaptive software is generally organized around closed feedback loops aiming at adjusting itself during its operation. The Autonomic Computing refers to self-managing characteristics of computing resources that manage themselves given some high-level objectives by adjusting their operation in the face of changing environment and user needs [1].

Making autonomic computing a reality necessitates to put together and evolve results from several research disciplines, from Artificial Intelligence (planning, decision theory, machine learning, agents, etc) to Control Theory and engineering [6], [3]. Moreover numerous challenges are directly related to the software engineering of feedback control loops, to ease the application of other disciplines. Loops have been originally designed following the *sense-plan-act* control decomposition, or refined to make four steps appear, i.e. collect, analyze, decide, act. When engineering such loops, it has been first shown that it is hard to find an appropriate model that controls it [2]. Besides, building the software artifacts around this control model is far from being trivial, as monitoring data must be consistently collected and actions on the adapted systems well coordinated. Finally, overlapping in all concerns of a loop may appear when

several feedback loops need to be deployed, leading to unexpected and potentially harmful behaviors.

Previous work tackle these challenges. IBM proposed a form of standardized approach for the feedback loops with the *Monitor-Analyze-Plan-Execute-Knowledge* (MAPE-K) architecture [1], [7]. This makes explicit the behavior of autonomic managers which activity follows the *M-A-P-E* decomposition using some shared *Knowledge* (indicators, policies, plans...). Several systems and frameworks have been developed according to this principle [8], [9], [10][1], generally aiming at providing an architecture to organize the self-adaptive part. For example, Garlan et al. [9] proposed two-layers framework with an external fixed control loop that can be customized. The controlled and control system are clearly separated and a mapping allows data to be transmitted between them. The control loop is then made explicit but several feedback behaviors cannot be separately designed.

Recently, it has been advocated that control loops and their elements must be explicit to facilitate reasoning about them as well as their reuse [4], [5]. Even if recent work made some advances, with hierarchical MAPE-K coordination [11] or component-based analysis architecture [12], there is, to the best of our knowledge, still the need for an architecture that would provide the right abstractions for rapid prototyping, but yet remain flexible enough to cope with the diversity and dynamic aspects found in current systems. In our vision, such architecture should support i) several kinds of loop element grouping and loop architectures, so that different forms of coordination between loops can be designed together with some sharing between monitoring elements or effectors on the controlled system; ii) adaptive capabilities over the controlled system but also on the control system, so that the architecture would capture adaptations over the monitoring, parameterizations on the models used by the controllers, etc. Moreover, the architecture should be versatile enough to capture the relevant parts of the feedback systems while being as abstracted from technological details as possible. In reusing some generative approach, this would foster reuse while avoiding painful details of low-level system implementation for most parts.

III. Approach

In this section we present the main principles of the proposed approach, together with the model for the architecture.

A. Principles

In our approach, we focus on an externalized self-adaptation that is based on a closed loop feedback control. We express an architecture of such a self-adaptive system in a technologically agnostic model that is centered around the notion of a feedback loop and where each of the loop's

[1]See [5] and [6] for surveys.

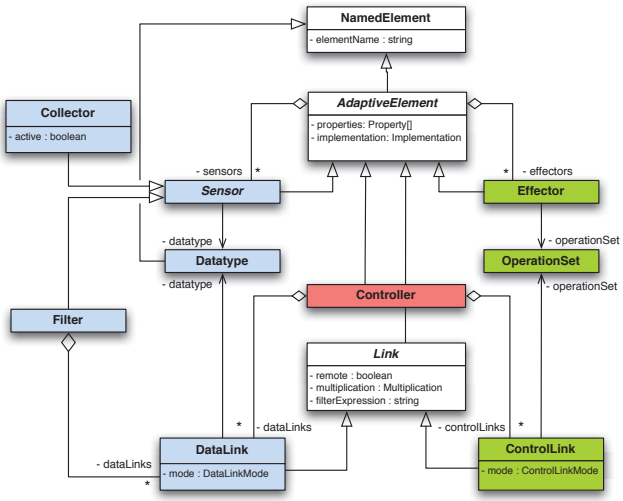


Figure 1: Architectural model.

elements is made explicit [4]. Four main elements represent the main responsibilities in the feedback control. The target system and its context observability is captured in *sensor* elements, the decision making process is represented by *controller* elements and the actual system adaptation is carried out by the *effector*. We also explicitly model the data and control flow in the architecture via *links* that represent the dependencies between the components. Figure 1 presents the resulting architectural model. In the following paragraphs, we detail the different modelling capabilities of our proposal to design the different elements of a loop.

One of the main originality of the model is that it supports reflective self-adaptation. In the model, each element inherits from *adaptive element*, which itself can provide its own sensors and effectors. This feature allows elements to be introspected (meta-data, state) and modified in the very same way as the target controlled system is. One can hierarchically compose not only control on the top of controllers, but also on the top of sensors, effectors and links. The added adaptation then becomes self-adaptable as well, and in an uniform way. This notably distinguishes the model from other model-driven proposals [15]. In this paper, we do not deal with the distribution of loops and loop elements, but in that case, as links can be themselves adaptive, remoting issues like the inevitable delays and network failures could be addressed with our architectural model.

Besides, by using a model to express the adaptation concerns it allows us to support different degrees of separation in respect to the target system. From being completely external, running aside the target system in its own runtime platform, to be completely integrated inside the system using aspect-oriented techniques or direct source code generation. The model is technologically agnostic thus it only captures

the semantic of the adaptation: the definition of the components and with relevant inputs and outputs for data and control flow respectively.

B. System Observation

The monitoring part of the model (left part in fig. 1) is responsible for supplying information about the system into controllers so they can reason about the state of the system and its environment in order to take the appropriate decisions. This information is provided by hierarchically organized sensors elements in a form of a directed acyclic graphs with *data links* connecting them together. There are different types of sensors: collectors and filters.

Collector: The leaf nodes in the hierarchy are called collectors. A collector provides data of a defined *datatype* that are directly gathered from an external entity like various operating system resources, services calls, user preferences, etc. Essentially, it can operate in two modes: active or passive. An *active collector* is responsible for updating itself. This is used in cases where the update is based on some external notifications like a file change, new socket connection, etc. A *passive collector*, on the other hand, waits until it is explicitly requested by an associated link to provide data.

Filter: The other nodes in the graph, which are not leaves, are filters. These sensors are used to aggregate or in some other way process data that are coming from one or more connected sensors. They can be real data filters, stabilization mechanisms, converters, rules inference engines, etc. A typical example of a filter is a noise filter that is used to stabilize the context information that comes from other sensors. Since each filter is therewith an adaptive element, its parameters can be advertised using specific sensors and effectors, thus providing a uniform means to adjust them at runtime.

Each of the data links connects a producing node (a sensor) with consuming node (a filter or a controller) and can be configured *before and at runtime* to be operating in different modes. With *periodic notification or observation* (▶, ◀) a link requests, at a fixed rate, the producer to get data and forwards it to the consumer. With *reactive notification* (▷) a link is explicitly requested by a producer to forward given data to the connected consumer. *Reactive observation* (◁) is the opposite of the notification, a link is explicitly requested by a consumer to get data from the connected producer.

When all elements of the monitoring part are connected, the model can easily be checked against the well-formedness of the resulting data flow (non-interrupting, correct typing). As the model relies on strong typing, the controller (through its data link) and its sensors must have compatible data types in order to be connected. The resulting data flow itself logically originates in collectors and terminates in controllers, however, the actual data transfer that triggers the adaptation is driven by either active collectors or the periodic links. These are the only entities in the system that are actively running, the rest of the system is event based.

The links themselves supports adaptation so that their transmission properties, such as the notification or observing periods, are advertised and can be used by other loops. For example it can be used to model some adaptive monitoring or to handle timeouts on remote links.

C. Decision Making

The decision making part is in the model represented by the controller element. Essentially, the idea behind the decision making is to choose an appropriate action among the set of all permissible actions based on the observed state of the subjected system. There are many different kinds of controllers that can be used for the decision making process. A good overview of the some strategies is given in [13], but different kinds of controllers can be designed, for example supporting a decomposition between the analysis and planning phases, like in the MAPE-K decomposition [1]. In this paper we are not concerned by the actual design of the adaptation behavior itself, but rather we focus on supporting the architecture of the system as a whole.

D. System Alteration

The actual system modification is carried out by effectors that are orchestrated by controllers based on the actual decisions. An effector encapsulates a set of operations and provides them to the controlling elements via *control links*. An operation is a named action that can take an arbitrary number of arguments. Similarly to the data links, control links also use strong typing.

IV. Illustration

In this section we illustrate our approach with a simplified feedback based overload control system in the high-throughput computing domain.

A. Scenario Overview

We consider an environment made for executing compute-intensive scientific workflows [14] with the Condor infrastructure [16]. Condor is a well established distributed batch computing system that has been used extensively in both academia and industry. We use a typical Condor deployment with one scheduler, a *schedd* agent, that is responsible for managing user submitted jobs and mapping them onto a set of resources where the actual execution is performed. Condor default support for executing workflow is provided by DAGMan (Directed Acyclic Graph Manager) execution engine, which acts as job meta-scheduler on the top of Condor batch system. Each workflow execution starts a new instance of the *DAGMan* that carries out the actual submission of the tasks into the schedd agent.

Since especially scientific workflows tend to be rather large, containing many computer-intensive tasks, the scheduler can easily become overloaded, as the more tasks it has to handle the more resources it uses. The default behavior of the schedd is to accept all valid submission requests regardless the current state of the system. There are configuration options for both schedd and DAGMan, but they are static and do not take into account the current state of the system nor the number and nature of workflow executions varying over time.

B. Overload Control

In order to maintain a certain utilization of the system and prevent its overload, we design a basic controller that will be integrated in our illustrative self-adaptive architecture. The control maintains a certain number N^* of jobs in the queue. There is a configuration option in DAGMan that controls the number of seconds it waits before submitting a task. By making this option reread at before each submission we can impose an adjustable delay d for each client. We denote m the number of clients representing the DAGMan instances at some sample time t. Each client is submitting at rate $\lambda_i = \frac{1}{d}$ therefore the total arrival rate at t is $\lambda = \sum_{i=1}^{m} \frac{1}{d} = \frac{m}{d}$. The control optimizes the utilization of the queue $\rho = \frac{\lambda}{\mu}$ depending on the number of jobs N in the queue where μ is service rate. We use three cases for the state of N: if $N = N^*$ then the buffer is ideally filled so we only maintain the $\lambda = \mu$; if it is less, we linearly increase the arrival rate and when it is more we vigorously decrease it all the way to 0 shall $N = N_c$. The $N_c > N^*$ denotes some critical number of jobs in the queue that must not be reached.

$$\rho(N) = \begin{cases} \rho_0 + \frac{N(1-\rho_0)}{N^*} & \text{for } 0 < N < N^*, \rho_0 > 1 \\ 1 & \text{for } N = N^* \\ \alpha(N - N_c)^p & \text{for } N > N^*, \text{ where } \alpha = \frac{1}{(N^* - N_c)^p} \end{cases}$$

From the utilization and total arrival rate we can derive the target delay d:

$$\rho(N) = \frac{\lambda}{\mu} = \frac{\frac{m}{d}}{\mu} \qquad d = \frac{m}{\rho(N)\mu}$$

The ρ_0 denotes the maximum growth rate allowed in the system.

C. Architecture

In figure 2 we present the resulting architecture using a graphical domain specific language. To increase readability, the names of the implementation classes have been added to the model while removing the names of links.

The first controller (1) is responsible for the submission rate control elaborated above. It thus requires three inputs: the current number of jobs N provided by the `queueStats` collector and further stabilized by a moving average filter, an information about the current service rate μ that is coming from the `serviceRate` collector also stabilized, and the number of clients m is obtained from the `dagmanCounter` collector. The control output is linked to the `dagmanDelayFile` effector which simply writes the given delay to the appropriate file to make it available to the running DAGMans. The `queueStats` collector internally executes the `condor_q` command and parses its output. However, the more jobs are queued the longer it takes to execute and the more system resources it uses. In order to control it, we make this monitoring part self-adaptive as well by adding the controller (2) that adapts the trigger period of the link towards the controller (1) based on the average time that it takes to execute. Using again the self-adaptive capabilities of each adaptive element, we make the properties of the model in controller (1) i.e. the target number of jobs in the queue N^* and the critical number of jobs N_c, changeable over time. In our simplified case, it is only dependent on the system memory. The controller (3), running at a slower pace, is responsible for controlling these properties.

Finally, in Condor there is a `condor_master` daemon responsible for keeping all the rest of the Condor daemons running. If it crashes, the pool looses its managing authority and has to be restarted. In our model, this can be done easily by adding a controller that gets its input from a process heartbeat sensor and start the process if it is not running anymore. Similarly to the master daemon, it uses an exponential back off delay before each attempt. While not being coordinated with other loops in this simplified version, several behaviors are currently added and coordinated with our approach. They can then be used to provide high-level indicators or notifications to system administrators. The overall system shows some relevant capabilities of our model to explicitly design various feedback control loops and self-adaptive behaviors on loop elements themselves. Some potentially reusable parts are also shown, with monitoring filters or loop patterns (adaptive monitoring, parameterization of the controller).

D. Implementation

Our current runtime support is based on the Equinox OSGi framework[2]. In this runtime each instance of an element from the model, including links, is registered as an OSGi service. For each dependency a proxy is generated. Upon request, it consults the OSGi service registry to locate the concrete instances. This allows us to employ the whiteboard pattern [17] for handling the data and control flow in the system. For example, instead of registering a listener for each reactive link to the respective elements, the link itself is registered as an OSGi service with appropriate service properties, so that the proxy can discover it and call when a notification or observation is needed.

[2]http://www.eclipse.org/equinox/

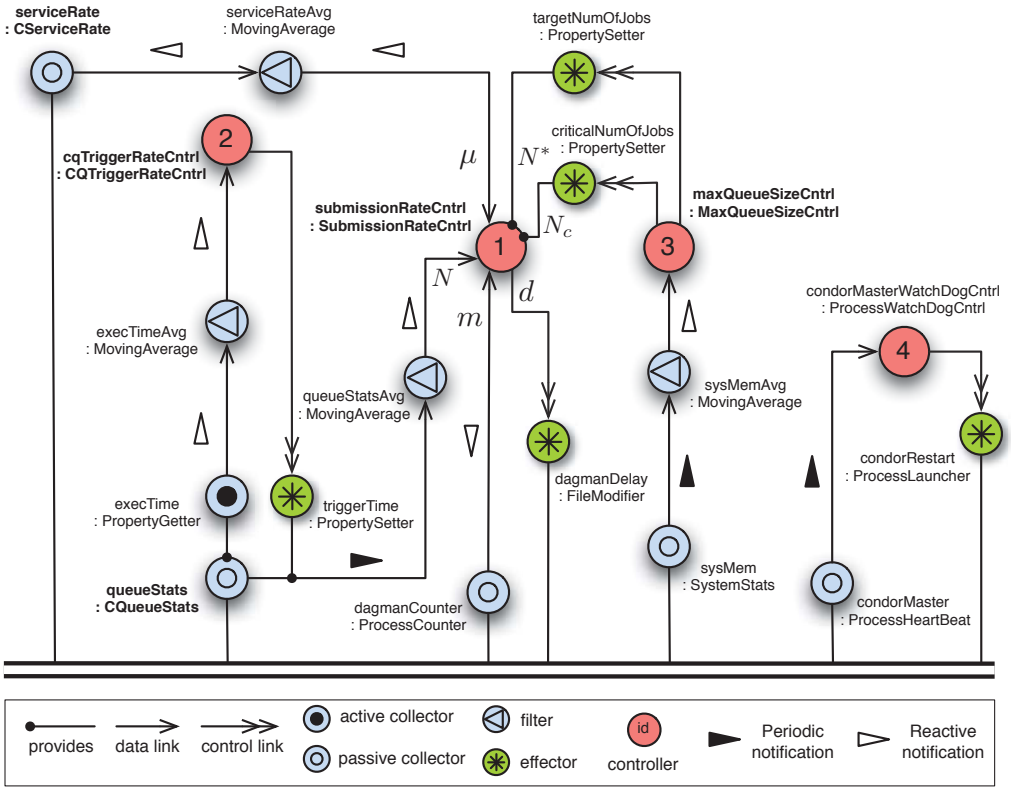


Figure 2: Architecture of the WMS overload scenario. The names of the links are omitted. Each element also contains its corresponding implementation type. In case of controller (1) also the individual inputs and outputs are marked.

In order to turning the architecture into a running system, we first need to formally describe the model of the architecture from the fig. 2. We use a Domain Specific Language (*not shown in this paper*) that defines both the type model and the concrete instances together with values for all required properties and parameters needed at runtime. Additionally we need to provide an implementation for all the custom sensors, controllers and effectors (see elements marked bold in the figure 2). The rest are general reusable elements provided by our framework for the runtime support.

Listing 1 shows an excerpt of a custom controller. This generated Java class skeleton is used as a delegate for the `DelegatingController` provided by the framework. This controller is responsible for handling the observation and notification of the data and control links and providing all the necessary inputs for the annotated control method of the delegate.

```
public class SubmissionRateController {
...
  @ControlMethod
  public void control(
      @DataInput("queueStats") int queueStats,
      @DataInput("serviceRate") float serviceRate,
      @DataInput("dagmanCounter") int dagmanCounter,
      @ControlOutput("writer") FileManipulation writer) {
    ...
  }

  @ProvidedEffector ("targetNumOfJobs")
  public void setTargetNumOfJobs(int targetNumOfJobs) {
    ...
  }

  @ProvidedEffector ("criticalNumOfJobs")
  public void setCriticalNumOfJobs(int criticalNumOfJobs) {
  ...
  }
  ...
}
```

Listing 1: Excerpt of controller (1) implementation

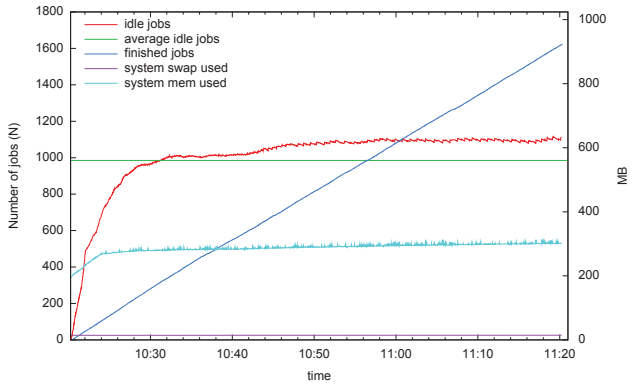
Once we have the model in the DSL and an implementation of all the elements, we transform it into an OSGi bundle. The generated resulting bundle contains the implementation classes with their resources and an activator that is responsible for instantiating and registering the individual elements as OSGi services as well as resolving their dependencies by creating necessary proxies and managing their life-cycles.

This bundle together with the other bundles from the framework can be installed and started by the Equinox container.

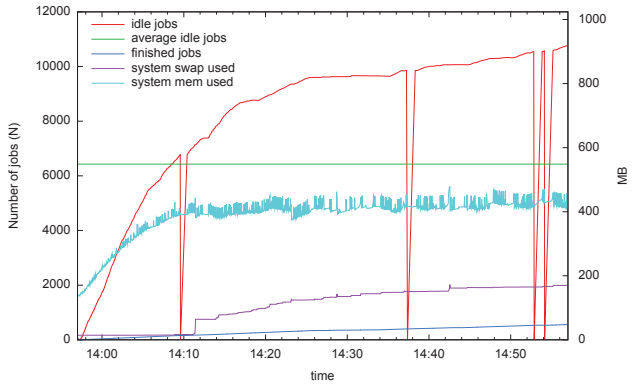
Since Condor DAGMan only evaluates its configuration options when it starts, we made a small modification that refreshed the delay time before every submission.

E. Experimental Results

To evaluate the capacity of the resulting architecture, we set up two Condor deployments, with and without the main feedback control loops. For both runs we used 20 clients



(a) controlled



(b) uncontrolled

Figure 3: System behavior with and without control

together submitting a wide workflow of 1000 tasks each with different running time. We initialized the overload control model with $N^* = 1000, N_c = 1500, p = 5, \rho_0 = 10$. Results are show in figure 3a and figure 3b presenting the system behavior with and without control respectively.

In the feedback controlled system, the amount of idle jobs were for most of the run slightly (6.9%) above the N^*, on average by 69 jobs (116 max). The average load on the host with control was 0.6 in comparison to 3.37 in the system without any control. As shown in figure 3b, major problems arose when the system started running out of memory and started swapping. After that, it spent most of the time waiting for I/O (on average 51.29% of CPU time were *iowait* comparing to 15.6% in case of the managed system). The gaps in figure 3b were caused by timeouts when `condor_q` tried to get the queue information from the schedd agent during the period the system was too stressed (corresponding to load > 9).

Finally, one of the important differences is in the amount of work done during the observed period. Because of the resource waste caused by the overload, the unmanaged system executed only 560 tasks while the controlled version did 1620 tasks. This demonstrates the basic capability of the feedback system generated from our architectural model to regulate the load on this example.

V. Related Work

In addition to the previous work on architecting loops evoked in Section II, some recent works attempt to propose an explicit fine-grain modelling of feedback control loops. In FORMS [18], the focus is more in characterizing the kinds of autonomic architecture with appropriate metamodels so that they can be compared. Similarly, in [12] Hebig et al. provides an UML profile to explicitly architect several coordinated control loops with component diagrams. Nevertheless the loop elements are not adaptive and there is no link to any runtime support.

Regarding control, in [13] Litoiu et al. describe a hierarchical framework that accommodates scopes and timescales of control actions, and different control algorithms. Their architecture considers three main types of controller reflecting the three different stages that they focus on: tuning, load balancing and provisioning. While being similar, our architecture is more general but provides less fine-tuned building blocks to control the behavioral models used inside controllers. A lot of research efforts [2], [20], [21] has been put into tackling the problem of overload control of dynamic distributed systems. The major focus has been in designing a scalable and robust adaptive algorithm. In contrast, our main focus is complementary, dealing with the surrounding architecture in order to provide an appropriate supporting environment for the researchers to focus on the adaptation design.

VI. Conclusion

In this paper, we have presented an approach to support the architectural design of control loop elements and their connection links. The approach relies on an architectural model that makes control loops and their elements explicit. Moreover each element (sensor, controller, effector, links between them) is uniformly designed as a first-class adaptive element. The proposed model does not help in designing the behavior inside the controllers, but rather in architecting all the surrounding elements. Software architects can thus capture different patterns of interactions and controls (coordination of loops, adaptive monitoring, etc.) while reasoning about complex self-adaptive systems. The presented model is also technology agnostic and we provided a default implementation using an OSGi runtime support and a DSL to define architectures from which main parts of the final code are generated.

We also illustrated the architectural model on a overload problem arising in the Condor distributed batch computing system, showing both the capabilities of the model and the efficiency of the resulting code at runtime. The results are obviously only partial and we plan to conduct larger experiments on different and more complex scenarios, both

on Condor and on the gLite grid middleware [3]. This should enable us to identify some bottlenecks on both the architecture and the runtime platform.

Regarding the architectural model, ongoing work consists in turning it into a fully recursive model, so that composition of adaptive elements is supported and larger models can be more easily expressed and managed. Another way to simplify resulting architectures and to improve scalability of the approach is to support inlining of the elements in case there is no direct interaction. Besides, we are currently developing the parts of the runtime support to deal with distribution of the loop elements and making remote links between them self-adaptive.

ACKNOWLEDGMENTS

The authors would like to thank the Condor team from the University of Wisconsin-Madison for all their support, as well as Mireille Blay-Fornarino and the reviewers for valuable comments on the model.

The work reported in this paper is partly funded by the ANR SALTY project[4] under contract ANR-09-SEGI-012.

REFERENCES

[1] J. Kephart and D. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, Jan. 2004.

[2] J. Hellerstein, Y. Diao, S. Parekh, and D. Tilbury, *Feedback control of computing systems*. Wiley Online Library, 2004.

[3] B. Cheng, R. de Lemos, H. Giese, P. Inverardi, J. Magee, J. Andersson, B. Becker, N. Bencomo, Y. Brun, B. Cukic, and Others, "Software Engineering for Self-Adaptive Systems: A Research Roadmap," *Software Engineering for Self-Adaptive Systems*, pp. 1–26, 2009.

[4] H. Müller, M. Pezzè, and M. Shaw, "Visibility of control in adaptive systems," *Proceedings of the 2nd international workshop on Ultra-large-scale software-intensive systems - ULSSIS '08*, pp. 23–26, 2008.

[5] Y. Brun, G. Di Marzo Serugendo, C. Gacek, H. Giese, H. Kienle, M. Litoiu, H. Müller, M. Pezzè, and M. Shaw, "Engineering Self-Adaptive Systems Through Feedback Loops," *Software Engineering for Self-Adaptive Systems*, pp. 48–70, 2009.

[6] M. Salehie and L. Tahvildari, "Self-adaptive software: Landscape and research challenges," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 4, no. 2, pp. 1–42, 2009.

[7] A. Computing, "An Architectural Blueprint for Autonomic Computing, 4. edt," *IBM Autonomic Computing White Paper*, vol. 36, 2006.

[8] G. Kaiser, J. Parekh, P. Gross, and G. Valetto, "Kinesthetics eXtreme : An External Infrastructure for Monitoring Distributed Legacy Systems," in *Proceedings of the Autonomic Computing Workshop, 5. International Workshop on Active Middleware Services*, Seattle, 2003, pp. 4–13.

[3] http://glite.cern.ch/

[4] https://salty.unice.fr

[9] D. Garlan, B. Schmerl, and P. Steenkiste, "Rainbow: architecture-based self-adaptation with reusable infrastructure," *International Conference on Autonomic Computing, 2004. Proceedings.*, pp. 276–277, 2004.

[10] H. Liu, M. Parashar, and S. Hariri, "A component based programming framework for autonomic applications," in *Proc. of 1st International Conference on Autonomic Computing*, vol. 9984357, no. 82. Citeseer, 2004, pp. 10–17.

[11] H. Muller, H. Kienle, and U. Stege, "Autonomic Computing Now You See It, Now You Don't," *Software Engineering*, pp. 32–54, 2009.

[12] R. Hebig, H. Giese, and B. Becker, "Making Control Loops Explicit When Architecting Self-Adaptive Systems," *Proceeding of the Second International Workshop on Self-Organizing Architectures - SOAR '10*, p. 21, 2010.

[13] M. Litoiu, M. Woodside, and T. Zheng, "Hierarchical Model-Based Autonomic Control of Software Systems," *Proceedings of the 2005 Workshop on Design and Evolution of Autonomic Application Software - DEAS '05*, p. 1, 2005.

[14] A. Barker and J. Van Hemert, "Scientific workflow: A survey and research directions," in *Proceedings of the 7th international conference on Parallel processing and applied mathematics*. Springer-Verlag, 2007, pp. 746–753.

[15] L. Broto, D. Hagimont, E. Annoni, B. Combemale, and J.-P. Bahsoun. Towards a model driven autonomic management system. In *Fifth International Conference on Information Technology: New Generations*, pages 63–69, 2008.

[16] D. Thain, T. Tannenbaum, and M. Livny, "Distributed computing in practice: The Condor experience," *Concurrency and Computation Practice and Experience*, vol. 17, no. 2-4, pp. 323–356, 2005.

[17] OSGi Alliance, "Listeners Considered Harmful : The "Whiteboard " Pattern," Technal Whitepaper, 2004. http://www.osgi.org/wiki/uploads/Links/whiteboard.pdf

[18] D. Weyns, S. Malek, and J. Andersson, "FORMS: a formal reference model for self-adaptation," in *ICAC 2010*, 2010, pp. 205–214.

[19] R. Nzekwa, R. Rouvoy, and L. Seinturier, "Modelling Feedback Control Loops for Self-Adaptive Systems," *Electronic Communications of the EASST Volume 28 (2010)*, vol. 28, 2010.

[20] N. Bartolini, G. Bongiovanni, and S. Silvestri, "Self-* overload control for distributed web systems," *The IEEE Proceedings of the*, no. November 2008, pp. 50–59, Jun. 2008.

[21] S. Parekh, N. Gandhi, J. Hellerstein, D. Tilbury, T. Jayram, and J. Bigus, "Using control theory to achieve service level objectives in performance management," *Real-Time Systems*, vol. 23, 2002, pp. 127–141.

A Metamodel for Distributed Ensembles of Virtual Appliances

Xabriel J. Collazo-Mojica and S. Masoud Sadjadi
School of Computing and Information Sciences
Florida International University
Miami, FL 33199, USA
Email: {xcoll001, sadjadi}@cs.fiu.edu

***Abstract*—We present our work on modeling distributed ensembles of virtual appliances (DEVAs) on Infrastructure as a Service (IaaS) clouds. Designing solutions on IaaS providers require a good understanding of the underlying details such as the software installation or the network configuration. We propose the use of DEVAs, a modeling approach built on top of the notion of virtual appliances, that allows easy-to-compose and ready-to-use cloud application architectures that are IaaS-agnostic, and that abstract away unnecessary details for web application developers. In this paper, we extend the definition of a DEVA from previous work by presenting an underlying metamodel and how that metamodel can be transformed to an actual deployment. We also present a case study where we model a web application architecture and we discuss how we can instantiate it in an IaaS cloud. We argue that the DEVA modeling approach is suitable for typical cloud use cases.**

***Index Terms*—virtual appliance, non-functional requirements, cloud computing architectures.**

I. Introduction

In this paper, we extend our work on modeling groups of interdependent virtual machines in the cloud. Designing these compositions require a good understanding of the underlying details such as the software installation or the network configuration. They are typically deployed in a cloud layer called Infrastructure as a Service (IaaS). Each IaaS provider has its own API to configure the virtual machines and its connections, requiring users to learn the details of yet another API. Similarly, manually installing software stacks in these virtual machines is a tedious task. In [1], we presented a visual modeling approach to make these architectures easy-to-compose and ready-to-use. We call these models *distributed ensembles of virtual appliances* (DEVAs)[1]. We now extend our work by presenting an underlying meta-model for DEVAs.

Sapuntzakis et al. presented in 2003 the idea of a virtual appliance, effectively treating full stacks of software applications and OS as updatable image files [2]. These virtual appliance files could then be cloned in bare metal computers, or instantiated in virtual machines. Still, when the time comes to compose multi appliance systems with interdependencies, Sapuntzaki's implementation relied on defaults from software vendors for most of the configurations. This may not be the case for web applications (e.g., when trying to configure interdependencies between a database server and its clients). Similarly, the configuration of the network connection between appliances had to be done by hand. Various recent attempts at automatically configuring virtual appliances and their network dependencies have been presented [3], [4], [5], but they all rely on having experts on appliance configurations, or on specific IaaS providers.

In [1], we proposed that with proper modeling of these kind of scenarios, the configuration problems could be abstracted away. By designing DEVAs, non-expert users can easily architect interdependent virtual appliances. DEVA models include quality of service (QoS) constraints, which can account for the non-functional requirements of the modeled architecture. In this paper, we extend our modeling approach by 1) presenting an underlying meta-model, and 2) how a model generated from the meta-model can be instantiated to an actual deployment.

The main technical challenge for this work was to come up with a meta-model with sufficient details to be able to instantiate DEVAs on the spectrum of current IaaS providers. To demonstrate our modeling approach, we present a case study where we model a web application architecture and discuss how we can instantiate it in a current IaaS provider.

We argue that the DEVA modeling approach is suitable for typical cloud use cases, and that having a model of a cloud architecture can simplify co-allocations and migrations across-different IaaS providers. In Section II, we present background information on the methodology used and on our previous work. In Section III, we present our metamodeling framework. In Section IV-A, we discuss a system that can transform DEVA models into instances, and in Section IV-B, we present a simple case study. In Section V, we discuss related work, and finally, in Section VI, we enumerate some concluding remarks.

II. Background

In this section, we present background information on model-driven engineering, the methodology used for this work, and present details on previous work including the definition of DEVAs.

[1]Note that in [1] we called our models *virtual environments* instead of DEVAs. We have desisted of the previous name as it is already used in other CS areas.

A. Model-Driven Engineering

MDE is a methodology that aims to effectively apply models to software development. Instead of looking at models as simple diagrams, MDE captures the problem domain in a modeling language. This language can later be used to generate solutions that can be automatically generated. MDE has many goals, including accelerated development, automated transformations, and platform independence [6].

In this work, we utilize the MDE methodology to present a modeling approach that simplifies cloud architecture design, as well as to achieve platform independence from IaaS providers. In addition, we use MDE's metamodeling approach as presented in [6] to realize an abstract syntax definition for our models. Static semantics are presented in Object Constraint Language (OCL) notation [7].

B. Distributed Ensembles of Virtual Appliances

A *distributed ensemble of virtual appliances* is a model of a logical architecture of interdependent virtual appliances. From the point of view of users of DEVAs, these models should present the following properties:

1) **Easy to understand**: Views for the design, deployment, change management, and monitoring of these architectures should only present what is strictly necessary to realize them. Advanced options should be available but normally hidden.
2) **Self-configurable**: Once the user has specified a description of the architecture needed, our solution should be able to instantiate the model and configure all the details automatically by following the constraints and policies specified.
3) **Present deployment choices**: Modeling should be abstract enough to allow for an implementation to present deployment choices. Given the heterogeneity of current IaaS APIs, this is one of the main challenges of our approach.

C. Visual Concrete Syntax for DEVAs

Research has shown the benefits of visual aids when designing system architectures, claiming a gain of over 60% in comprehension [8]. Since our target users are web developers, which may not be experts on architectures for the cloud, a graphical representation is desirable. We model virtual appliances with boxes with *service endpoints*. The boxes represent all the necessary software, the OS and the configuration necessary to support the services provided or consumed. Figure 1 presents an example of a DEVA composed of two virtual appliances. The box entitled 'RoR Node' is a representation of an appliance provisioned with the Ruby on Rails web framework (and all other needed software). Similarly, the box entitled 'MySQL DB' is an appliance provisioned with a MySQL database. These appliances have been interconnected with a 'db' link by joining the corresponding endpoints. This connection assumes that any interdependent configuration will be resolved by our solution. For example, to be able to provide the db service, a username and password must be agreed by the db provider and db consumers. In our approach, there is no need to configure IP addresses, ports or configuration files.

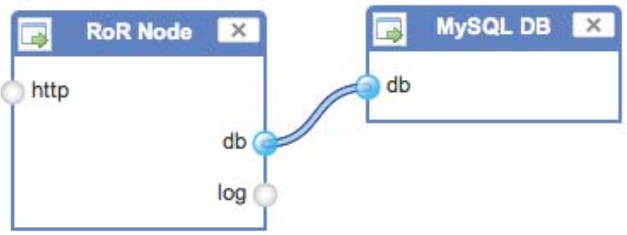


Fig. 1. An example of a simple DEVA as presented in [1].

III. A Metamodeling Framework for DEVAs

In [1], we investigated the first property of our model by presenting an uncomplicated visual concrete syntax to design DEVAs. In this paper, we investigate the second property. We realized that for our solution to be able to model arbitrarily complex cloud architectures, we needed a formal definition of our modeling approach. The third property, to be able to present deployment choices, remains as future work.

Having an underlying metamodeling framework for DEVAs facilitates the steps necessary to go from a model to an instance in a similar way that the Meta-Object Facility (MOF) defines the UML language. Figure 2 presents an overview of the framework. Note that to keep our modeling as simple as possible, we do not currently use the MOF, or UML Profiles to define our framework.

At the top of the figure, we have the DEVA Meta-metamodel. This level defines what the valid constructs are for our modeling approach. Based on this constructs, we propose two different DEVA metamodels with the main difference being that one allows the instantiation of resource-independent (RI-DEVA) models, and the other allows resource-dependent (RD-DEVA) models. Resources in this paper mean computational resources as we would typically obtain from a IaaS provider, such as CPU allocations, RAM memory, and network connection speed and IP addresses.

We make this resource dependency distinction for two main reasons. First, to present the user with a simple designing tool that separates the concern of modeling a DEVA, and the concern of modeling the resources needed to run such DEVA. By having a RI-DEVA, a user can let our framework allocate the required resources based on the specified high-level policies and constraints. We can achieve this by transforming an RI-DEVA to a RD-DEVA using model to model (M2M) transformations [6].

Second, the resource performance from IaaS providers has been shown to have high variability [9]. Similarly, different IaaS providers have different metrics for specifying available resources. Thus, we believe it is desirable to model application architectures by quantifiable SLA constraints (*i.e.,* "able to do 100 transactions per second"), rather than subjective resource

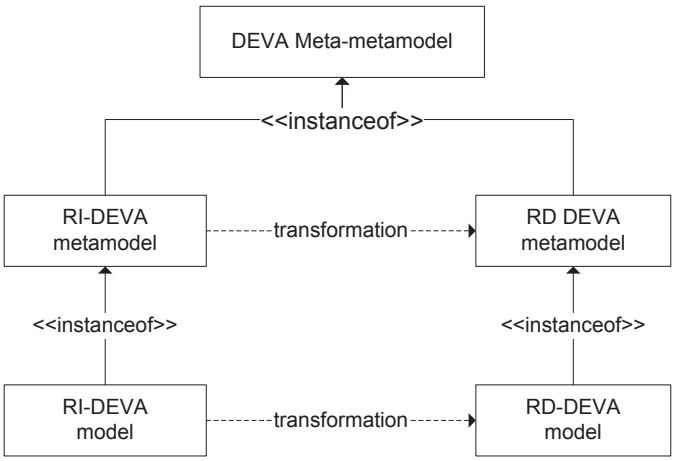
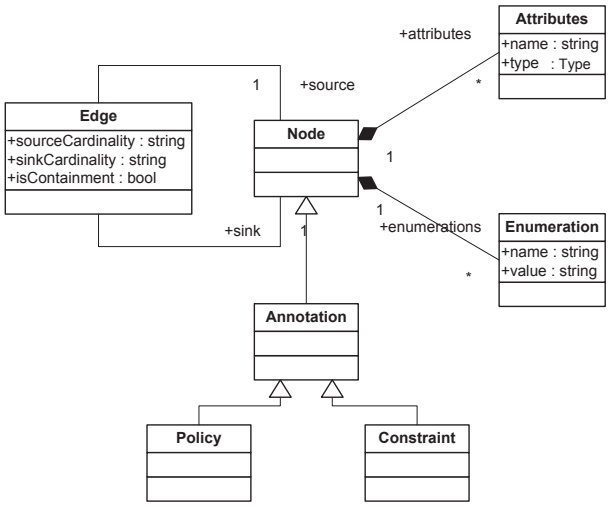


Fig. 2. Metamodeling Framework for DEVAs.



(a) Abstract syntax for DEVA meta-metamodel.

```
context Node
inv: self.attributes.size == 0 or
     self.enumerations.size == 0

context Edge
inv: self.source != self.sink
```

(b) Static semantics for DEVA meta-metamodel.

Fig. 3. Meta-metamodel for DEVAs.

metrics (*i.e.,* "having an equivalent of a 1GHz CPU with 256MB RAM") as IaaS providers currently offer [10]. Our main concern is to allow developers to specify RI-DEVAs that are transformed to RD-DEVAs by our system.

A. DEVA Meta-metamodel

Our proposed DEVA meta-metamodel abstract syntax is presented on Figure 3(a), and the static semantics on Figure 3(b). At this level, we can construct simple graphs, with Nodes and Edges. This metamodel is a simple extension of a graph which allows Annotations against Nodes. These Annotations can be either Policies or Constraints. Each of the Nodes can have a list of Attributes, or a list of Enumerations, but not both. Edges have a source Node and a sink Node, which cannot be the same Node. An attribute is composed by a name and a type. Enumerations have names and literal values. With these basic constructs, we can generate the metamodels for both RI-DEVAs and RD-DEVAs, which we describe below.

B. RI-DEVA Metamodel

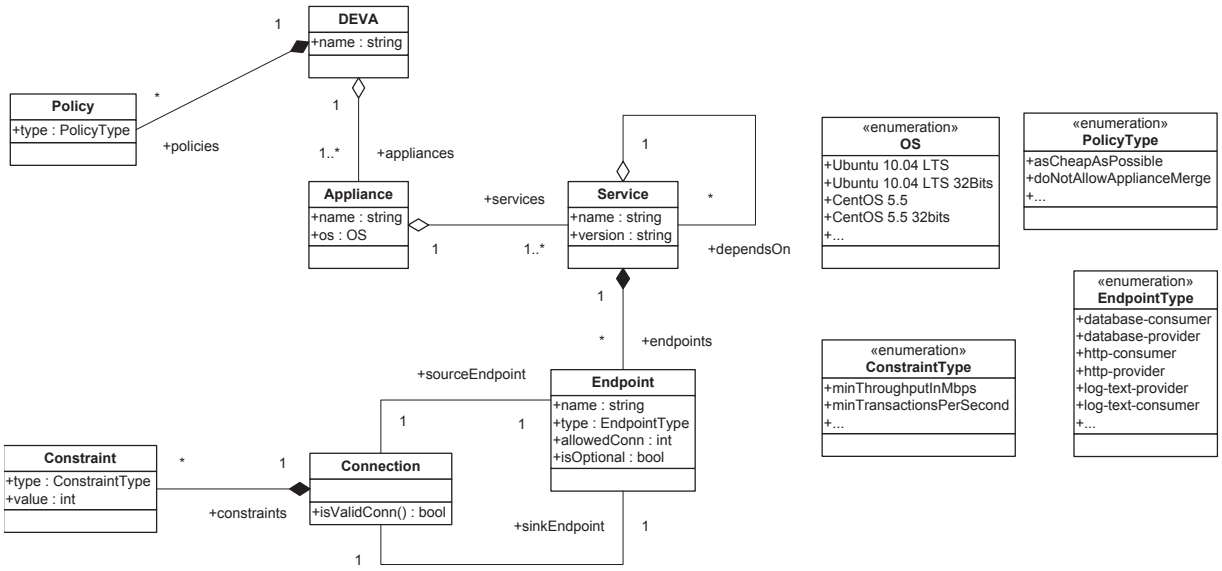
Our proposed RI-DEVA meta-model abstract syntax is presented in Figure 4(a), and the static semantics in Figure 4(b). An RI-DEVA is a named composition of zero or more global Policies and one or more virtual Appliances. In this context, Policies refer to high-level guidelines that a DEVA instance will try to meet. An example of a policy is to be able to instantiate a RI-DEVA with the least amount of resources possible (*i.e.* 'asCheapAsPossible' in our PolicyType enumeration). This means that economy will have a bigger weight whenever we try to realize this RI-DEVA. A RI-DEVA could include policies that are contradicting. In such cases, they would be accommodated in an arbitrary order.

At least one virtual appliance is necessary to convey a RI-DEVA. Note again that our Appliance definition is isolated completely from resource usage. These Appliances are represented by a name, a specific operating system, and a list of one or more services.

Each service is identified by a name, a version number, a list of the services it depends on (if any), and a set of zero or more endpoints. These endpoints are key to our approach, as can be seen in Figure 1. They allow to specify service production or consumption at the service level, rather than specifying network level details. This approach implicitly generates meta-data about the architecture that allows for easier appliance provisioning, as all the connection and service-to-service dependencies are known *a priori*.

Endpoints are represented by a name, a type, and an integer that specifies how many connections are allowed. A connection is composed of a source endpoint linked to a sink endpoint; sink and source cannot be the same endpoint. Connections should have valid type matches at each endpoint. Going back to Figure 1, a database-consumer can only be connected to a database-provider, and the provider endpoint can specify a maximum number of consumer connections. For this, we include a function $isValidConn : EndpointType \times EndpointType \rightarrow bool$ which determines Connection validity based on Endpoint type and their allowedConn property. Endpoints also specify if they can be left unused, or if a connection is necessary.

Each connection can have zero or more Constraints. These Constraints specify QoS non-functional requirements for the service-to-service connections. For example, in the case of a database connection, a constraint "at least 50 transactions per second" could be applied. Again, this meta-data could be used in the model transformation to try to guarantee the constrained connection performance.



(a) Abstract syntax for RI-DEVAs.

```
context Connection
inv: sourceEndpoint != sinkEndpoint and isValidConn()

context Service
inv: self.dependsOn->forAll(dep: Service | dep != self)

context Constraint
inv: self.value >= 0

context Endpoint
inv: self.allowedConn > 0
```

(b) Static semantics for RI-DEVAs.

Fig. 4. Meta-metamodel for RI-DEVAs.

C. RD-DEVA Metamodel

Figure 5(a) presents a partial abstract syntax for RD-DEVAs, and Figure 5(b) the static semantics. These models have a similar metamodel as RI-DEVAs. The main difference is that we do not have policies nor constraints; rather, we have IaaS resources mapped to each Appliance. These Appliances must be mapped to 3 or more resources, and there should be at least one resource of CPU, RAM, and NIC, respectively.

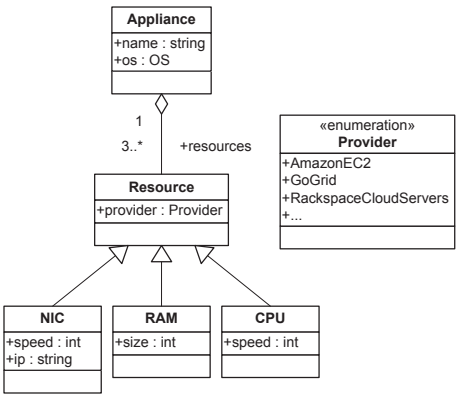
In this paper, we focus on how to construct RI-DEVAs and how to transform them to RD-DEVAs. Work on directly modeling RD-DEVAs is presented elsewhere [11].

IV. Transforming DEVAs

A. A model to instance reference system

Figure 6 presents an overview of a system that can transform DEVA models into instances. As input, the system would have a RI-DEVA model. This model would be received by a Model to Model (M2M) transformation engine aware of the metamodels for RI- and RD-DEVAs. The transformation between these models can be done in various ways. The task is to translate quantifiable constraints and policies to available resources in an IaaS cloud. This could be realized by having a mapping $Instantiate : ConstraintType \times Constraint.value \longrightarrow < R_{CPU}, R_{RAM}, R_{NIC} >$ applied to all of the services in a specific virtual appliance, where $< R_{CPU}, R_{RAM}, R_{NIC} >$ is a tuple representing the allocated resources in a specific IaaS cloud. This mapping could be implemented by using resource usage predictions based on empirical data in a black box manner, as in previous work where we model resources used by specific software [12], or in a white box approach as presented in [13]. The results of the mapping can then be aggregated to have the total needed resources for an Appliance.

Once we have a RD-DEVA, we need to have a DEVA model to instance (M2I) transformation engine that is aware of the IaaS resources. Again, we have various choices. Either the engine could be an IaaS provider which is aware of DEVAs, as in [11], or the engine could be a middleware which translates RD-DEVA models to IaaS API calls. To resolve interdependencies and to provision the appliances with the necessary software, this middleware would generate configuration scripts



(a) Partial abstract syntax for RD-DEVAs.

```
context Appliance
inv: self.resources->
          exists(c | c.oclIsKindOf(CPU))
inv: self.resources->
          exists(r | r.oclIsKindOf(RAM))
inv: self.resources->
          exists(n | n.oclIsKindOf(NIC))
```

(b) Partial static semantics for RD-DEVAs.

Fig. 5. Partial metamodel for RD-DEVAs. All other entities are as in RI-DEVAs if we eliminate the Policies and Constraints.

which would then be run on the instantiated Virtual Machines (VM) on top of the IaaS provider. Note that in our approach, a different M2I engine would be needed for each target IaaS provider. Nonetheless, some work from this engine can be modularized. For example, to make the software provisioning process repeatable, a software installation utility like Chef could be used [14].

B. A Simple Case Study

Because of space constraints, we present a simple case study of our approach based on Figure 1. A small composition like this can be instantiated as follows. First the user would define this RI-DEVA by using our prototype DEVA designer. We chose to have a web-based designer for DEVAs to provide a cross-platform solution. The model would then be sent to the M2M engine. Note that we can always compare the DEVA being designed against the metamodel to find discrepancies, and acknowledge the designer accordingly.

Since the model in Figure 1 does not present any specific Constraints, our system would choose the simplest transformation, which could be a direct mapping to, say, two Virtual Machines in Amazon EC2 of the type "small instance". A small instance in Amazon EC2 provides one CPU allocation, with 1.7GB of RAM, and a basic network connection with two IPs, one public and one private [10]. This would be the transformation to an RD-DEVA.

Now that we have an RD-DEVA, we can send the model to the M2I engine. On this engine, we can instantiate the architecture by making the IaaS-specific API calls necessary to launch 2 VMs, and then provisioning the software of each virtual appliance. After this, we would need to resolve the interdependencies (such as username and password for the database) by running configuration scripts on the instantiated VMs. Note that the meta-data of the Connections could be used in this step. For example, on Amazon EC2, the private IP could be used to make the DB connection, instead of using the public one. Similarly, we also know that there is only one connection possible to the database server. Therefore, the instance could be configured to reject any connection other than to the DB port.

V. Related Work

We have identified the need for better abstractions from the current IaaS implementations provided by vendors such as Amazon [10] or GoGrid [15]. Thus, we propose a modeling approach that is abstract enough to allow these interdependent systems to be easy-to-design, fast-to-deploy, and that limits the effects of IaaS vendor lock-in. We do not currently use a standardized IT information model, such as DMTF's Common Information Model [16], as current IaaS providers do not support them.

Commercial applications implementing a similar modeling approach are available [17], [18]. They only offer closed-source implementations and only work on their proprietary cloud platforms. IBM has worked on a similar project, but their implementation assumes that users are experts in the domains of virtual image provisioning, image composition, and composition deployment [3]. While they target enterprise customers, we target non-expert cloud users.

Platform as a Service (PaaS) providers, such as Google AppEngine [19], abstract away the underpinnings of a fully working web application. Of course, this means that the user has to learn the vendor's API, and that migrating the application to other PaaS provider implies changing most of the implementation. Our work envisions models that once specified do not need to be changed because of a vendor switch.

Recently, Amazon started to offer a service called CloudFormation. Customers can now specify groups of virtual machines with provisioning scripts that provide repeatable architecture instantiation. Compared to our solution, Amazon's offering only work on their IaaS service and of course the details of their API need to be well known. Nonetheless, this is a welcomed addition to their cloud offering, and validates the current and future importance of easier-to-compose solutions for the cloud.

VI. Concluding Remarks

In this paper, we presented a modeling approach for DEVAs. First, we defined a metamodeling framework to be able to formulate two metamodels, one for RI-DEVAs, and the other for RD-DEVAs. We argued that RI-DEVAs are more desirable for describing cloud architectures, and that an automatic transformation between these two metamodels allows developers to specify DEVAs with quantifiable QoS constraints rather than

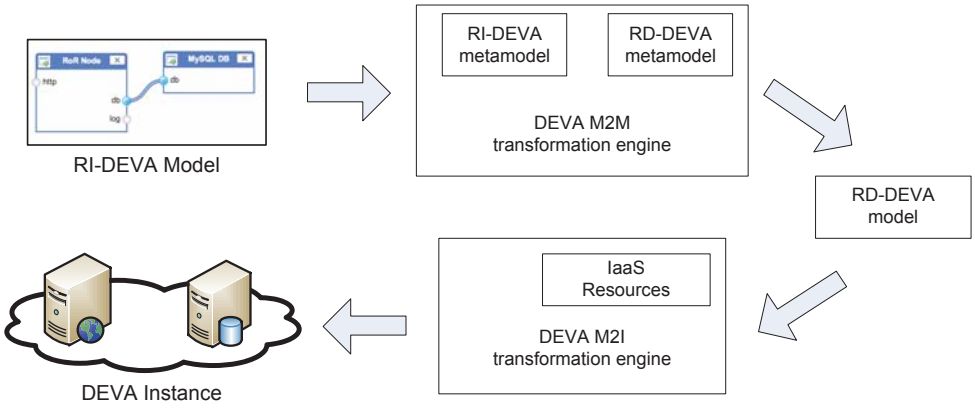


Fig. 6. Transforming DEVA models to DEVA instances.

by subjective performance metrics provided by current IaaS providers.

We then presented the details of our framework, by describing RI-DEVAs and RD-DEVAs with their abstract syntax and static semantics. We also described our key modeling abstraction, which consist of making service-to-service connections between Appliances, so that the architecture developer does not have to specify network level details. This abstraction aids in two ways: network level details may not be the developer's expertise area, and also its configuration can vary between different IaaS APIs.

We also presented how a system could implement our modeling approach, by describing the various stages needed to transform a RI-DEVA modeled by a developer to an actual instance in an IaaS Provider. Finally, a simple case study of how a typical web architecture could be modeled with our approach was discussed.

Designing solutions on top of IaaS providers is a growing problem domain in the cloud. A modeling approach such as the one presented can potentially make these architectures easy-to-compose and ready-to-use. We are in the process of prototyping a system that follows the details presented in Section IV. We are also investigating how to expand our approach to monitor and autonomically enforce Constraints and Policies of instantiated DEVAs.

ACKNOWLEDGMENT

We appreciate the discussions held with David Villegas. This work was supported in part by a GAANN Fellowship from the US Department of Education under P200A090061 and in part by the National Science Foundation under Grant No. OISE-0730065.

REFERENCES

[1] X. J. Collazo-Mojica, S. M. Sadjadi, F. Kon, and D. D. Silva, "Virtual environments: Easy modeling of interdependent virtual appliances in the cloud," *SPLASH 2010 Workshop on Flexible Modeling Tools*, Aug 2010.

[2] C. Sapuntzakis, D. Brumley, R. Chandra, N. Zeldovich, J. Chow, M. S. Lam, and M. Rosenblum, "Virtual appliances for deploying and maintaining software," *LISA '03: Proceedings of the 17th USENIX Large Installation Systems Administration Conference*, pp. 181–194, Aug 2003.

[3] A. Konstantinou, T. Eilam, M. Kalantar, A. Totok, W. Arnold, and E. Snible, "An architecture for virtual solution composition and deployment in infrastructure clouds," *VTDC '09: Proceedings of the 3rd international workshop on Virtualization technologies in distributed computing*, Jun 2009.

[4] T. Chen, Y. Wang, Y. Ren, C. Luo, D. Qian, and Z. Luan, "R-ECS: reliable elastic computing services for building virtual computing environment," *ICIS '09: Proceedings of the 2nd International Conference on Interaction Sciences: Information Technology, Culture and Human*, Nov 2009.

[5] K. Keahey and T. Freeman, "Contextualization: Providing one-click virtual clusters," *IEEE Fourth International Conference on eScience*, pp. 301–308, 2008.

[6] T. Stahl and M. Voelter, *Model-Driven Software Development: Technology, Engineering, Management*. John Wiley & Sons, 2006.

[7] "Object Constraint Language," March 2011. [Online]. Available: http://www.omg.org/spec/OCL/2.2/

[8] J. Knodel, D. Muthig, and M. Naab, "Understanding software architectures by visualization–an experiment with graphical elements," *WCRE '06: 13th Working Conference on Reverse Engineering*, pp. 39–50, 2006.

[9] S. Ostermann, A. Iosup, N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema, *A Performance Analysis of EC2 Cloud Computing Services for Scientific Computing*, ser. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. Springer Berlin Heidelberg, 2010, vol. 34, pp. 115–131.

[10] "Amazon Elastic Compute Cloud," March 2011. [Online]. Available: http://aws.amazon.com/ec2/

[11] D. Villegas and S. M. Sadjadi, "DEVA: Distributed ensembles of virtual appliances in the cloud," Florida International University - School of Computer and Information Sciences, Tech. Rep. FIU-SCIS-2011-03-22, March 2011.

[12] S. Sadjadi, S. Shimizu, J. Figueroa, R. Rangaswami, J. Delgado, H. Duran, and X. Collazo-Mojica, "A modeling approach for estimating execution time of long-running scientific applications," in *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, 2008, pp. 1 –8.

[13] A. Ganapathi, Y. Chen, A. Fox, R. Katz, and D. Patterson, "Statistics-driven workload modeling for the cloud," *Data Engineering Workshops (ICDEW), 2010 IEEE 26th International Conference on*, pp. 87 – 92, 2010.

[14] "Chef Systems Integration Framework," March 2011. [Online]. Available: http://wiki.opscode.com/display/chef/Home

[15] "GoGrid Cloud Hosting," March 2011. [Online]. Available: http://www.gogrid.com/cloud-hosting/cloud-servers.php

[16] "DMTF's Common Information Model," May 2011. [Online]. Available: http://www.dmtf.org/standards/cim

[17] "3Tera Inc." March 2011. [Online]. Available: http://www.3tera.com/

[18] "Elastra Corporation," March 2011. [Online]. Available: http://www.elastra.com/

[19] "Google App Engine," March 2011. [Online]. Available: http://code.google.com/appengine/

Towards Automated Conformance Checking of ebBP-ST Choreographies and Corresponding WS-BPEL Based Orchestrations

Matthias Geiger, Andreas Schönberger and Guido Wirtz

Distributed and Mobile Systems Group, University of Bamberg
Feldkirchenstr. 21, 96052 Bamberg, Germany
E-mail: {matthias.geiger | andreas.schoenberger | guido.wirtz}
@uni-bamberg.de

Abstract

Web Services technologies are a natural candidate for Business-to-Business integration (B2Bi). For cross-organizational processes, the concepts of "choreography" and "orchestration" are important. The term choreography denotes a model of a global view over message exchange scenarios, whereas the term orchestration focuses on models of the local implementation. While WS-BPEL is already kind of a de-facto standard in the field of orchestration languages, there does not exist a standard choreography language. We propose the usage of ebXML BPSS (ebBP) in order to provide choreography modeling at the business level. A frequent problem is to ensure and enforce the consistency and conformance of choreography and orchestration models which is often referred to as "conformance checking". In this paper we examine a way to check the conformance between ebBP-ST (a subset of ebBP) choreographies and corresponding WS-BPEL based implementations. To achieve this check well-known and approved model checking methods and tools are used: First ebBP-ST choreographies are directly transformed into the process algebra CCS. Second, the low level WS-BPEL processes are analyzed for code blocks that implement choreography elements and the sequence of these code blocks is then mapped to CCS, too. Afterwards these formalized representations will be checked for bisimulation equivalence in order to reveal inconsistencies between the choreography and their implementations.

Keywords: SOA, choreography, orchestration, conformance checking

1. Introduction

In the domain of Business-2-Business integration (B2Bi) Web Services technologies provide a suitable solution for integrating the cross-organizational business processes. As in B2Bi scenarios a global coordinator cannot be assumed, the distinction between choreographies and orchestrations is important: Choreographies define a process from a global perspective which may be seen as a communication contract between the integration partners involved. Orchestrations describe the local, executable implementation for each of the partners. In the B2Bi domain the *ebXML Business Process Specification Schema* (ebXML BPSS or ebBP; [8]) is a suitable choice for choreography modeling as it provides several features that are well-suited for the business domain. One core concept of ebBP is the usage of so-called *Business Transactions* (BTs) to model the exchange of a single business document from a sending to a receiving role - optionally followed by a response document and so-called *Business Signals* which notify the senders about the processing status of the exchanged documents. More complex scenarios composed of more document exchanges may be modeled as *Business Collaborations* (BCs). Within BCs, Business Transaction Activities (BTAs) are used to require the execution of BTs and to map BC roles to the roles of the previously defined BTs. The control flow and the ordering between BTAs is modeled in BCs using basic control flow constructs like *Transitions*, *Decisions*, *Forks* and *Joins*.

In [9], an ebBP dialect called ebBP-ST is presented to introduce explicitly modeled *Shared States* (STs) into ebBP choreographies which leads to clearer models in complex scenarios by providing control flow synchronization points.

In order to execute the choreography models defined in ebBP-ST, they have to be transformed into executable orchestrations. In the area of Web Services technologies the *Web Services Business Process Execution Language* (WS-BPEL; [7]) is the de-facto standard for realizing orchestrations which is also used in [9] to implement ebBP-ST choreographies.

For enabling executable orchestrations the integration architecture depicted in figure 1 is assumed between two partners A and B. These partners use backend components to

encapsulate business logic and control processes to enforce the correct sequence of message exchanges as defined in ebBP-ST choreographies. In this paper, we investigate automated conformance checking of WS-BPEL based control process implementations and ebBP-ST choreography definitions.

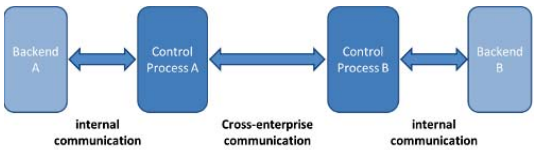


Figure 1. Integration Architecture (adapted from [9])

The paper proceeds as follows: First, in Sec. 2 a use case which serves as a running example is introduced. In section 3, the approach of checking ebBT-ST choreographies and WS-BPEL orchestrations is presented by clarifying which process models are supported, choosing a suitable conformance notion, introducing algorithms to transform ebBP-ST resp. WS-BPEL to the process algebra CCS [5], and describing the actual conformance check. The paper concludes with a discussion of related work and an outlook on ongoing research.

2. Use Case

Throughout the paper an excerpt of a purchasing use case between two partners shall be used to clarify the proposed approach. A visual representation of this ebBP-ST choreography is given in Fig. 2. The use case consists of three different STs and BTAs: The process starts with the ST "*initPurchase*" in which the BTA "*requestQuote*" has to be performed. This BTA defines the exchange of a business document containing a quote request. In the subsequent decision it is checked whether the request could be successfully transfered to the receiver ("*BusinessSuccess*") or not ("*TechnicalFailure*"). In case of a "*TechnicalFailure*", the current ST "*initPurchase*" is not left and the BTA has to be repeated. Otherwise, if the result is "*BusinessSuccess*" the process enters the new ST "*receivedQuote*".

When a quote has been received successfully it may be accepted by a corresponding BTA "*acceptQuote*" which leads to another ST "*concludedContract*" in case this BTA finished with a "*BusinessSuccess*", otherwise the BTA shall be repeated. The other option in ST "*receivedQuote*" is to request another quote by performing the BTA "*requestQuote*" again. The decision afterwards evaluates whether the BTA has been executed successfully or not. Here in both cases the process will be resumed in ST "*receivedQuote*". The difference between both paths is that the ST shall not be left in case the BTA finished with a "*TechnicalFailure*", i.e., the timer defined for the BTA shall not be reset, but shall be reentered with resetting the timer in case of a "*BusinessSuccess*". Note that this fact is not respected in the visualization of the use case in Fig. 2. The last aspect to consider is that a ST may be left by a timeout event. Here, a timeout occuring in ST "*initPurchase*" and "*receivedQuote*" will terminate the process in the end state "*TechnicalFailure*". The further progress of the purchase scenario, e.g., shipment and billing is left out here.

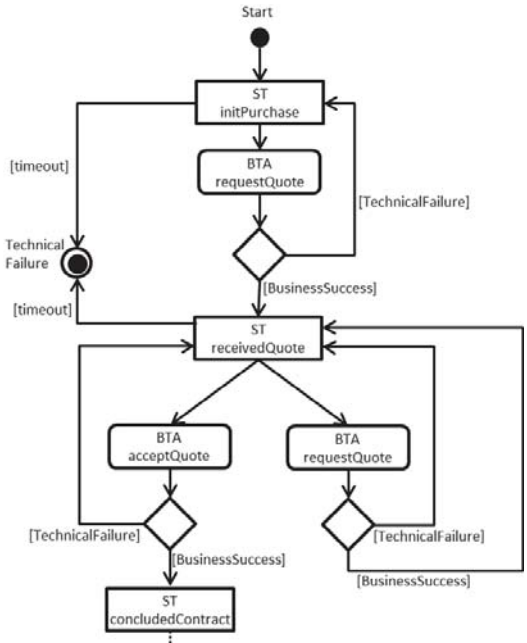


Figure 2. ebBP Structure of the Use Case

3. Conformance Checking ebBP and WS-BPEL

Figure 3 gives a brief overview of the proposed conformance checking approach: First, the given ebBP-ST choreography model and the corresponding WS-BPEL orchestrations which should be checked have to be transformed into a common formal representation. The process algebra CCS is appropriate for the scope of our analysis. After transforming ebBP resp. WS-BPEL into CCS models, the orchestration CCS models can be checked for conformance to the choreography model successively. As conformance is seen as a binary decision in our work, the conformance check returns whether the checked orchestration model conforms to the choreography specification or not.

3.1. Supported Process Definitions

Before describing the approach of checking the conformance between ebBP and WS-BPEL it has to be clarified which kind of processes are supported. As mentioned in the previous section, ebBP-ST models are used to model ebBP

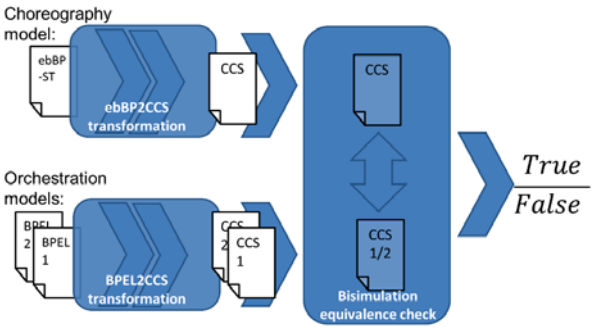


Figure 3. Proposed Approach (Overview)

choreographies. Our approach supports arbitrary ebBP-ST models which are valid regarding the formal definitions and rules for well-formedness in [9]: ebBP-ST models may be formalized as a 5-tuple (R, N, G, ϕ, θ) where R is a set of participants, N is a set of allowed ebBP nodes, G is a set of guards, ϕ is a function that assigns timeouts to each ST and θ is a transition relation which defines all allowed transitions. The set of nodes $N = \{s_0\} \cup ST \cup SBTA \cup DEC \cup T$ contains the ebBP concepts of a starting node, shared states, business transaction activities, decisions and end states which are used to model choreographies. Some important restrictions of ebBP-ST models are that only binary collaborations are supported, ebBP Forks and Joins are not allowed and STs may not overlap. The introduced use case is an example of a rather simple, well-formed model which uses all main aspects of ebBP-ST.

While a formal model of ebBP-ST and its semantics, is given in [9], such a formalization is not provided for the corresponding WS-BPEL implementations. There exist various formalisms for WS-BPEL (see [10]) but all of them are intended to directly formalize low level WS-BPEL language constructs, which would create much too detailed models for our use cases as we are only interested in a formal representation of the control flow between WS-BPEL patterns that realize ebBP choreography concepts. For this purpose the ebBP formalization is adapted to the WS-BPEL realizations: With respect to the set of nodes N it is obvious that all of these aspects have to be represented in WS-BPEL. Concepts such as BTAs may not be directly mapped to WS-BPEL but need to be expressed by more complex combinations of various elements. Possible WS-BPEL patterns to implement a given ebBP-ST choreography are also introduced in [9].

3.2. Conformance Notion - Why Bisimulation?

Conformance checking is not a new problem in the field of Web Services technologies, in fact, conformance checking techniques have been proposed for various combinations of choreography and orchestration languages. But as the notion of conformance is highly dependent on the context there does not exist a commonly accepted definition of the term conformance. A straightforward informal definition is: *An orchestration should be declared as conform to a choreography if all orchestration executions do not violate the predefined protocol.* As conformance checking problems are a specialization of the widely researched process equivalence problem, solutions from this domain can be used in order to specify the conformance notion more precisely. When looking at the message level, it is widely accepted that so-called *trace equivalence* which only analyzes the flow of exchanged messages is too weak to check conformance (e.g., [4]). *Bisimulation equivalence* [6], as another classical process equivalence notion, frequently is regarded as too strict for conformance checks. In most of the related work, the conformance notion distinguishes between incoming and outgoing message flows. An orchestration has to respect all incoming messages in order to be conform to the choreography. In case of outgoing communication, at least one (of possibly more) alternatives has to be implemented in the orchestration. An example is the ST "*receivedQuote*" in our use case, as two different BTAs ("*acceptQuote*" and "*requestQuote*") may be initiated by the potential customer here. The customer does not have to implement the possibility of performing the BTA "*requestQuote*" which means that he always performs "*acceptQuote*" and therefore all quotes have to be accepted (or the process is left by a timeout). Conversely the seller has to implement both execution possibilities as from his viewpoint it is not clear which BTA may be performed next.

However, for our conformance checking scenarios this conformance notion is not suitable as it cannot be applied to our integration architecture: We differentiate between the actual orchestration implementations in the so-called control processes and the existing backend systems that encapsulate business logic. Regarding the incoming information the already proposed conformance condition is not critical: The control processes must be able to react to all possible incoming events as they do not know how the partner will react. But the less strict requirements for outgoing information are only possible if the orchestration is directly deciding which steps should be performed next. In our architecture (cf. Fig. 1), the control processes delegate this decision to the backend systems which trigger the subsequent process flow. So, the control processes must be able to react to all possible backend decisions and must also be able to produce all allowed outgoing communication specified in the choreography. Therefore the conformance requires the rather strict notion of *weak bisimulation equivalence*.

The formal model used to perform conformance checks is the process algebra *Calculus of Communicating Systems* (CCS) developed by Milner [5]. For our purposes, the main aspects of CCS are sufficient to express our integration models: Processes (starting with a capital letter) are defined by an assignment. Linking to processes is done by

using the name of a predefined process in a process body. Sequences are built using the "." operator and choices may be created with the operator "+". Therefore, e.g., "`P1 = action1.((action2.P2)+(action3.P3))`" defines the process `P1` in which after performing action "`action1`" either "`action2`" is performed followed by the execution of the process "`P2`" or "`P3`" is performed after "`action3`".

3.3. ebBP-ST to CCS Transformation

The basic principle of transforming an ebBP-ST choreography model into a CCS representation is to define a CCS process for each ST and each end state used in the choreography whereas timeout events, the execution of BTAs and their evaluation in decisions will be represented by CCS actions. Algorithm 1 shows the concrete steps needed in order to transform ebBP-ST choreographies to CCS.

As shown in line 1-3, for each end state $t \in T$ a process is defined using the prefix "`END_`" followed by the name of the state. The process body simply contains a single action named by the corresponding end state in order to distinguish different end states and the CCS *empty* element "`0`". The actual creation of the process is performed by the method `CreateProcess(`*`processName`*`,` *`processBody`*`)` which creates a CCS process with the name *`processName`* containing the *`processBody`*.

In comparison to this, the transformation rules for STs are much more complex: As described before and in [9] a timer should be started when entering a ST. But it also should be possible to reenter an already visited ST without resetting this timer. In order to describe this behavioral difference, two different CCS processes for each shared state are created. An outer process definition "`ST_`*`STname`*" simply performs a `start_timer` action and then links to an inner process definition (named "`INNER_ST_`*`STname`*") which contains the actual control flow logic (stored in variable `processBody`) of a shared state (ll. 28/29, Alg. 1).

BTAs, Decisions and Timeouts are realized in CCS using actions. Therefore each possible BTA in each ST followed by the decision evaluation is added to the ST process body (Alg. 1, line 20) using the method `AddBTA(`*`processBody`*`,` *`btaToAdd`*`)` which combines the allowed BTAs with the "+" operator. BTAs are simply mapped by a single CCS action named like the BTA sequentially followed by the decision. The different decision branches are realized using the CCS choice operator ("+"): First, each guard is translated to a CCS action followed by a link to the CCS process of the subsequent ST and afterwards all of these constructs are combined using the `AddDec` method. A flag f indicates whether a timeout should be reset or not: If $f == \{tt\}$ (l.10) the outer process definition will be used, otherwise it will be linked to the inner process definition of the subsequent ST.

Applying the algorithm to our use case will clarify the output of the algorithm which is presented in listing 1: Line 1 shows the outer process definition "`ST_receivedQuote`", the following lines 2-11 describe the actual control flow logic: The two BTAs allowed and the timeout event are combined as a CCS choice. The execution of a BTA is represented by a CCS action (e.g., "`bta_requestQuote`" in line 7). The decision after this BTA evaluates whether the BTA has been a "*BusinessSuccess*" (BS) or a "*TechnicalFailure*". If the BTA was successful the ST "*receivedQuote*" should be reentered and the timer should be reset. Therefore the referenced CCS process is "`ST_receivedQuote`" (line 9 in List. 1). As the timer should not be reset in case of a "*TechnicalFailure*", the internal process definition "`INNER_ST_receivedQuote`" is referenced in line 8.

The other STs in the use case are transformed in the same way.

Listing 1. CCS Representation of the Use Case (Excerpt)

```
1  ST_receivedQuote = start_timer.INNER_ST_receivedQuote
2  INNER_ST_receivedQuote =
3  (bta_acceptQuote.(
4    (dec_acceptQuote_TF.INNER_ST_receivedQuote)+
5    (dec_acceptQuote_BS.ST_concludedContract)
6    )
7  )+(bta_requestQuote.(
8    (dec_requestQuote_TF.INNER_ST_receivedQuote)+
9    (dec_requestQuote_BS.ST_receivedQuote)
10   )
11 )+(timeout.END_TechnicalFailure)
```

3.4. WS-BPEL to CCS Transformation

The aim of transforming WS-BPEL to CCS is to check whether the WS-BPEL implementations conform to the predefined ebBP-ST choreography models. As mentioned before, we are not interested in low-level formal models but in a representation which allows for this check. For example, there is no need to model the concrete implementation of a BTA including all WS-BPEL *sequences*, *scopes*, *invokes*, etc. in CCS because for our purposes it is only relevant whether a BTA may be performed in a ST or not. So, the most important task for transforming WS-BPEL to CCS is to detect the used patterns which express the different concepts of ebBP and afterwards transforming the sequence of patterns into CCS. When looking at an ebBP *decision*, a possible WS-BPEL implementation pattern may send the previously exchanged *Business Document* to the backend systems (*invoke* statement) which analyze the outcome and the result is stored in a variable (*assign*). Afterwards in a

Algorithm 1: ebBP to CCS Transformation

Input: A valid ebBP-ST choreography $(R, (\{s_0\} \cup ST \cup SBTA \cup DEC \cup T), G, \phi, \theta)$ to be transformed
Output: A CCS representation of this choreography
Algorithm:

```
1  foreach t in T do
2  |  CreateProcess("END_" + t.name, "end_"+t.name+".0")
3  end
4  foreach st in ST do
5  |  processBody = " ";
6  |  foreach (st, {tt}, bta) in θ do
7  |  |  decisions = " ";
8  |  |  foreach (bta, {tt}, dec) in θ do
9  |  |  |  foreach (dec, g, f, nextST) in θ do
10 |  |  |  |  if (f == {tt}) then
11 |  |  |  |  |  AddDec(decisions, "(dec_"+dec.name+"_"+g.name+".ST_"+nextST.name+")")
12 |  |  |  |  else
13 |  |  |  |  |  AddDec(decisions, "(dec_"+dec.name+"_"+g.name+".INNER_ST_"+nextST.name+")")
14 |  |  |  |  end
15 |  |  |  end
16 |  |  |  foreach (dec, g, t) in θ do
17 |  |  |  |  AddDec(decisions, "(dec_"+dec.name+"_"+g.name+".END_"+t.name+")")
18 |  |  |  end
19 |  |  end
20 |  |  AddBTA(processBody, "(bta_"+bta.name+".("+decision+"))")
21 |  end
22 |  if ∃(st, g^to, nextST) in θ then
23 |  |  processBody += "+(timeout.ST_"+nextST.name+")"
24 |  end
25 |  if ∃(st, g^to, t) in θ then
26 |  |  processBody += "+(timeout.END_"+t.name+")"
27 |  end
28 |  CreateProcess("ST_"+st.name, "start_timer.INNER_ST"+st.name)
29 |  CreateProcess("INNER_ST_"+st.name, processBody)
30 end
31 CreateProcess("Start", "ST_"+firstST.name)
```

series of *if* statements it is checked which result has been evaluated and the switch to the next state is initiated.

After all such patterns and their ordering have been identified, the transformation to CCS is similar as for ebBP-ST. In case of the decision example, each decision branch is transformed to a CCS sequence, containing an action indicating the decision, followed by a reference to the CCS process of the next state.

3.5. Checking the Conformance

After transforming the ebBP-ST choreography model and the two corresponding WS-BPEL orchestrations to CCS, the actual conformance checks can be performed. Therefore each CCS orchestration model has to be checked against the choreography representation for bisimulation equivalence. The result of each bisimulation equivalence check is (see Fig. 3) the binary answer whether an orchestration is conform to the choreography definition or not.

An advantage of this checking approach is that the two different implementations may be analyzed independently, i.e., it is not necessary to analyze the implementation of a partner who possibly does not want to publish his internal orchestration models.

Note that CCS is supported by various model checking tools (e.g. the *Edinburgh Concurrency Workbench* (CWB)[1])

[1] available at: `http://homepages.inf.ed.ac.uk/perdita/cwb/`

and therefore the check can be automatically executed using such a tool. Using the proposed CWB model checker we are able to prove the conformance of the WS-BPEL implementations as well as to detect various intentionally produced faults in orchestrations.

4. Related Work

Conformance checking problems have been investigated for various choreography and orchestration languages:

For example the work of Baldoni et al. (i.a.,[1]) is rather generic and uses automata representations to model choreographies and orchestrations. Martens ([4]) discusses rather extensively suitable conformance notions for conformance checking problems and proposes a formalization of abstract and executable WS-BPEL processes using Petri Nets. A proposal for the *Web Services Choreography Description Language* (WS-CDL) has been developed by Foster et al. ([3]) who use *Finite State Processes* (FSP) to represent WS-CDL and WS-BPEL. The actual check is based on *trace equivalence*. The only work that we are aware of which is dealing with conformance checking ebBP and WS-BPEL is [12] which uses the process algebra *Communicating Sequential Processes* (CSP) to perform so-called traces refinement to check the conformance. The common problem of all these approaches is that they do not use the strict conformance notion we propose in this work. As we have shown above the proposed weaker notions are suitable for the contexts considered in the various papers, but it is not appropriate for the integration scenario we assume.

Another approach which uses a stricter conformance notion when checking WS-CDL choreographies is presented in [11]: The authors derive a new formalism called piX-Model to formally represent WS-CDL choreographies. The piX-Model is based on the well-known π-calculus and open bisimulation is used as conformance notion.

Apart from conformance checking there exist various approaches to formalize WS-BPEL (see [10] for an overview). A WS-BPEL formalization using CCS is presented in [2]. Generally these formalizations are not perfectly suitable for our approach because they provide direct mappings for basic WS-BPEL constructs while we concentrate on those patterns that realize ebBP concepts.

5. Conclusion and Future Work

In this paper, we have introduced an approach for checking the conformance between ebBT-ST choreographies and corresponding WS-BPEL based orchestrations. The key elements of this approach are the proposed transformation algorithms to CCS which allow for a common representation of ebBP-ST and WS-BPEL in order to perform the actual conformance check by a bisimulation equivalence check. Preliminary results show the general applicability and correctness of our approach.

Ongoing research now concentrates on relaxing the requirements concerning the structure of the WS-BPEL processes by using existing WS-BPEL semantics. Furthermore we are working on better usability, such as integrating the proposed tool chain to improve the work flow and direct highlighting of detected conformance issues in the original ebBP-ST and WS-BPEL definitions.

References

[1] M. Baldoni, C. Baroglio, A. K. Chopra, N. Desai, V. Patti, and M. P. Singh. Choice, interoperability, and conformance in interaction protocols and service choreographies. In *8th Int. Joint Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009), Budapest, Hungary, May, 2009*, pages 843–850. IFAAMAS, 2009.

[2] J. Cámara, C. Canal, J. Cubo, and A. Vallecillo. Formalizing wsbpel business processes using process algebra. *Electr. Notes Theor. Comput. Sci.*, 154(1):159–173, 2006.

[3] H. Foster, S. Uchitel, J. Magee, and J. Kramer. WS-Engineer: A Model-Based Approach to Engineering Web Service Compositions and Choreography. In *Test and Analysis of Web Services*, pages 87–119. Springer, 2007.

[4] A. Martens. Consistency between Executable and Abstract Processes. In *2005 IEEE Int. Conf. on e-Technology, e-Commerce, and e-Services (EEE 2005), Hong Kong, China*, pages 60–67. IEEE Computer Society, 2005.

[5] R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer, Berlin, Heidelberg, 1980.

[6] R. Milner. *Communication and concurrency*. Prentice Hall, Harlow, 1989.

[7] OASIS. *Web Services Business Process Execution Language Version 2.0 (WSBPEL)*, 2.0 edition, April 2007.

[8] OASIS. *ebXML Business Process Specification Schema Technical Specification*, v2.0.4 edition, Oktober 2006.

[9] A. Schönberger, C. Pflügler, and G. Wirtz. Translating shared state based ebXML BPSS models to WS-BPEL. *Int. Journal of Business Intelligence and Data Mining*, 5(4):398–442, 2010.

[10] F. van Breugel and M. Koshkina. Models and Verification of BPEL. Unpublished Draft, Available at: `http://www.cse.yorku.ca/~franck/research/drafts/tutorial.pdf`, September 2006.

[11] G. van Seghbroeck, B. Volckaert, F. D. Turck, B. Dhoedt, and P. Demeester. Web service choreography conformance verification through the pix-model. *Int. J. Cooperative Inf. Syst.*, 19(1-2):1–30, 2010.

[12] W. L. Yeung. A Formal Basis for Cross-Checking ebXML BPSS Choreography and Web Service Orchestration. In *APSCC '08: Proc. of the 2008 IEEE Asia-Pacific Services Computing Conf.*, pages 524–529, Washington, DC, USA, 2008.

Proactive Problem Management and Event Correlation

Werner Zirkel
Siemens AG
Healthcare IT
Henkestr. 127, 91052 Erlangen, Germany
werner.zirkel@siemens.com

Guido Wirtz
University of Bamberg
Distributed and Mobile Systems Group
Feldkirchenstr. 21, 96047 Bamberg, Germany
guido.wirtz@uni-bamberg.de

Abstract

In the service business, event correlation systems are used to predict and avoid system failures. Strategically, this may contribute to higher system availability and better service activity planning. Companies that want to benefit from this competitive advantage often meet two fundamental problems: the complexity of service monitoring increases with the number of correlation patterns. At the same time, the cost transparency for the whole process is lost. This paper shows on a business process level why these problems arise and how they can be avoided.

Keywords: *Service Management, Event Management, Event Correlation, Predictive Maintenance*

I. INTRODUCTION

Manufacturers of hardware systems are often faced with enormous cost pressure. In some industrial sectors, the profit achieved by selling the hardware device itself becomes very limited. In this case, a value-adding service organization may help to differentiate from others. An important efficiency criterion of a service organization is the degree of automation in the case of a system failure. In this context, a change is happening in many industrial domains: *reactive* service or "waiting for the failure" is no longer seen as an adequate support by customers. By using event correlation systems, it is possible to make *predictive* statements about imminent system failures. By predictability, failures can be prevented (*predictive/preventive maintenance*). This may contribute to higher customer satisfaction. At the same time, additional customer-specific contract options may be set up, such as an agreement of a guaranteed system availability level. The possibility to plan service activities may increase, which implies a positive impact on personnel costs and material costs. At best, predictable system failures can be handled within the standard planning processes. However, the use of complex failure patterns/correlations may cause high development costs. This challenge can be faced on a technical level by automization for example by using data mining systems and machine learning methods [1]. In the long run the uncontrolled creation of a huge set of failure patterns may cause problems by increasing complexity of configuration, deployment and execution. With increasing handling, the pattern efficiency control gets harder. Thus, many companies are faced with the question: how do we have to structure our operational service processes to handle these problems?

II. SERVICE MANAGEMENT

It is possible to investigate a service management system from two different views. On the one hand, the system consists of business management targets. The "top-down" view which is deduced from the business core tasks and the definition resulting from the value-adding business processes is called *business service management* [2]. On the other hand, there exists a "bottom-up" IT-view on a service management system. From this perspective, the formal goal of a service management system is providing the right services for the business. Those services may be combined in IT Service Level Agreements. On a process level, adequate IT processes have to be defined. To certify an efficient service organization, the regulation ISO/IEC-20000 was created. It is divided in two parts. While part 1 defines service management basics, the second part proposes structures which should be used in service management organizations [2]. The requirements which are derived from ISO 20000 can be implemented using the IT Infrastructure Library (ITIL). Both methods are based on the British standard BS15000 and for this reason, most ISO 20000 definitions may be found in the ITIL framework, too. ITIL was created by the English Office of Governance Commerce (OGC) as a collection of "good practices" to ensure an efficient IT structure. This definition implies that its basics are not derived scientifically. In fact, ITIL proposes one way how to structure service processes. There exist similar approaches like COBIT [3], MOF [4] eTOM [5]. The basic idea is to create an IT organization which acts as a uniform interface between customers and the company sectors IT development and operation. Therefore, ITIL defines a generic process model, which includes the major service processes. ITIL is structured in a lifecycle model in the core sectors service strategy, service design, service transition and service operation [6]. Additionally, the idea of continual service improvement was introduced in the actual ITIL version V3. Service Strategy includes processes for strategic planning of an efficient IT organization. In detail, this sector consists of

the processes financial planning, portfolio planning and demand management. Service design and service transition primarily handle the development and roll-out of new services. On the design level, service managers arrange Service Level Agreements (SLAs) with customers. To keep these agreements, capacity, security and suppliers has to be managed. New IT services are brought into the field by transition processes like application development, change management, release and deployment and testing. Service operation deals with services which are already used. ITIL defines event management, incident management, problem management, request management and access management. In the following, this paper takes a closer look of what events, incidents and problems may look like and how they fit together.

A. Event Management

An *event* is a change of state of an object which has significance for the service. Event management is the activity of observing objects, identifying such events and categorizing them [10]. Usually, an event management process is used to implement a remote monitoring service, which could be reactive and/or proactive. The architecture of event management platforms is usually based on the manager-agent-model [7]. A manager takes the role of a central distributor/collector's node. On each connected device ("managed node"), a software agent is installed. The agents receive tasks and send information on the device status and the workflow to the manager. Historically, the idea of event management can be tracked to the first approaches of remote monitoring of network devices [8]. Over time, the networks sector became bigger and more complex. Also, the remote monitoring service was expanded. To structure the growing complexity, literature differentiates between the management levels component management, system management, application and enterprise management [9]. *Component management* refers mainly to the handling of network components, i.e. routers or switches. *System management* focuses on more complex nodes with a set of several sub-components. System management organizes single distributed functions. Thus from an outside view the systems behave like one (virtual) system [9]. For service, the component set is considered as an integrated system where a single failure may no longer depend on one single component, but also on a complex connection between several components. If the component set is based on software functions, the level *application management* may be separated [9].

The event management definition assumes that events are created in a continual process. Event management agents have to read these events (event detection) and to filter and categorize in simple groups. For example, ITIL suggest errors, warnings and informational events. In many cases, the total amount of events is rather big; therefore, a manual analysis by humans is usually not possible or (economically) not reasonable because of complexity. For this reason, event correlation systems are used.

Event correlation systems use logical, temporal and statistical functions to model semantic connections between events. By aggregating a set of events, one new event may be created containing a better information quality. This reduces the effort for (human) analysts. The reference process defines a trigger mechanism to escalate critical events. The alarm is handled in the incident management process. Before the event is being discarded, the service organization may start an event review to assure the correctness and the efficiency of correlation patterns. The following figure shows the activities of the ITIL reference process event management.

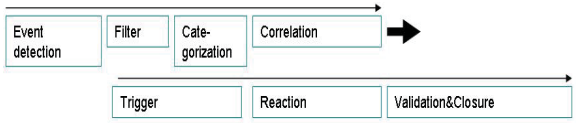


Figure 1. event management activities, according to [6]

B. Incident Management

An *incident* is an unplanned failure or quality reduction of a service. Incidents are handled in a incident management process. They may be identified in several ways, i.e. by customer information, a service engineer's report or via the event management process. According to ITIL, incidents may be handled by a service desk. This function acts as a single point of contact to the customer. Usually, the incident management uses a ticketing system. One important task is the categorization by impact and urgency. Urgent incidents with high impact are marked as "*major incidents*". In this context, the roles first, second and third level support are proposed: if a ticket cannot be solved quickly by a first level diagnosis, it may be escalated to a second level, where better a qualified service team will try to solve the problem. If the second level fails, the problem may be forwarded to a third level expert team. The incident management is usually controlled to ensure that all SLA requirements are kept in time. The following figure shows the activities of the ITIL reference process incident management.

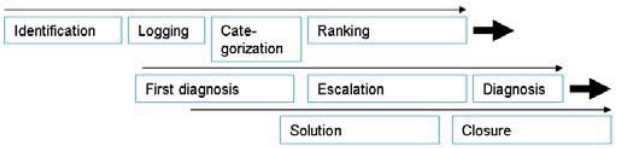


Figure 2. incident management activities, according to [6]

C. Problem Management

A *problem* defines an unwanted situation which is considered as trigger for one or more (active or potential) incidents. A problem causes at least one incident. If the root cause of a problem is known, the problem is called *known error* [11]. The focus of incident management is primarily set on the elimination of actual failure symptoms. Instead, problem management is focussed on finding a final solution which removes the root cause of all problem-related incidents. In comparison to incident management, it does not depend on a rigid, customer-specific time plan. The problem management process structure resembles to incident management. Once a problem was defined, categorized and prioritized, a workaround may be defined which tells the incident management team how to act on upcoming incidents.

Identify applicable sponsor/s here *(sponsors).*

Additionally, the problem is stored in the *known error database*. The problem is closed when a final solution is provided as a result of a *request for change*. Problem management can be handled in a reactive and a proactive way. *Reactive* problem management starts after a set of incidents occurred. A *proactive* problem investigation uses operative service data sources and generates analyses and reports to minimize the impact of potential future problems. Beside the incident records, event management may be an important source and tool for proactive problem management. If a problem is detected early, a *workaround* may be defined. This workaround could i.e. be an event pattern, which indicates an upcoming incident based on that specific problem. By this means, an incident may be prevented or its impact may be minimized. The following figure shows the activities of the ITIL reference process problem management.

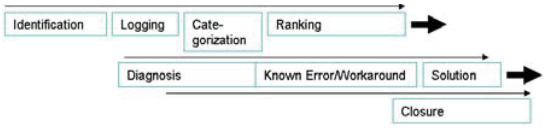


Figure 3. problem management process, according to [6]

The ITIL framework is originally meant to be an IT service approach. However, incident management and problem management are generic processes. Moreover, event management is usually done as a remote service, which is a distinct IT product. Thus, the ITIL framework is also applicable outside of the plain IT business.

III. CASE STUDY - MEDICAL DEVICE SERVICE

To give an idea to how the processes of event management, incident management and problem management can be linked together, we present a case study which focuses on a sample of the actual structure of a service organization which sells medical devices and associated remote services. The case study focuses on the three service operation processes described above.

The service organization is a provider of various products in the field of medical technology. The product range includes imaging systems from different modalities, such as nuclear medicine, computed tomography, magnetic resonance imaging, angiography and ultrasound. In most cases, the systems are complex and service-intensive objects. The healthcare service organization structure implements the service operation processes event management, incident management and problem management.

A. Event Management

The organization uses a management platform (Common Platform Remote Service Platform *cRSP*) to provide a set of IT services, such as *software distribution*, *inventory management* and *event monitoring*. The event monitoring service implements the event management process and includes among other components the software as HP OpenView Operations (HP OVO). HP OVO provides an event agent which reads events from different data sources and filters them by special rules (*policies*). For event correlation, HP OVO provides three mechanisms. On the one hand policies can be used to generate simple event correlations. Additionally, the Event Correlation Services (ECS) Composer and Event Correlation Services (ECS) Designer may be used to perform complex correlations. ECS Designer correlations are known as *circuits*. In comparison to ECS Designer, ECS Composer offers limited functionality for event correlation [12]. The HP OVO Event Agent is able to send alerts/messages to a back-end HP OVO Event Management Server. There, the alerts can be monitored by device experts in a Regional Support Center (*RSC*) using HP OVO Java Console.

B. Incident Management

Incidents may be created based on a customer notification. In this case, the service organization acts reactive, since such measures can be initiated only after the perception of the issue by the customer. Reactive messages are processed by a service desk team called Uptime Service Center (*USC*). To perform a classification, the current situation is pre-clarified by device experts or solved via cRSP remote access. If it is not possible to solve the problem in a given time frame, the Regional Support Center (*RSC*) and Headquarter Support Center (*HSC*) are being involved as second and third level support. If necessary, a service technician is being sent. Incidents are solved and completed by the USC. The service organization acts *proactively* if incidents are reported before they are found by the customer. Proactive tickets can be generated using event correlation patterns. The RSC provides a monitoring team that checks the incoming messages and evaluates them. If onsite actions are necessary, the RSC forwards this information to the USC which dispatches a service engineer. Proactive tickets are closed by the RSC. Alarms, which indicate an upcoming system failure with high confidence, will automatically generate a new incident.

C. Problem Management

The service organization offers a variety of different products. To analyze the associated problems, domain knowledge is necessary. Therefore, the function-oriented organization structure is replaced by an object-oriented modality organization. In order to present a consistent case study, we refer to one of the modalities.

To investigate the root causes of frequently recurring incidents, information from the operative business processes is obtained from a data warehouse system. In addition, workflow data from the devices and from spare part logistics are included. Personnel and material expenses are compared with respect to various temporal, geographic, systems and customer-specific characteristics. There exists a set of criteria to decide whether to open a problem ticket or not, including equipment availability, degree of market introduction, effort and cost criteria. When a new problem ticket is being opened, an activity plan is being developed, resulting in a final problem solution. In many cases, additional organizational units are being involved, such as the product development. The problem management is organizationally separated from the incident management and is therefore not subject to downtime-based time-limits. However, there may exist strict time limits in the problem management, especially when a product when new products are introduced to the market.

IV. A PROPOSOAL FOR INTEGRATING EVENT CORRELATION SYSTEMS INTO PROACTIVE PROBLEM MANAGEMENT

ITIL does not make an explicit statement on how the interaction of event management, incident management and problem management has to be organized, as this depends substantially on internal structure of each service organization and on external conditions. If the organization's processes are not coupled properly, two problems in the event management process may arise.

- *Complexity problem*

The more correlation patterns are generated, the higher is the cost handling, maintaining, improving them. This increases the total costs of the event management process. The more correlation patterns exist, the more messages are generated on the management server. This makes the monitoring task more difficult and therefore, the system may come to a point where an appropriate in-time reaction is no longer possible.

- *Transparency problem - missing business case*

Another problem arises when correlation patterns are created which cannot be associated with a particular issue. Such correlations may support the incident management in a short-term manner. However, it may become hardly possible to set up a cost and efficiency controlling mechanism. Thus, the added value of the monitoring service can hardly be calculated and the event management is in a permanent need for justification.

A. Problem identification and categorization

The goal of the following approach is trying to prevent the emergence of these problems or at least minimize their impact by integrating incident management, problem management and event management. The approach should be perceived as a best-practice-proposal for *one* possible connection of the three processes and does not exclude other solutions or existing approaches. The basic idea is to introduce proactive problem management based on event correlations. In particular, major incidents (system failures) shall be avoided which lead to high service costs. First, a problem ticket has to be opened, based on existing, historical data. As a basis for an analysis, the number of incidents within a specific time frame may be used. The incidents should be grouped by their associated material data. To identify material defects as root causes for this group of incidents it is reasonable to include the spare part consumption data in this analysis. By doing so, a filter mechanism may be created which groups only homogenous material problems. This approach presumes that incidents have been created and categorized properly. Similarly, the spare parts consumption has to be recorded. Defective parts need to be investigated in case of service in order to determine the underlying material defect. Wherever it is possible, new incidents are assigned to previously identified problems. With this connection, patterns may be identified easier; moreover, the information can be used in a problem management controlling.

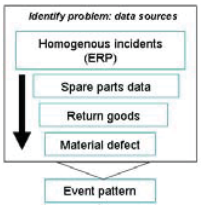


Figure 4. *Pattern identification based on material problem*

B. Problemprioritization

By referring a problem ticket to a material defect, an individual cost analysis can be done for each pattern. Therefore the mean emergency extra charge of service organization has to be identified. Extra costs for device failures include for example additonal personnel costs, emergency spare part logistics costs, the cost of consequential damages or penalties for not reaching a certain service level. By linking the extra charges costs for one failure with historical incident data, the potential savings may be extrapolated (assuming an equal number of service calls in period t and t +1).

- Potential savings = number of incidents (based on the same defect) * mean emergency extra charges

Since the approach relates to imminent device failures, the number of incidents may also be used for the calculation of additional usage time (uptime). By preventing a system failure the customer saves the time for the restoration of service, therefore, the incidents of a period may be set in a relation to the average time to resolve an incident (Mean Time to Restore Service, MTRS), which is basically:

- Potential additional uptime = Number of incidents (based on the same defect) * mean time to restore service

Based on this cost analysis, the problem tickets may be prioritized. Assuming that the number of material defects in t1 is equal to the number of material defects in the incidents of the period t0, the incidents of t0 are analyzed. The ranking may relate to the quality improvement (the increase of potential savings). If product quality is used instead, the potential uptime may be used as decision criterion. If both goals are important to the service management, a pareto-optimized decision may be used. The following table shows an example.

Materialdefekte	Anzahl Incidents Periode t_0	Einsparungspotenzial (EUR)	Uptime-Potenzial (Stunden)
Defekt A	152	325.000	2200
Defekt B	86	254.000	1900
Defekt C	32	175.000	700
Defekt D	32	173.000	750

Figure 5. Problem ranking based on potential cost savings from a previous period.

The transparency provides several advantages. The prevention of failures and the uptime potential may contribute to higher customer satisfaction. The improvement in quality of service enables the organization to create new service level agreements, based on a guaranteed uptime level. Therefore it is necessary to prevent a majority of the defects with high uptime potential through proactive correlations. At the same time, unprofitable correlation patterns are excluded already in the planning phase. The effort for pattern creation can be set into

relation to the potential cost savings, so the event management process may be perceived as a value-adding service.

C. Problem diagnosis and workaround

By referring a problem ticket to a (hardware-based) material defect patterns may be identified and future system failures may be predicted. In this sense, a pattern is part of the proactive service strategy. For problem management, the pattern serves as a workaround to an impending major incident. It is therefore not to be regarded as a solution to the problem because the actual cause of the incident is not affected. By defining a pattern as workaround, its usage is limited in time. The solution of the problem itself- and the replacement of the work-around - is the improvement of the material or the introduction of an improved component version. To use these workarounds, patterns have to be identified first. As we have already shown in [13], the pattern identification process may be standardized and supported by data mining tools. A prerequisite is that a certain material defect shows up in the same manner in the event data. The data analysis bases on the deviation between failure data compared to normal operation data of the device. Besides counting single events, those events that occur exclusively or more frequent in the case of failure are connected to sequences ("*semantic block*s") and used as part of an event correlation. Additionally, sensor data may be used. If an abnormal situation has been detected, the event correlation system raises an alert. Further follow-up actions, such as automatic incident generation can be performed. The activities incident logging, categorization, ranking and initial diagnosis can be automated in many cases, because it the expected failure is already known (by pattern definition). Since the event correlation indicates a specific defect, it is possible to start product-specific actions, such as staff planning spare parts ordering.

D. Problem solution

The actual problem solving addresses the cause of the defect and is not part of the incident and event management process. The problem ticket is closed if the solution was rolled out to all service objects. With the closure of the problem tickets, the event correlation is obsolete as a workaround and can be removed from the event correlation system. The efficiency of event correlations can be controlled. As a prerequisite, the incident tickets have to be linked to the corresponding problems.

- Event correlation efficiency = number of incidents proactively processed / number of total incidents (related to the incidents that have been associated with this problem in context)

By using temporary event correlations and controlling mechanisms it is possible to prevent a steady increase of the total correlation set. Thus, a lack of transparency and complexity can be avoided.

V. CASE STUDY - USING PROBLEM-BASED EVENT CORRELATIONS IN MEDICAL DEVICE SERVICE

In the service organization, event correlation patterns are defined by the serviceability group. The pattern development is mostly based on product specifications that are given by product development. For example, error messages for certain components are defined by external suppliers and integrated into event correlation in the form of simple correlation rules. Other correlations are based on empirical knowledge. For that, single incidents are analyzed. Error messages that may be associated with the incidents are summarized in event correlations. The service organization is using some statistical methods to assess the performance of new event correlations before they are used. In a few cases, event correlations are generated based on problem tickets. On the whole, the organization produces a growing number of event correlation patterns. Without a consistent efficiency controlling mechanism, this leads to growing complexity of the correlation set. From a cost perspective, there may occur the risk that the event management process is questioned, if the overall process costs get to high. By introducing the problem-based event management approach, this risk could be reduced. By reducing the number of patterns, the costs for implementation, distribution and operation may be kept as low as possible. This effect may be intensified by a better efficiency controlling. By bringing the data from actually performed service calls together with the number of associated alert messages, a sensitivity and specificity rate of each individual pattern can be determined. The problem-impact may serve as a basis for the importance of event correlation.

Example: within the period 2008 - 2009 a part of the organization reported 119 incidents. These incidents are caused by one specific device problem. Assuming that a correlation pattern would have detected 70% of these incidents proactively, 83 cases would have been predictable. For this specific problem type, three important cost drivers were identified, that is cost of emergency orders, personnel costs and costs for operations with multiple onsite presence.If the pattern would have been used in the given time period, this would have led to cost savings of approximately 24,500 €. It is possible to extrapolate this potential for worldwide use, assuming a constant or increasing incident frequency in the future. This calculation is the basic motivation for the development of an event correlation. Besides the cost analysis, the device downtime has to be considered. For this purpose, the mean time to restart (mean time to recover) is calculated for this use case and multiplied by the total number of failures. If the solution of a problem requires onsite technical service and / or material usage, the problem is often a medium or long term issue, because the changes have to be rolled out to all devices in the field. Therefore, the product support team may set a workaround (a correlation pattern) which is available faster. The involved incident teams USC, RSC and HSC take advantage of such patterns in several ways. First, proactive alerts may lead to higher device availability, so unplanned service activities may be reduced, which improves the planning and dispatch of service engineers. The organization may provide detailed handling options, because they know which defect is about to occur. Depending on the classification

precision of the pattern, the incident management process activities *identification, logging, categorization, ranking, first diagnosis and escalation may be automated, as the figure below shows*. The diagnosis itself tries to confirm the pattern-based failure expectation and follows the given handling options.

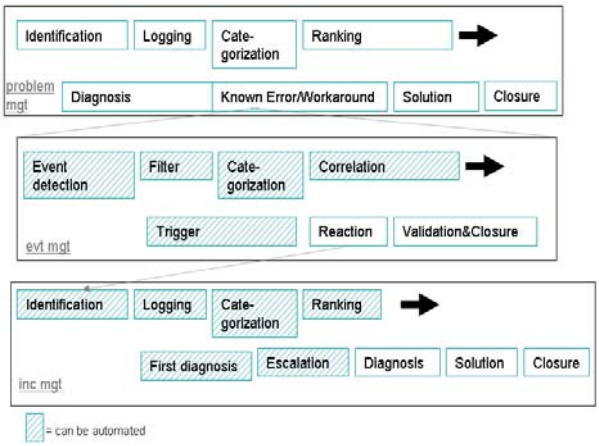


Figure 6. Proposed process structure. Evt. correlation as workaround

The introduction of problem-based event correlations may have a positive impact on a number of service-relevant indicators. By partial automation of incident management, the cost of troubleshooting for proactively detected incidents (Mean Time To Repair, MTTR). By clear description of the fault service calls can be reduced, fixed by the First Visit Rate (FVFR), or the relative number of successful one-off operations can be measured. In the best case, and the index Site Visit Avoidance (SVA) will be positively impacted by an increase of cRSP Remote Access operations. From the perspective of the customer results from the use of proactive correlations increased system availability, which prevented on the basis of the problem based on the downtime incidents (Mean Time to Restore Service MTTRS) can be calculated. The introduction of problem-based event correlation may have a positive impact on a number of service-relevant indicators. By (partial) automization of incident management activities, the cost of troubleshooting my be reduced, which could be measured by the performance indicator Mean Time To Repair, MTTR. A clear description of the failure may help to reduce multiple onsite visits, which can be expressed by the performance indicator First Visit Rate (FVFR). In the best case, the performance indicator Site Visit Avoidance (SVA) will be positively impacted, with a parallel increase of remote access operations. From the perspective of a customer, proactive actions increase the device availability, which may be expressed by the performance indicator (Mean Time to Restore Service MTTRS).

IV. CONCLUSION

Based on the ITIL v3 framework, the service operation processes incident management, problem management and event management were described. The presentation of a medical device service organization was used as an example how these processes may be implemented outside of the "IT world". Based on the reference processes, we presented an integrated process model for problem-based event correlation. The goal of this approach is to avoid the complexity and transparency problem. These problems are a logical consequence when using a "classical" approach, based on single incidents and product development knowledge. However, the problem-based approach implies that historical device data is available. So, it has to be complemented by a correlation strategy for first-time defects/new devices. Also, the approach focuses on hardware defects, where defects can be tracked down to exactly one device failure. This prerequisite may not be given for software failures [14]. By applying the approach to the case study service organization, we showed a possibility how to implement the model and what benefits would arise from its introduction. As a conclusion, we presented key performance indicators which could be used to prove the efficiency of problem-based event correlations.

REFERENCES

[1] Flatin/Jakobson/Lewis: Event Correlation in Integrated Management: Lessons Learned and Outlook, 2007, in:Journal of Network and Systems Management,Vol. 17, No. 4

[2] Wischki: ITIL V2, ITIL V3 und ISO/IEC 20000,1.Aufl.,Hanser Fachbuchverlag,Munich, 2009

[3] Control Objectives for Information and Related Technology; http://www.isaca.org/cobit; last access:04.05.2010

[4] Microsoft Operations Framework 4.0; http://technet.microsoft.com/en-us/library/cc506049.aspx; last access:04.05.2010

[5] Hanemann: Refining ITIL/eTOM Processes for Automation in Service Fault Management, 2007 in:Proceeding of the 2nd IFIP/IEEE International Workshop on Business–Driven IT Management

[6] itSMF: IT Service Management basierend auf ITIL v3: Das Taschenbuch,2. Aufl.,Van Haren,Zaltbommel, 2008

[7] ISO/IEC 10040 : 1998 E - Information technology-Open Systems Interconnection-Systems management Overview, ISO, Geneva, 1998

[8] Hegering/Abeck/Neumair: Integriertes Management vernetzter Systeme: Konzepte, Architekturen und deren betrieblicher Einsatz,dpunkt,Heidelberg, 1999

[9] Hegering/Abeck: Integriertes Netz- und Systemmanagement, Addison Wesley Verlag,Bonn,1993

[10] ITL 2010 :ITIL Glossary, http://www.itil.org/en/glossar/glossarkomplett.php?filter=E, last access:15.03.2010

[11]Ebel: ITIL V3 Basis-Zertifizierung: Grundlagenwissen und Zertifizierungsvorbereitung für die ITIL Foundation-Prüfung, Addison Wesley Verlag,Munich,2008

[12] ECS Designer,HP(2010), http://www.openview.hp.com/products/ecs/ds/ecs_ds.pdf, last access:15.03.2010

[13] Zirkel/Wirtz: A Process for Identifying Predictive Correlation Patterns in Service Management Systems, Proc. 7th International IEEE Conference on Service Systems and Service Management (ICSSSM 2010) , Tokyo, 2010

[14] Hanemann: Automated IT Service Fault Diagnosis Based on Event Correlation Techniques, Dissertation, LMU Munich, Fakultät für Mathematik, Informatik und Statistik, Munich, 2007

A Regression Test Technique for Analyzing the Functionalities of Service Composition

Huiqun Yu[1,2], Dongmei Liu[1], Guisheng Fan[1], Liqiong Chen[3]
[1] Department of Computer Science and Engineering
East China University of Science and Technology, Shanghai 200237, China
[2]Shanghai Key Laboratory of Computer Software Evaluating and Testing, Shanghai 201112, China
[3] Department of Computer Science and Information Engineering
Shanghai Institute of Technology, Shanghai 200235, China

***Abstract*—Regression testing is a hot research area for Web service composition, which has direct impact on QoS of applications. This paper proposes an aspect-oriented regression test technique for analyzing functionalities of service composition. An extended version of Petri net is proposed as the underlying formalism. Services and components are modeled by Petri nets, service evolution is specified as crosscutting concerns, and in turn woven into the testing model of service composition by following weaving rules. A regression testing algorithm of service composition is designed. Theories of Petri nets help prove the effectiveness of the testing method. A case study shows that the approach simplifies the process of testing, and improves its efficiency as well.**

***Index Terms*—Service composition; regression test; aspect orientation; Petri net; formal method**

I. Introduction

Service oriented computing (SOC) is a paradigm for building applications by selecting and composing software resources or services available on the Internet[1]. Web services are considered an implementation of *SOC* that facilitates the integration of heterogeneous applications. Web service testing is important because it helps evaluate various quality attributes, and prevent late detection of errors. However, some specific requirements need to be carefully considered: (1) The relationship between the service providers and requesters is highly unstable. Therefore, the testing method must be dynamically adjusted to different services and users according to the actual environment; (2) Web service testing is based on its interface, so the system must have sufficient flexibility to dynamically weave the test modules. In addition, service composition involves a variety of standard protocols, which brings new challenges to system testing.

Several techniques have been developed to test Web services, which includes unit testing, integration testing, black box testing and white box testing [2,4]. In [2], the BPEL specification is transformed into an intermediate format model that is based on timed automata. A test generation framework for BPEL composition is proposed in [3], where BPEL semantics was defined by Web service automata. Many works in Web service testing are in code level. However, the source code of a service is usually inaccessible, which limits applicability of those techniques. A large number of coverage-based test case reduction techniques have been proposed by using statement-coverage, branch-coverage, or define-use association criterion[5,6] . Meanwhile, model-based testing is also getting more and more attention both in industry and academia as a test prioritization approach[7−9]. However, these techniques prioritize test cases by means of code coverage achieved, such as the number of statements or branches covered by individual test cases. However, the internal logic and states of a service are encapsulated by the service boundary, which cannot be realized effectively in service-oriented testing.

Aspect-oriented programming (*AOP*)[10] is a software engineering approach to separation of concerns, which is capable of capturing dynamics of system evolution. Moreover, formal methods can be used to enhance *AOP* with a solid foundation. This paper proposes an approach to testing the functionalities of service composition, and proposes a composition Net(*CN*) by extending the formalism of Petri net[11]. Different components of service composition are modeled by *CN*. *AOP* is used to abstract the evolution process of service composition as the crosscutting concerns, and weaves into the testing model of service composition. Based on this, the regression testing algorithm is given, the operation semantics and related theories of Petri nets help prove its effectiveness. A case study shows that the approach can effectively simplify the testing process and improve efficiency of testing.

II. Service Composition Modeling

A. Aspect-oriented service model

An application system is composed of a number of independent components, and each component can be implemented by a bunch of available services. To capture dynamic evolution of service application, we take service composition as the core concern, while its evolution as crosscutting concerns.

Before formally defining the model, we first introduce some notations. C, WS, TC are the sets of finite components, available services, and test suites before the evolution respectively. $RL : C \times C \rightarrow \{>, +, \|, n\}$ is the relation function between components, where $>$, $+$, $\|$, n represent the sequence, choice, parallel and loop relationship. $TW : C \rightarrow WS^*$ is the available service of component. $TW(C_i) = WS_i = \{WS_{i,1}, WS_{i,1}, \ldots, WS_{i,m}\}$ is the available services of component C_i, $WS_{i,j}$

is the *j*th available service of component C_i; $RO = \{It_0, Bi_0\}$ describes the interface and component binding before the evolution; $RN = \{It_n, Bi_n\}$ is the evolution interface and binding; $TR : TC \rightarrow WS \wedge C$ is the action set of test case.

Definition 1 (Base Net): A 7-tuple Σ = (N, IO, D, A_T, A_F, λ, M_0) is called Base Net (*BN*) model if:

(1) $N = (P, T, F)$ is a Petri net, where P, T, F represent the place, transition, arc, and they are disjoint;

(2) $IO \subset P$ is a special place, which is called the interface of Σ and denoted by dotted circle;

(3) D is the non-empty finite individual set, f_D, f_S is the predicate set and symbol set;

(4) $A_T : T \rightarrow f_D$, for $t \in T$, the free variable in $A_T(t)$ must be in the directed arc with one end of the arc is t;

(5) $A_F : F \rightarrow f_S$, if $(p, t) \in F$ or $(t, p) \in F$, then $A_F(p, t)$ or $A_F(t, p)$ is the *n*-symbol set, the default value is empty;

(6) $\lambda : T \rightarrow N^*$ is the priority of transition, the default value is 0, the smaller of λ_i, the greater of transition's priority.

(7) $M_0 : P \rightarrow f_S$ is the initial marking of Σ.

For $\forall x \in (P \cup T)$, we denote the pre-set of x as $^\bullet x = \{y | y \in (P \cap T) \wedge (y, x) \in F\}$ and the post-set of x as $x^\bullet = \{y | y \in (P \cap T) \wedge (x, y) \in F\}$.

Definition 2 (Composition Net): A 6-tuple $\Omega = (\Sigma, \Gamma, TI, TA, PI, PA)$ is called Composition Net (*CN*), where:

(1) Σ is a *BN* model for the basic structure of Ω;

(2) $\Gamma = \{\Gamma_i | i \in N^*\}$ is a finite set of pages, each page is a *BN* or *CN* model, and it is disjoint between pages;

(3) $TI \subset T, PI \subset P$ are sets of substituted node and interface node;

(4) $TA : TI \rightarrow \Gamma$ is the page allocation function, which allocates page to the substituted node;

(5) PA is used to map the interface node of substituted node into the input and output interface of the page.

The individual set D is mainly used to describe the resource of service composition. We abstract the element as the individual d_i=(it, i, RW_i), where $it \in \{w, c, d\}$ represents the described object of individual, w, c, d represent the service, component and data packet, i represents the position of individual. For example, service $WS_{i,j}$ in the service set WS, when it is equal to w, RW_i is used to describe the interface of service; when it is equal to c, RW_i is used to describe the binding of component; when it is equal to d, RW_i presents the content of data packet. $k\#(d_i)$ represents the kth element in the individual d_i. In addition, we denote $N_{i\bullet}x$ as the element x in net N_i.

The mapping $CutN$:$\{N_{i\bullet}x_j, \ldots N_{f\bullet}x_k\}$ is called a pointcut of the model, where $CutN$ is the name of pointcut, $N_{i\bullet}x_j, \ldots N_{f\bullet}x_k$ are joinpoints of the pointcut.

Definition 3 (Aspect): A pair asp=($CutN$, AD) is called an aspect, where $CutN$, AD represent the pointcut, and advice respectively. An advice consists of rules for service integration, and the main part of a rule is an introduction net that models the implementation logic for that pointcut.

Let Ω be a CN model, M is a marking of Ω. For transition $t_i \in T$, if $\forall p \in^\bullet t_i$, there is $M(P) \neq \varnothing$, and $A_T(t_i)$ is true under marking M, then transition t is enable under marking M, denoted by $M[t >$. All the enabled transitions under marking M are denoted by set $ET(M)$.

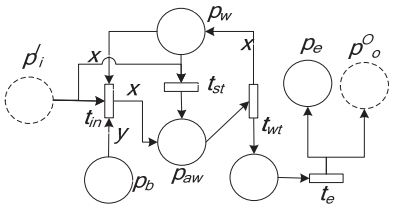


Fig. 1. BN model of component

The transition t is effective under marking M if and only if transition t_i is enable and there doesn't exist transition in $ET(M)$ whose priority is greater than t_i. All the effective enabling transitions under marking M are denoted by $FT(M)$. The model Ω will reach new state M' by effectively firing all transitions in $FT(M)$, denoted by $M[t_i > M'$, M' is a reachable state of M. The computation of M' is: $\forall p_j \in^\bullet t_i \cup t_i^\bullet$: $M^{'}(P_j) = M(P_j) - W(P_j, t_i) + W(t_i, P_j)$.

If there exists a firing sequence $t_1, t_2, \ldots, t_k$ and a marking sequence $M_1, M_2, \ldots, M_k$ such that $M[t_1 > M_1[t_2 > M_2 \ldots M_{k-1}[t_k > M_k$, then M_k is called a reachable state of M. All the possibly reachable states of M are denoted by $R(M)$.

B. The Modeling Method

a) Modeling composition net: The BN model of component C_i is shown in Fig.1, where the place p_i^I and p_o^O represent the input and output interface, p_{in}^I and p_{ou}^i represent the input and output interface in the top model. The process is: (1) if component C_i has got the input parameters p_i^I, then firing transition t_{in} to select an individual x from the library of available service p_w to complete its function; (2) if the component couldn't get the binding service, then calling transition t_{st} to randomly select an available service to complete its function; (3) if the service has completed the function, then calling the termination transition t_e to output the results φ to place p_e and p_o^O.

The composition net is constructed by the following steps.

(1) Constructing the *BN* model of all components in composition process by using the construction method of component.

(2) Introducing transition t_s and place p_s to describe the beginning operation and position, and doing the initialing operation for the system; transition t_e and place p_e are used to describe the termination operation and position, and merging the output of component based on their relationships.

(3) Modeling the basic relationships between components.

(4) Setting the initial marking $M_0(p_s)$ = φ

We obtain *CN* model of service composition based on the above steps, which is shown in Fig.2, the substituted node T_i is mapped into the page of component C_i.

b) Modeling aspect: From the principles of AOP, we can get that advice net is included in the introduction, therefore, we only give the model of introduction in the process of

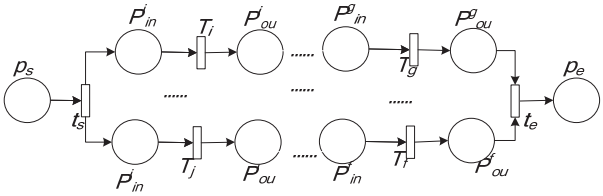


Fig. 2. Composition net Ω of service composition

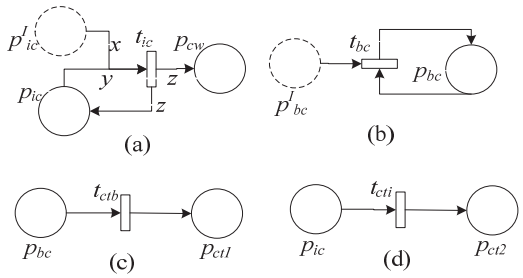


Fig. 3. Advices of crosscutting concern

modeling crosscutting concern.

- Interface aspect $con_{ic} = \{p_{ic}, AD_{ic}\}$

The pointcut of interface is defined as: p_{ic} : $\{CN_{1\bullet}p_w$, $CN_{2\bullet}p_w$, ..., $CN_{|C|\bullet}p_w\}$, the advice is shown in Fig.3(a). p_{ic} is used to describe the change process of service's interface. Place p_{cw} is used to save the evolution records of service. p^I_{ic} is used to store the current invoked service. When the interface of available service $WS_{i,j}$ has modified, then calling transition t_{ic} to update the interface information of place p_{ic}, and the result is stored in place p_{cw}. The weaving rules of con_{ic} are: for each connection point, the system will insert the point into the after set of advice and its associated subnet. The priority of weaving transitions is set to 3.

- Binding aspect $con_{bc} = \{p_{bc}, AD_{bc}\}$

The pointcut of interface is defined as: p_{bc} : $\{CN_{1\bullet}p_b$, $CN_{2\bullet}p_b$, ..., $CN_{|C|\bullet}p_b\}$, the advice is shown in Fig.3(b). Pointcut p_{bc} is used to describe the change process of component's banding. Place P^I_{ic} is used to save the current binding service. When the binding of component has changed, then calling transition t_{bc} to update the binding information of place p_{bc}. The weaving rules of con_{bc} are: for each connection point, the system will insert the corresponding point on CN_i into the after set of advice and its associated subnet, the priority of weaving transitions is set to 2.

- Evolution notify aspect con_{nc}= $\{\{p_{nc1}, p_{nc2}\}, AD_{nc}\}$

The pointcut is defined as: p_{nc1} : $\{P^1_{bc}, P^2_{bc}, \ldots, P^{|C|}_{bc}\}$, p_{nc2} : $\{P^1_{ic}, P^2_{ic}, \ldots, P^{|C|}_{ic}\}$, the advice is shown in Fig.3(c)-Fig.3(d). Pointcut p_{ct1}, p_{ct2} are used to notify the evolution of system to all components. Place p_{bc}, p_{ic} are used to save the current binding and the evolution data of interface. The weaving rules of con_{nc} are: The system will firstly weave the evolution advice concern, then weaving the evolution concern in the current evolved component. In order to handle the notify of evolution timely, the priority of weaving transitions is set to 1.

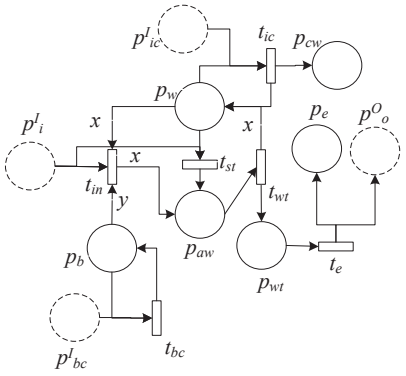


Fig. 4. Testing aspect of component

According to the above model, the steps of aspect weaving are:

(1) Adding all the advices in the set con to Ω, thus forming a new composition net;

(2) Adding the corresponding elements according to the weaving rules and merging the same element;

(3) Weaving aspects in decreasing priority order when more than one aspect is applied to the same pointcut.

The model got by weaving the above concerns into the composition net is called the test model Ω_t. If $M(p_e) = \varphi$, then M is called a normal termination marking. The testing aspect CA_i of component C_i is constructed according to the weaving rules of concern, which is shown in Fig.4.

III. Regression testing techniques

Regression testing of basic service is to ensure that its function and interface information can still meet the requirements after the evolution. When service provider does regression testing for basic services, the system selects the required testing path and test cases. In contrast, the service consumers can generally select and modify the test case according to the change of interface, and getting the results to complete the test based on the invoked services. When the original binding service is replaced by another service, it is necessary to re-run the test cases of all paths that called the replaced service, thus to verify whether the system can operate normally and the results are correct.

We first reduce the size of regression test by applying the following selection strategy.

(1) Initialing the regression set $CTC = \varnothing$, the component need to be tested is $DC = \varnothing$, the available service need to be tested is $DWS = \varnothing$;

(2) Computing DC: a. $\forall C_i \in C$, if the binding of C_i has changed, then $DC = DC \cup C_i$; b. $\forall C_i \in DC$, $\forall C_j \in C - DC$, $RL(C_i, C_j) =>$, then $DC = DC \cup C_j$;

(3)Computing DWS: $\forall WS_{i,j} \in WS$, if the interface of service $WS_{i,j}$ has changed, then $DWS = DWS \cup WS_{i,j}$;

(4) Constructing the test suite of regression test: $\forall tc_k \in TC$, if $TR(tc_k) \cap (DC \cup DWS) \neq \varnothing$, then $CTC = CTC \cup tc_k$;

(5) Filtering the test suite of regression test: $tc_k \in CTC$, if there exists $tc_f \in CTC$, which makes $TR(tc_k) \subseteq TR(tc_f)$, then $CTC = CTC - tc_k$.

According to the different types of evolution, the paths of regression testing p consist of two parts: the path set p_i affected by interface change and the path set p_b affected by binding change. The algorithm is shown in Algorithm 1. Let the path set of composition net Ω be *path*(Ω).

(1) Computing the set p_i: The system will compare the number of parameters in the interface, the definition of data type of each parameter in the interface, then do regression test in case any change occurs.

(2) Computing the set p_b: According to the relationship between the components, if the binding of component C_i has changed, then the associated components of C_i should be re-tested to ensure that the binding is effective.

Algorithm 1 Regression test path generating

Require: Composition net Ω' and its path set $path(\Omega)$;
Ensure: The path set needed to test by regression testing;

Analysis(Ω) //Computing the evolution service and component
{EC=EWS=ϕ;
For k=0,i≤| C |, k++ do
{ if $\exists d \in M_0(p_{bc}) \wedge 2\#(d) = k$ *then* $EC = EC \cup \{C_k\}$
For i=1,j<| WS_k) |, i++ do //Verifying the evolution of available service
{ If $\exists d \in M_0(p_{ic}) \wedge 2\#(d) = (k,i)$, then $EWC = EWC \cup \{WS_{k,i}\}$;}
Foreach $C_k \in C - EC$ do
Foreach $C_i \in EC$ do
If $RL(C_k, C_i) =>$, then $EC = EC \cup \{C_k\}$; }}
Compute_pa(EC, EWS, $path(\Omega)$)//Computing the paths of regression testing;
{ $Npath = \phi$;
Foreach $\delta \in path(\Omega)$ do
{If $\exists C_i \in EC \wedge CA_{i\bullet}t_{in} \in \delta$ then $Npath = Npath \cup \delta$;}
{ If $\exists WS_{i,j} \in EWS \wedge 2\#M_\delta(CA_{i\bullet}p_{ac} = (k,i)$, then $Npath = Npath \cup \delta$;}}

We will analyze the effectiveness of algorithm based on the state space of model, that is, (1) If the binding of component changes, then the related paths will be tested; (2) If the bindings of the afterward component of component C_i changes, then the related paths of component C_i will be tested; (3) If the interfaces of service changes and the service has been called, then the related paths of service will be tested.

Theorem: In the test model Ω_t, let the path set of composition net Ω be $path(\Omega)$, $Npath$ is the corresponding path set of regression test, $\forall \delta \in path(\Omega_t)$, there are:

(1) $\forall C_i \in C$, if $C_i \in Bi_n$, then $CA_{i\bullet}t_e \in \delta \rightarrow \delta \in Npath$;

(2) $\forall C_i \in C$, if $C_j \in c, RL(C_j, C_i) =>$, then $CA_{i\bullet}t_e \in \delta \rightarrow \delta \in Npath$;

(3) $\forall WS_{i,j} \in WS$, if $WS_{i,j} \in It_n \wedge 2\#M_\delta(CA_{i\bullet}p_{ac}) = (k,i) \rightarrow \delta \in Npath$;

TABLE I
EVOLUTION INFO OF EXPORT SERVICE

WS	It_o		WS	It_o		WS	It_o		WS	It_o	
	Pn	Pt		Pn	Pt		Pn	Pt		Pn	Pt
$WS_{1,1}$	1	{I}	$WS_{2,1}$	1	{I}	$WS_{3,1}$	2	{B,S}	$WS_{4,1}$	1	{I}
$WS_{1,2}$	2	{I,S}	$WS_{2,2}$	1	{I,B}	$WS_{3,2}$	2	{B,S}	$WS_{4,2}$	1	{I}
$WS_{1,3}$	2	{I,S}	$WS_{2,3}$	2	{I,B}	$WS_{3,3}$	2	{B,S}	$WS_{4,3}$	1	{I}
$WS_{1,4}$	1	{I}	$WS_{2,4}$	2	{I}	$WS_{3,4}$	2	{B,S}	$WS_{4,4}$	1	{I}
$WS_{5,1}$	3	{I,S,B}	$WS_{6,1}$	1	{S}	$WS_{7,1}$	2	{I,S}	$WS_{8,1}$	1	{D}
$WS_{5,2}$	3	{I,S,B}	$WS_{6,2}$	1	{S}	$WS_{7,2}$	2	{I,S}	$WS_{8,2}$	2	{D,B}
$WS_{5,3}$	2	{I,S}	$WS_{6,3}$	2	{S,I}	$WS_{7,3}$	2	{I,S}	$WS_{8,3}$	2	{D,B}
$WS_{5,4}$	1	{I}	$WS_{6,4}$	1	{S}	$WS_{7,4}$	2	{I,S}	$WS_{8,4}$	2	{D,B}

Proof: let the marking included in δ be the set $R(\delta)$, $\forall C_i \in C$, because $C_i \in Bi_n$, according to the modeling process of aspect, $\exists d \in M_0(p_{bc})$ which makes $2\#d = (k,i)$; and because $p^\bullet_{bc} = t_{ctb}, P^i_{in} \in t^\bullet_{ctb}$, therefore, it exists $M \in R(\delta)$, which makes $d \in M(CA_{i\bullet}p^I_{in})$; Because $CA_{i\bullet}p^I_{in}{}^\bullet$=$\{CA_{i\bullet}t_{in}, CA_{i\bullet}t_{st}\}$, $(CA_{i\bullet}t^\bullet_{in})^\bullet$=$(CA_{i\bullet}t^\bullet_{in})^\bullet$= $CA_{i\bullet}t_e$; and because $CA_{i\bullet}t_e \in \delta$, according to Algorithm 1, we can get $\delta \in Npath$.

Similarly, proposition (2) and (3) can be proved. Theorem 1 says that the regression path set can describe the evolution of the interface and the binding of service composition according to Algorithm 1.

IV. A CASE STUDY

This section presents a simple case study of Export Service System. The business process includes looking up the relevant information and selecting the destination (C_1) according to the requirements, and packaging services (C_2) responses for processing and packaging export products. The system will implement the operation of export transport ordering (C_3) and insurance processing (C_4) in parallel, while Export transport ordering include water transport ordering(C_5) and airport ordering (C_6). All the information is confirmed by exporter and will be sent to the regulatory authorities (C_7) for checking. Finally, financial services (C_8) is used to check the related instruments of product. The composition process can be represented by expression $C_1 > C_2 > (C_3 \parallel C_4 \parallel (C_5 + C_6)) > C_7 > C_8$, the specific available service and test cases are shown in Table I. Let the binding service set of each component be: $WS_{1,1}$, $WS_{2,3}$, $WS_{3,1}$, $WS_{4,3}$, $WS_{5,3}$, $WS_{6,4}$, $WS_{7,2}$, $WS_{8,3}$. The binding evolution of Export Service be $\{WS_{2,2}, WS_{5,2}\}$, that is, the binding of component C_2, C_5 have changed. The changing interface of available service be: $WS_{1,1} : \{2, \{\{Int, String\}\}\}$, $WS_{3,3} : \{1, \{\{String\}\}\}$, $WS_{4,2} : \{2, \{\{Int, String\}\}\}$, $WS_{6,4} : \{2, \{\{Int, String\}\}\}$, $WS_{8,1} : \{2, \{\{Double, Boolean\}\}\}$.

We can weave the evolution concerns into the composition net Ω, which is shown in Fig.6. the model mainly describes the composition process of Export Service. According to the construction method of state space, the reachable states of composition net and aspect are 216 and 312 respectively. $CA_{i\bullet}t_{bc} \in \delta(M, M_e)$, that is, the binding evolution of component C_2 has been tested, in the similar way, we can get the evolution of all services have been tested before the end of service composition. We can compute the paths in Export

Service(component and service), which are $< C_1, C_2, C_3, C_4, C_5, C_7, C_8 >$, $< C_1, C_2, C_3, C_4, C_6, C_7, C_8 >$, followed by regression testing paths and services $WS_{1,1}$, $WS_{6,4}$.

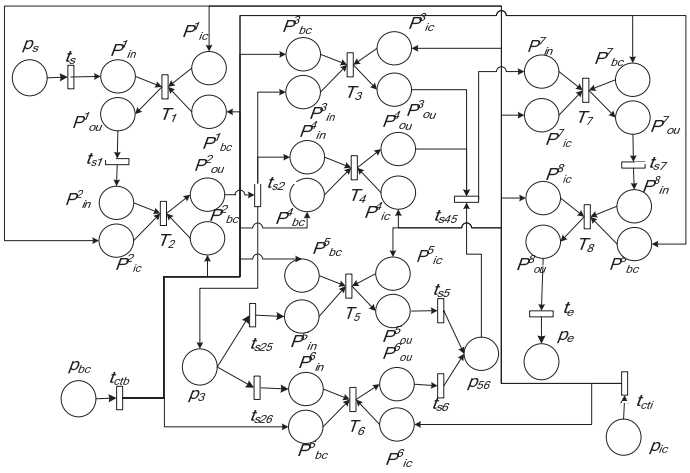


Fig. 5. Model of Export Service

In order to effectively estimate the proposed approach, we conduct an experiment. The system randomly generates 8000 services as service resource and 260 test cases. And each service contains the historical and the actual testing info. A test case can be used to test available service for several times, the testing results of each time may be different. The purpose of Experiment is to analyze the necessity of using *AOP* for regression test. The steps are:

(1) Taking 30 test cases and 1120 available services as the resource of 5 Export Service, and supposing component in each Export Service contain 10, 20, 30, 40 and 50 available services;

(2) Constructing the composition net, testing model of Export Service, and then computing the size of its state space.

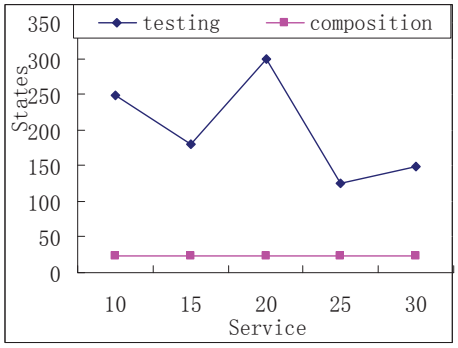


Fig. 6. The experiment result

The results of Experiment is shown in Fig.6. The result of Step(1) says that the state space of base net will not change with the number of service resource increasing, for example, the reachable states of base net of 5 Export Service is 24. The reason is: base net is mainly used to describe the implementation process of service composition, which is unrelated with the test case and service resource. Therefore, the state space of base net is unrelated with the test suite and the set of available service. The result of Step(2) means: (1) The state space of test model will not strictly decrease with the available services increasing, when the evolution is more complex, the state space of test model will be relatively large.

V. Conclusion

This paper proposes a regression test technique for analyzing the functionalities of service composition. Services and components are modeled by Petri nets, while service evolution is specified as crosscutting concerns, and in turn woven into the testing model of service composition by weaving rules. The benefits of this work include formal reasoning correctness of service composition, flexibility of composition process, and efficiency of regression testing. In the future, we plan to investigate more concerns such as performance and security and develop the related tools.

Acknowledgment

This work was partially supported by the NSF of China under grants No. 60773094 and 60903020, Shanghai Shuguang Program under Grant No. 07SG32, the National Key Technology R&D Program of China under Grant No.2009BAH46B03.

References

[1] Q. Yu, X. M.Liu, B. Athman, M. Brahim. Deploying and managing Web services: issues, solutions, and directions. The International Journal on Very Large Data Bases. 2008, 17(3): 537-572.

[2] M. Lallali, F. Zaidi, A. Cavalli, et al. Automatic timed test case generation for Web services composition. Proceedings of the Sixth European Conference on Web Services(ECOWS'08). IEEE Computer Society, 2008:53-62.

[3] Y. Zheng, J. Zhou, P. Krause. An automatic test case generation framework for Web services. Journal of Software, 2007, 2(3):64-77.

[4] S. Noikajana, T. Suwannasart. An improved test case generation method for Web service testing from WSDL-S and OCL with pair-wise testing technique. Proceedings of the 2009 33rd Annual IEEE International Computer Software and Applications Conference. IEEE Computer Society, 2009:115-123.

[5] S. G. Elbaum, A. G. Malishevsky, G. Rothermel. Test case prioritization: a family of empirical studies. IEEE Transactions on Software Engineering, 2002, 28 (2): 159-182.

[6] D. Jeffrey, N. Gupta. Test suite reduction with selective redundancy. In Proceedings of the 21st IEEE International Conference on Software Maintenance (ICSM'05), IEEE Computer Society, 2005:549C558.

[7] C. R.Panigrahi, R. Mall. Model-based regression test case prioritization. ACM SIGSOFT Software Engineering Notes, 2010, 35(6):1-7.

[8] B. Korel, G. Koutsogiannakis, L. H. Tahat. Model-based test prioritization heuristic methods and their evaluation. Proceedings of the 3rd international workshop on Advances in model-based testing. ACM New York, NY, 2007:34-43.

[9] B. Athira, P. Samuel. Web services regression test case prioritization. Processing of the 2010 International Conference on Computer Information Systems and Industrial Management Applications (CISIM). IEEE Computer Society, 2010:438-443.

[10] G. Kiczales, J. Lamping, A. Mendhekar, et al. Aspect-oriented programming. Proceedings of the European Conference on Object-Oriented Programming, LNCS 1241. Springer-Verlag, 1997:220-242.

[11] M. TADAO. Petri nets: properties, analysis and application. Proceedings of the IEEE. 1989, 77(4): 540-581.

[12] G. Fan, H. Yu, L. Chen, et al. An aspect oriented approach to constructing secure service composition. The 2010 Asia Pacific Software Engineering Conference (APSEC2010), IEEE Computer Society, 2010:176-185.

A view towards Organizational Learning: An empirical study on Scrum implementation

Viviane Santos and Alfredo Goldman
Institute of Mathematics and Statistics
University of São Paulo
São Paulo, Brazil
{vsantos, gold}@ime.usp.br

Ana Carolina M. Shinoda and André L. Fischer
Faculty of Economics and Management
University of São Paulo
São Paulo, Brazil
{carol.shinoda, afischer}@usp.br

***Abstract*—Scrum is one of the agile methods gaining more relevance among academics and practitioners. It is mainly applied in the context of software development, which is a knowledge-intensive activity and depends on learning to evolve. Therefore, it becomes crucial to understand the question: *Does Scrum implementation trigger a process of Organizational Learning? If 'Yes', how does the OL occur?* This relation has never been established in previous studies. The answer to these questions was based on a qualitative research, involving key members from UOL - the most important Brazilian company regarding content and services on the Internet, key members from an academic project and an expert in agile methods implementation. Among the main findings, we highlight that the process of Organizational Learning could be verified through the individual members' learning and through the changes within the organization in management, people, process and technology. Beyond the relation established between Organizational Learning and Scrum implementation, this study contributes to academic and practical fields by the identification of changes occurred in type of knowledge valued, physical structure, promotion criteria, and individual dependence decrease when implementing Scrum. It is perceived that knowledge management, as a way of perpetuating the learning in the organization is still a challenge for agile software organizations.**

Keywords- Organizational Learning; Agile Software Development; Scrum implementation.

I. INTRODUCTION

Agile methods have been in evidence lately. Among these, one of the most popular agile methods is Scrum [1], which is focused on project management and holds ceremonies and roles that follow the agile values and principles formalized by the Agile Manifesto [2]. Its reputation is due to the increasing chances of project success, when applied properly [3].

Software development is a knowledge-intensive activity [4]. Given that, learning is crucial for organizations in this area. As more software development organizations are adopting Scrum, it becomes relevant to understand whether it contributes or not to the process of Organizational Learning (OL).

Every organization learns, as it is a basic requirement for its sustainable existence; and its learning is acquired through their individual members. Yet, OL is a more complex phenomenon than the simple sum of individual's learning [5, 6, 7, 8].

The definition of OL is not consensual and it involves multiple aspects. In this study, we take Nicolini and Meznar's definition that relates learning to change: OL corresponds to investigating why and how the organization changes [9]. Fiol and Lyles [7] associate change with behavior development and learning with cognitive development. Based on that, they add that not all changes imply on learning and that might be learning without any accompanying change in the behavior. Therefore, in this study we will consider cognitive developments and changes in behavior accompanied by cognitive development as learning.

Through a wide search on the topic, including several electronic databases pointed hereafter, the relation between Scrum implementation and the OL process has never been fully established in literature. However, it is quite relevant to study this relation since learning is decisive for software development organizations. For this reason, this research aims to perform an empirical study to understand whether the Scrum implementation triggers OL and, if positive, how it occurs. This study represents an innovative focus on Agile Software Development (ASD) field.

Cognitive and behavioral changes were verified through the analysis of whether the perceived individual learning of the organizational members increased with the Scrum implementation (as it is the first step for OL to happen). And also through the modifications in four aspects: management, people, process and technology. These aspects were pointed out by Nerur, Mahapatra and Mangalaraj [10] as the key issues in migrating to agile. Since they clearly relate them to organizational change, we used these key issues to guide our study. We complement their findings, as contributions for academy and practice, providing other identified changes, such as in types of knowledge valued, physical structure, promotion criteria and in individual dependence decrease.

This research consists of a qualitative analysis involving three lectures on the theme of Scrum implementation and six interviews with actively involved people on its implementation.

The structure of the paper is as follows. In Section II, we review the literature related to OL and ASD implementation. In Section III, the research methodology is described. In Section IV, we present the data analysis. In Section V, we discuss the findings. Finally, in Section VI, the conclusions are presented.

II. Literature Review

After searching within several electronic databases, such as ACM Digital Library, Compendex, EBSCO, Elsevier ScienceDirect, Google Scholar, IEEExplore, ISI Web of Science, JSTOR, SpringerLink, and Wiley Online Library, no specific study was identified exploring the relation between the Scrum implementation and the OL process.

However, there are few relevant previous studies relating ASD to OL concepts, such as Argyris double-loop learning [11, 12], maintenance of corporate experience repositories as a way to guarantee continuous learning [13], and March's exploitation-exploration balance [14] [15]. We also found studies outlining the importance of OL in Scrum/Agile adoption, but they stress the need to support the process itself to improve long-term learning [16, 17, 18, 19, 20, 21, 22]. Likewise, one study relates Scrum adoption as organizational becoming [23]. Also, two studies relating software process improvement to organizational change were identified [24, 25]. Besides, studies point out the need for future research on OL in ASD [21, 26, 27].

Despite the wide acceptance of OL and its importance to strategic performance, there is no theory or model largely accepted [7], and the concept is still evolving [8].

We use the definition of Nicolini and Meznar [9] that relates learning to change. They state that OL rely on exploring why and how the organization changes. Other authors support this view. Antonello [8], in a literature review about OL, stated that the literature promotes a strong relation between learning and change, and some authors contend they are synonyms. As stated by Fiol and Lyles [7], OL is the process of improving actions through better knowledge and understanding. The authors differentiate organizational learning (cognitive development) and change (behavior development). Therefore, changes in the organization can be considered learning as long as they are accompanied by cognitive development.

There is a consensus that OL is first acquired by the organization individual members, but OL is not the simple sum of each member's learning [5, 6, 7, 8]. It means that the first step onto OL comprehension is the understanding of its individual's learning. As the synergy of the shared understandings and consensus increase within groups through knowledge propagation and socialization, the organization starts to adjust its behavior as a response to performance problems [5].

Past studies in ASD acknowledge that there is learning at individual and group levels [1, 21, 28, 29], however none has actually described the OL process in ASD.

For a deep comprehension of OL, there are other aspects to be analyzed such as changes in management, people, processes and technology. Specifically in the software development area, it is important to take into account Nerur, Mahapatra and Mangalaraj [10] study where they detail key issues (Table I) that an organization may have to face when migrating to agile.

TABLE I. Key issues in migrating to agile

Management and Organizational: Organizational Culture, Management Style, Organizational Form, Management of Software Development Knowledge and Reward System.
People: Working effectively on a team, High level of competence and Customer relationships (commitment, knowledge, proximity, trust, respect)
Process: Change from process-centric to a feature-driven/people centric approach, Short iterative, test-driven development that emphasizes adaptability, Managing large/scalable projects and Selecting an appropriate agile method
Technology (tools and technique): Appropriateness of existing technology and tools and New skill sets (refactoring, configuration management, J-units)

These issues can be seen as causes and consequences of migrating to agile because at the same time that they represent assumptions or enablers for the implementation to happen, they are also affected when the implementation takes place.

Chang and Thong [30] have provided a critical review of the literature on the acceptance of agile methods. They found only eight case studies discussing the acceptance of agile methods. Only two of them were specifically about Scrum implementation [31, 32]. Table II depicts those main factors.

TABLE II. agile methodologies acceptance adapted from [30]

Factors for agile methodologies acceptance	Method studied
Negotiation skills	Scrum
External support, Career consequences, Micromanagement, Resistance due to past experience	Scrum, XP
Compatibility with agile methods	Scrum, Agile
Individual ability / competence/ motivation / Experience in software development	Scrum, Agile, XP
Teamwork, Management support, Communication, Project type, Team size, Organizational culture and form, Management style, Management of software development knowledge, Reward system, Customer relationship, Existing technology and tools, Training	Agile

After the Scrum implementation, several challenges related to Knowledge Management (KM) need to be faced to perpetuate and leverage the learning in the organizations. Levy and Hazzan [4] highlight how the agile approach initiates a culture change that is in line with the one needed for a KM initiative. They describe KM enablers that are embedded in the agile approaches. For Chang and Thong [30], KM is critical for ASD, but it has to be conducted in a different way, since agile methods are more people-oriented and deal mostly with tacit knowledge.

An important aspect related to OL observed in previous studies is the lack of KM practices integrated to the Scrum management process [4], even though the practices encourage the knowledge exchange among individuals in a team. In summary, the process of OL that started with Scrum implementation must be supported by KM practices in order to perpetuate the learning in the organization.

III. Research Methodology

This empirical study is a qualitative research [33], since it aims to discover and understand the complexity and interaction of factors related to the phenomenon in study. As Seaman [34] state, the main advantage of using qualitative methods is that they force researchers to deepen the complexity of the problem, rather than generalize it. Thus, the results are richer and more informative. On the other hand, they are more difficult and stressful than a quantitative analysis.

A. Research Question

This research can be classified as exploratory [35], since the goal is to examine a topic or research problem scarcely studied, for which we have many questions not addressed before [36]. We aim to answer the following research question:

- *Does the Scrum implementation trigger a process of OL? If 'Yes', how does the OL occur?*

B. Data Collection

Semi-structured questionnaires with open questions were held to allow improvisation and to invite a broad range of answers and issues from the phenomenon in study.

The questionnaires were constructed from the literature, in order to answer the research question, considering the necessary assumptions for Scrum implementation to happen and their perceived changes after the implementation (in areas related to Management, People, Process and Technology). There were also questions regarding the challenges related to KM (what is being applied to perpetuate the learning initiated with Scrum implementation). The questions were adapted to each interview. All interviews were recorded and transcribed.

C. Data Analysis Method

The analysis of the collected data was made inspired by open coding in grounded theory [37]. Interesting expressions were categorized to describe the phenomenon in study.

D. Data Sources

The empirical study considers the qualitative analysis of three lectures on the theme and six interviews involving three different data sources to provide data triangulation:

- Two key members from Universo Online (UOL), the most important Brazilian company regarding content and services on the Internet (http://sobreuol.noticias.uol.com.br/index_en.jhtm). The expansion of services offered to the web market has demanded changes in company's process, which has led to the Scrum implementation in some of the software development areas. We attended lectures of both professionals, which are actively involved on Scrum implementation, and we conducted two group interviews with them to investigate the OL process at the organizational level.
- Three members from the Workbench Project (WP), an academic project that applies Scrum method (http://www.groupwareworkbench.org.br/gt). This project is an initiative of the Institute of Mathematics and Statistics from University of São Paulo to develop components aimed at social interaction and collective intelligence for building collaborative applications on Web 2.0. We interviewed individually the Scrum Master, the technical leader and one of the developers to analyze the learning at individual and group level.
- Specialist in implementing agile methods, including Scrum, Heitor Roriz, from Massimus Consulting (http://massimus.com/) conducted a lecture and was also interviewed by the authors to deepen the understanding of the relationship between Scrum implementation and OL. Given his experience in several software development organizations, his point of view was important to triangulate data and to perceive if the results verified at WP and UOL could be found in other organizations or if they were restricted to their context.

IV. Data Analysis

Data analysis was performed by coding interesting expressions detected by the authors, considering their expertise on the research topic. This section presents them in detail.

The qualitative analysis highlighted the importance of what we have called "assumptions", factors that contribute to the acceptance of the Scrum implementation. They are endorsed by the literature. Also, it raised the changes identified in the study, representing evidences of learning that have occurred.

A. Assumptions for the Implementation

It is important to understand the assumptions adopted for the Scrum implementation as they allow the organization to benefit from the subsequent organizational learning generated. The assumptions identified are detailed as follows.

1) Understanding of the Scrum method: An understanding of the Scrum method is important once many organizations have lack of knowledge of the practices required to use Scrum effectively. As well, many organizations confuse some of their practices with other methods. The specialist interviewed supported this point by stating that "*It happens that some companies say that they are using Scrum, when they are not actually doing so. An example is when they use Kanban – which belongs to Lean – thinking it is Scrum*". This way, it is important to achieve a broad understanding and a full implementation of the Scrum method to perceive its benefits.

2) Make sense in the context: It is crucial that the implementation makes sense for the organization. In the case studied, some factors contributed to this aspect: team dissatisfaction and overload with the previous method, the need to have a fast method to keep up with the organization's startups, competitors already using Scrum as well as some of the team members getting in contact with Scrum by external conferences, academic influence and books. One of the interviewees from UOL argued that *"if everybody were*

comfortable with the process… 'oh, it is ok, it works'… or 'leave it like this', it wouldn't have changed";

3) Top management support: To change the work method, it is necessary to convince the top management that it is worth. An interviewee reported the need to also line up the Scrum implementation with the business expectations and adapt the technical to management vision to support top management convincement: *"the development area had to 'speak the language' of the top management to be able to convince them. It was of no use to talk about the method. We needed to focus on the benefits it might bring: reductions in time, cost…"*;

4) Shared vision establishment: In a change process, it is important to establish a shared vision of the future. It implies expectations alignment and also changes understanding. Metaphors may help in this task as it was used at UOL, as reported in the lectures: *"We were changing the airplane's wing during the flight… so it would not be an overnight change as it is difficult to do such thing"*. Another statement related to the expectation alignment was: *"we made clear for the development team as well as for the top management that Scrum was not the solution for all the problems"*;

5) Team's competency and commitment: It was also pointed out to have a competent and committed team. As an interviewee from UOL stated: *"You need competent people, with a strong knowledge base, committed to the job… this is the base… without it… it is an assumption"*;

6) Non hierarchical culture: Not all organizations can implement Scrum. It was reinforced by the specialist interviewed as well as at WP and UOL. A WP member summed up this need by saying that *"the culture cannot be the 'I impose and you obey'"*.

B. Changes and Learning

According to Nicolini and Meznar [1], in order to understand the phenomenon of OL, it is important to analyze the changes that occurred in the organization. Fiol and Lyles [7] add that change can be considered learning since it is accompanied by cognitive development. Given the basis provided by these authors, cognitive developments of the members and changes in behavior accompanied by cognitive development were considered learning.

Through the data analysis and consensus among the researchers, subcategories were created below within the categories proposed by Nerur, Mahapatra and Mangalaraj [10].

1) Management and Organizational

a) Types of knowledge: The organization undergoes significant changes in the types of knowledge. Individual tacit knowledge increases due to a higher communication and teamwork. Many interviewees state that the individuals end up increasing their knowledge because of the higher interaction with other team members. A Workbench project member endorsed it by saying: *"I saw that other people's level was above mine and it forced me to learn (…) not because somebody told me to, but for self motivation.* The organizational tacit knowledge also increases by the reinforcement of shared culture and process. However, we identified a more unexpected change in the explicit knowledge. Despite the decrease in the documentation related in all the interviews, it was possible to conclude that the explicit knowledge increases with Scrum implementation. It happens because the documentation that used to be generated rapidly became outdated and, therefore, people did not access them: *"in the past, people used to write a lot, but anybody read that"*. Knowledge is related to action [38], so a document that is not accessed and does not lead to any action, cannot be considered knowledge, but information instead. As the documents generated when working with Scrum contain only the essential knowledge, it is therefore accessed and turned into action: *"today, everything that is critical is documented"* as well as *"what is updated is what you have the obligation to update in order to make it work"*.

b) Hierarchical structure: This indirect change happened because as the development teams must work together and sit near each other, these individuals tend to become part of the same functional area. This is confirmed by an interviewee from UOL: *"the hierarchical structure changed […] some departments disappeared, some remained but changed their way of working"*. The change in the way of working refers to the team members that respond to a different area becoming really part of the team, even working in the same space/room.

c) Physical structure: As the team members sit close to each other when working with Scrum, the physical structure in the development area has changed. It was seen as a very important change, as now the cross-functional team work for the same product: *"it used to be resource competition among the areas"*.

d) Promotion criteria: The change in the type of knowledge already presented, alongside with the transparency increase by Scrum implementation may improve the promotion criteria: *"the promotion criteria became more legitimate, more the essence of the person's value and not the circumstance"*.

2) People

a) Values and behavior: As Scrum is usually implemented along with agile principles, there are joint reflexes in changes of team values and behaviors. This is one of the main changes pointed out by all the interviewees. One of the interviewees from UOL commented about the changes he observes in the team: *"just the fact that everybody is in a smaller team […], the daily exposition by the daily Scrum, the task visibility on the board, everybody talking about what they are doing, the deadline commitment… and even when a new member comes to the team, somebody just go there to help him… it is very good for the team"*. Some of the several aspects related to this topic and indicated during the research were: increase of communication and transparency in the relationships, less imposition by managers, bigger sense of commitment, responsibility, organization and freedom of expression (questionings, opinions etc), better understanding of the work processes, commitment to the final product and objectivity during the meetings. The specialist in Scrum

implementation supported theses changes as he observes in several organizations: *"the characteristic of every team when you implement Scrum: feeling of responsibility…. Everybody say that. The feeling of organization […] you know what you are doing. Then you compromise"*.

b) Individual dependence: An interesting impact of Scrum can be the reduction of the individual dependence for the organization. It happens because the knowledge is largely widespread through several members, reducing the need for one specific individual: *"knowledge is spread […]there were component owners previously[…] today it doesn't exist anymore. Now the team takes everything. When we started with Scrum, we had to make an effort to stop that. So I would say that the dependence has decreased"*. However, the human dependence as a whole remains strong, only the individual one decreases: *"[the dependence is] less from the individual. People dependence still exists because software is a human activity"*. It was pointed out by the interviewees from UOL that "dependence" must not be misunderstood: it does not mean that the organization has started to diminish the importance of each person, but quite the opposite. It was only the negative part of the dependence that has decreased (for example: a person who is not delivering a good job but cannot be fired because he/she is the only one who has a specific knowledge in the organization).

c) Roles: Scrum proposes different roles in the organization: Scrum Master and Product Owner as well as a different role for the team, demanding an active participation in the decision taking. The team becomes self-organized and the leadership operates as a coach more than a manager. One of the interviewees from UOL commented about his team active participation in the process: *"there isn't that attitude anymore of 'I have finished so I move the task ahead'"*.

d) Particular language development: Another change was the development of a particular language by the team. It is partly because of the shared work and values. It was commented that the team starts to say things such as *"that is a sprint thing"*, that has no meaning for a person of the organization outside the development team.

3) Process

a) Delivery model: The delivery model proposed by Scrum – iterative and incremental – differs from the traditional long term ones. It changes the way of thinking the job by the team members. An interviewee from UOL stated that *"We didn't have a culture of delivering in pieces […] and the thought of maximizing value at each delivery"*.

b) Tests: In order to cope with the new way of working, developers build the software and test it at the same time. It may represent an overlap with the tester role. However this is faced as a positive aspect: *"there is an overlap with the tester role, because the developer ends up implementing tests alongside with the development, but this is seen as a positive thing because it improves the test estimates"*.

4) Technology

a) Tools: While Scrum does not strictly recommend the use of tools, developers use them for purposes of facilitating project management and improving their source-code. A WP member said: *"Virtual tools help us seeing the progress of the project […] also, they force us to improve our work […] you realize that people are afraid of generating bad code"*.

V. Discussion of the Findings

Our research question was: *Does Scrum implementation trigger a process of OL? If 'Yes', how does the OL occur?* We verified that the answer to the first question is positive as an increase in individual's learning was identified – the starting point for OL – and several changes involving intellectual growth and improved actions could be seen in the organization that took place with the Scrum implementation. As our definition of OL involves the understanding of how organizations change, these changes are the basis to verify if a process of learning has occurred.

The study innovates by focusing on the relation between Scrum implementation and OL, never explored in previous studies. Besides, it contributes to the understanding of the issues in migrating to Scrum. It could be added in the study of Nerur, Mahapatra and Mangalaraj [10] the possible change with the physical structure of the organization, not only in the organizational form in the sense of hierarchy. Moreover, the promotion criteria is another point that may undergo change as Scrum increases transparency and disseminates knowledge within the team, making a contribution to a more fair criteria for promotions.

A very interesting finding is related to the change in types of knowledge in the organization. Nerur, Mahapatra and Mangalaraj [10] pointed out the shift in the balance of power from management to development teams that may occur as the majority of knowledge in agile development is tacit and resides in the heads of team members. However, we have concluded that explicit knowledge also increases with Scrum implementation (but not as much as tacit knowledge). This is based on the fact that the documentation developed in traditional methods could not be considered knowledge as it was hardly used because it rapidly became outdated.

Another interesting finding in the study is related to the organization's decrease in individual dependence. As the knowledge becomes more diffuse among team members, the organization becomes less dependent on individuals. Although it was noticed that people become more valued, the harmful dependence of a single individual is reduced.

Then, when Scrum implementation is overcome, a further challenge raises: how to manage the organizational knowledge in this context? In this study, some initiatives being established were identified, but the interviewees confirmed that this is a very important aspect to be dealt with in the near future.

VI. Conclusions and Further Work

The findings of the present study may be useful in several ways. First, it shows that Scrum implementation triggers a process of OL, a crucial process for software development

organizations. Besides, it clarifies some of the changes that an organization may undergo when implementing Scrum.

This qualitative study presents some limitations that reflect in threats to validity of the results. It is not yet possible to generalize the findings, since the sample is not extensive, but adequate to the research goals. Only one organization was studied and yet few people were interviewed. Moreover, although the people interviewed are considered key, we cannot guarantee that their views are consistent with other team members and the organization respectively.

Future studies may complement the identified factors in different organizations to increase the power of generalizing the results. Moreover, there is an opportunity for further research on how to manage knowledge in organizations that have already implemented Scrum [29].

ACKNOWLEDGMENT

We are grateful to UOL, Heitor Roriz and WP team for all the collaboration with this study. We thank Ana Paula O. Santos for indicating and introducing us to the WP team. This work was financially supported by CAPES, CNPq-Brazil and Fapesp.

REFERENCES

[1] K. Schwaber, Agile project management with Scrum. ISBN 0-7356-1993-X. Microsoft Press, 2005.

[2] Manifesto for Agile Software Development, http://www.agilemanifesto.org/ (2001)

[3] T. Dybå, T. Dingsøyr, and N. B. Moe, Agile Software Development - Current Research and Future Directions. Springer, 1st edition, 2010.

[4] M. Levy, O. Hazzan, "Knowledge management in practice: the case of agile software development," Proc. Cooperative and Human Aspects on Software Engineering (CHASE). ICSE Workshop. Vancouver. 2009.

[5] D. H. Kim, "The Link between Individual and Organizational Learning", in the Sloan Management Review, pp. 37-50, Fall, 1993.

[6] C. Argyris and D. A. Schon, "Organizational Learning: A Theory of Action Perspective". Addison-Wesley, MA, USA, 1978.

[7] C. M. Fiol and M. A. Lyles, "Organizational Learning," in the Academy of Management Review, vol. 10, number 4, pp. 803–814. October, 1985.

[8] C. S. Antonello, "A metamorfose da aprendizagem organizacional: uma revisão crítica", in Os Novos Horizontes da Gestão: Aprendizagem Organizacional e Competências, pp. 12-33, 2004.

[9] D. Nicolini, M. B. Meznar, "The social construction of organizational learning: conceptual and practical issues in the field", in Human Relations (HR), v. 48, pp. 727-46, 1995.

[10] S. Nerur, R. Mahapatra, and G. Mangalaraj, "Challenges of migrating to agile methodologies," Communications ACM, vol. 48, no. 5, May 2005, pp. 72-78. DOI=10.1145/1060710.1060712.

[11] O. Kettunen. Agile Product Development and Strategic Agility in Technology Firms. Master thesis, Helsinki University of Technology 2010.

[12] A. Boden, B. Nett and V. Wulf. Operational and strategic learning in global software development. IEE, (27) 58-65, Nov/Dec, 2010.

[13] M. Nick, K. Althoff and C. Tautz. Systematic Maintenance of corporate experience repositories. Computional Intelligence, v. 17, n.2, 2001.

[14] K. Lyytinen, G. M. Rose, Information system development agility as organizational learning. In European Journal of Information Systems, vol. 15, pp. 183–199. 2006.

[15] W. Keplinger, Agility in information systems development: characterisation, motivation and conceptualisation. In iSChannel - The Information Systems Student Journal, vol. 2, pp. 25-28. September, 2007.

[16] J. Srinivasan, K. Lundqvist, Using Agile Methods in Software Product Development: A Case Study. In Proceedings of the Sixth International Conference on Information Technology: New Generations. pp. 1415-1420. 2009.

[17] T. E. Fægri, Improving General Knowledge in Agile Software Organizations - Experiences with job rotation in customer support. In Proceedings of the Agile Conference, pp. 49-56. Chicago, IL. August, 2009.

[18] M. R. J. Qureshi, M. Kashif, Seamless Long Term Learning in Agile Teams for Sustainable Leadership. In Proceedings of the 5th IEEE International Conference on Emerging Technologies, pp. 389-394. 2009.

[19] O. Salo, P. Abrahamsson, An Iterative Improvement Process for Agile Software Development Research Section. In Softw. Process Improve. Pract., vol. 12, pp. 81–100. Wiley InterScience. 2007.

[20] M. Pikkarainen, O. Salo, J. Still, Deploying Agile Practices in Organizations: A Case Study. In Proceedings of the EuroSPI, LNCS 3792, pp. 16 – 27. Springer-Verlag Berlin Heidelberg. 2005.

[21] T. Chau, F. Maurer, Knowledge Sharing in Agile Software Teams. In Logic versus Approximation, LNCS 3075, pp. 173-183. Springer-Verlag Berlin Heidelberg. 2004.

[22] T. Chau, F. Maurer, Tool Support for Inter-team Learning in Agile Software Organizations. In Advances in Learning Software Organizations. LNCS 3096, pp. 98-109. Springer-Verlag Berlin Heidelberg. 2004.

[23] J. Mateos-Garcia, J. Sapsed, Adopting Agile and Scrum Practices as Organizational Becoming. In Proceedings of the British Academy of Management Annual Conference, pp. 9-11. Harrogate, UK. Sep., 2008.

[24] G. Baxter, I. Sommerville, Socio-technical systems: From design methods to systems engineering. In Interacting with Computers, vol. 23, pp. 4–17. Elsevier. 2011.

[25] S. D. Müller, L. Mathiassen, H. H. Balshøj, Software Process Improvement as organizational change: A metaphorical analysis of the literature. The Journal of Systems and Software, vol. 83, pp. 2128–2146. Elsevier. 2010.

[26] K. Conboy, Agility from First Principles: Reconstructing the Concept of Agility in Information Systems Development. In Information Systems Research, vol. 20, no. 3, pp. 329–354. September, 2009.

[27] A. Y. Cabral, M. B. Ribeiro, A. P. Lemke. M. T. Silva, M. Cristal and C. Franco, A case study of knowledge management usage in agile software projects. International Conference on Enterprise Information Systems (ICEIS), pp. 627-638, Milan, 2009.

[28] T. Dybå, T. Dingsøyr, Empirical studies of agile software development: A systematic review. Information and Software Technology, 50:833–859, August 2008.

[29] B. R. Staats, D. J. Brunner, D. M. Upton, "Lean principles, learning, and knowledge work: Evidence from a software services provider", in Journal of Operations Management, doi:10.1016/j.jom.2010.11.005, 2011.

[30] F. K. Y. Chang and J. Y. L. Thong, "Acceptance of agile methodologies: A critical review and conceptual framework," in Decision Support Systems, vol. 46, no. 4, Mar. 2009, pp. 803-814, doi:10.1016/j.dss.2008.11.009.

[31] M. Cohn and D. Ford, "Introducing an agile process to an organization," Computer, 36(6)74-78, June 2003.

[32] B. Schatz and I. Abdelshafi, "Primavera Gets Agile: A Successful Transition to Agile Development," IEEE, 22(3)36-42, May/June, 2005.

[33] A. Strauss, Qualitative analysis for social scientists. New York, Cambridge University Press, 1987.

[34] C. B. Seaman, "Qualitative Methods in Empirical Studies of Software Engineering," IEEE Transactions on Software Engineering, vol. 25, no. 4, July/August 1999.

[35] R. K. Yin, Case Study Research: Design and Methods Applied Social Research Methods. Sage Publications, Inc., 4th edition, 2008.

[36] T. Diefenbach, "Are case studies more than sophisticated storytelling?: Methodological problems of qualitative empirical research mainly based

on semi-structured interviews", Quality & Quantity, vol. 43, no. 6, pp. 875-894, DOI: 10.1007/s11135-008-9164-0. 2009.

[37] J. Corbin, A. Strauss, Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory. Sage Publications, Inc, 3rd edition, 2007.

[38] I. Nonaka, H. Takeuchi, “The knowledge-creating company: how Japanese companies create the dynamics of innovation”, Oxford University Press, 1995.

Current State of Reference Architectures in the Context of Agile Methodologies

Vinícius Augusto Tagliatti Zani, Daniel Feitosa, Elisa Yumi Nakagawa
Dept. of Computer Systems
University of São Paulo - USP
PO Box 668, 13560-970, São Carlos, SP, Brazil
{vinicius, feitosa, elisa}@icmc.usp.br

***Abstract*—Software architectures and reference architectures have been playing a significant role in determining the success of software systems. In particular, reference architectures have emerged, achieving well-recognized understanding of specific domains, promoting reuse of design expertise and facilitating the development of systems. In another perspective, agile methodologies have been widely adopted as a promising iterative, incremental and collaborative software development process, including by the software industry. Considering the relevance of reference architectures, initiatives of agile methodologies exploring these architectures are also found. However, there is a lack of a panorama about the uses, impacts and perspectives of such architectures in the agile context. The main objective of this paper is present an detailed view about how reference architectures have been used in the context of agile methodologies. For this, we applied Systematic Review, a technique to systematically explore, organize, summarize, and assess all contributions of a specific research area. As main result, we have observed that reference architecture and agile methodology should be more investigated together. Furthermore, we intend to contribute to open perspectives of new and important research lines.**

I. Introduction

Software architectures have received increasing attention as an important subfield of Software Engineering [1]. Software architectures play a major role in determining system quality, since they form the backbone to any successful software-intensive system [2]. Motivated by these considerations, software architecture research area has grown up and, as known, it has accumulated important knowledge that has certainly contributed to the software development. In this context, reference architectures have emerged as a special type of architecture that provides major guidelines for the specification of concrete architectures of systems of a given domain [3]. They can therefore promote reuse of design expertise by achieving solid, well-recognized understanding of a specific domain. Considering their relevance, reference architectures for different domains have been proposed and successful used [4]–[7].

In another perspective, aiming at organizing the software development activities, different software processes has been proposed. At one extreme, there are industry-tested processes for software system development and for effective project management, such as RUP[1] (Rational Unified Process); at the other extreme, there are agile methodologies (also called agile methods or agile processes), such as XP[2] (eXtreme Programming) and Scrum[3], that have also attracted the attention of software industry, since by adopting these methodologies, industry has speed up the time-to-market of their products [8]. In general, agile methodology aims at achieving that by simplifying the release scopes and thus it generates less effort on complexity, management and keep the customer constantly satisfied and present during the software development [9].

It is worth highlighting that the agile methodology principles seem to contrast with the use of reference architectures in agile methodologies. However, considering the relevance of reference architectures, there are initiatives exploring reference architectures together with such methodologies [10]. Nevertheless, the extent of these researches are not broad enough to comprise how reference architecture has been explored within agile methodologies. In other words, there is a lack of an detailed panorama about why, how, when and which agile methodologies have currently been adopted reference architectures.

The main objective of this paper is present an detailed view about uses of reference architectures in agile methodologies. For this, we conducted the systematic review [11], a technique that comes from Evidence Based Software Engineering (EBSE) and makes possible to explore, organize, summarize, and assess all contributions of the current state of a research area. As results of our systematic review, we can point out benefits that can be gained by exploring reference architectures within the context of agile methodologies. However, we have observed that reference architecture and agile methodology should be more investigated together, aiming at exploring their advantages in a coordinated manner. Moreover, we intend that this work

[1]http://www-01.ibm.com/software/awdtools/rup/
[2]http://www.extremeprogramming.org/
[3]http://www.scrumalliance.org/

could identify interesting and important perspectives for future research.

This paper is organized as follows. In Section II background on reference architecture, agile methodology and systematic review are presented. In Section III we present the conducted systematic review. In Section IV we discuss about achieved results. Finally, in Section V we summarize our contributions and discuss perspectives for further work.

II. Background

During the 90s, the concept of reference architecture emerged [12] and since there it has more and more attracted attention of both academy and industry. Reference architecture is a special type of architecture that captures the essence of the architectures of a collection of systems of a given domain. The purpose of a reference architecture is to provide guidance for the development of architectures for new systems or extended systems and product families. In other words, they can be seen as a knowledge repository of a given domain. Therefore, as already stated previously by Basili et al. [12], when correctly implemented, reference architectures can yield to several benefits, including: productivity gains, accelerate the time-to-market, increase the overall product quality, streamline the software development, lower the production costs and the maintenance effort. It is worth highlighting that the increase of productivity, acceleration of time-to-market and the arise of quality are benefits shared with agile methodologies [8].

The first association of software process with the word *agile* was used in 1998 [13], although many claim that the agile methodologies themselves existed long before they were formalized. Later in 2001, a group of practitioners of those methodologies gathered and summarized their experience in what was named *The Agile Manifesto*[4]. In short, its main principles are [9]: individuals and interactions over processes and tools; working software over comprehensive documentation; customer collaboration over contract negotiation; and responding to change over following a plan. However, it is not possible to exactly define agile methodologies, since specific practices vary with the methodologies. In spite of that, short time boxed iterations with adaptive, evolutionary refinement of plans and goals is a basic practice that these methodologies share [9]. Even apparently against the agile methodology principles, the use of reference architectures in agile methodologies seems to be interesting, aiming at exploring advantages of both in a more effective software development.

In another perspective, it is noticed that when a research area is mature, there is almost always an increase in the number of reports and results made available. During the study of a new knowledge area, researchers usually conduct a bibliographical review (almost always an informal review) to identify publications related to a specific subject. However, this kind of review do not use a systematic approach and do not offer any support to avoid bias during the selection of the publications that will be analyzed. Thus, it is important to have mechanisms to summarize and provide an overview about an area or topic of interest. For this EBSE has investigated and proposed the use of systematic review technique [11]. In this context, an individual evidence (for instance, a case study or an experimental study divulged in a publication/paper) which contributes to a systematic review is called primary study, while the result of a systematic review is a secondary study. Systematic review aims at providing an overview of a research area to assess the quantity, quality and type of primary studies existing on a topic of interest. In short, systematic review is conducted by planning, conduction of search and screening of primary studies using inclusion and exclusion criteria [11]. Besides that, systematic review also conducts data extraction and quantitative and qualitative analysis through the of the primary studies. Thus, systematic review has been increasingly conducted in Software Architecture area [14], [15] and seems also to be interesting in our work.

[4]http://agilemanifesto.org

III. Conducted Systematic Review

The main objective of our systematic review was to possibly identify all primary studies that explore reference architectures in the context of agile methodologies. This systematic review was conducted from October/2010 to January/2011 and involved three people (one researcher in software engineering, one specialist in systematic review and one graduate student). In order to conduct the systematic review, we have used the process presented in Figure 1. In short, it is composed by three steps: (i) Planning of the systematic review; (ii) Conduction of the search and data extraction; (iii) Reporting of results of the qualitative and quantitative analysis. These steps are explained in more details during presentation of our systematic review. Following, we detail each step.

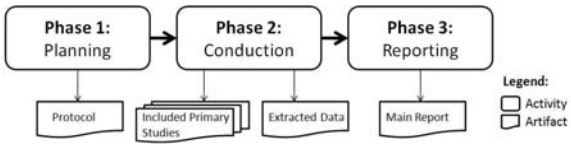


Fig. 1. Systematic review process (Adapted from [11])

A. Step 1: Planning

In this phase, we established the systematic review plan. For this, we specified: (i) research questions; (ii) search strategy; (iii) inclusion and exclusion criteria; (iv) data extraction and synthesis methods.

1) **Research Questions**: These questions are structured corresponding to the objective that is intended with

the systematic review and drive the review for further steps. Considering that our objective is the identification of the detailed view involving reference architectures and agile methodologies, the following research questions (RQ) were established:

RQ1: What is the current state of the adoption of reference architectures in agile methodologies?

RQ2: What are the benefits and limitations provided by the adoption of reference architectures in agile methodologies?

RQ3: Which are the agile methodologies that have adopted reference architectures?

2) **Search Strategy**: In order to establish the search strategy, considering the research questions, we identified initially the main keywords, as illustrated in Table I. We considered possibly all terms and their synonymous in order to cover the range of published studies. It is worth highlighting that the keywords chosen must be simple enough to bring many results and, at the same time, rigorous enough to cover only the desired research topic.

TABLE I
KEYWORDS SEPARATED BY AREA

Area	Keyword
Reference Architecture	"Reference Architecture", "Reference Model"
Agile Methodologies	"Agile Methodology", "Scrum", "Extreme Programming", "Agile Unified Process", "Feature Driven Development", "Agile Modeling", "Rapid Application Development", "Lean Software Development"

The search string is then built based on the keywords and using boolean connectors (such as AND/OR). The search string used in our systematic review was:

```
("Reference Architecture" OR
 "Reference Model")
AND
("Agile Methodology" OR
 "Scrum" OR "Extreme Programming" OR
 "Agile Unified Process" OR
 "Feature Driven Development" OR
 "Agile Modeling" OR
 "Rapid Application Development" OR
 "Lean Software Development")
```

In addition to the research questions and search strategy, we selected larger publications databases as sources of primary studies: IEEE Xplore[5], ACM Digital Library[6], Springer Link[7], Scirus[8], Web of Science[9], ScienceDirect[10], and SCOPUS[11]. Those sources are cited in [16] as trusted and efficient to the Software Engineering context. Furthermore, we decided that only papers written in English would be considered, since English is more widely adopted to write scientific papers.

3) **Inclusion and exclusion criteria**: The selected primary studies need to be assessed for their relevance, enabling the inclusion of studies that provide evidence for research questions as well as the exclusion of studies that do not. This is achieved by the definition of the Inclusion Criteria (IC) and Exclusion Criteria (EC). Thus, the inclusion criteria of our systematic review were:

IC1: The study involves software development using reference architectures and agile methodologies;

IC2: The study presents benefits and limitations by exploring reference architectures in the context of agile methodologies; and

IC3: The study presents which agile methodologies have explored the use of reference architectures.

The exclusion criteria established were:

EC1: The study does not address the use of reference architecture in agile methodologies;

EC2: The study presents an abstract and/or an introductory section that seem related to reference architecture and agile methodology; however, the rest of the text is not in fact related to;

EC3: The study does not present any abstract or it is not available for further reading;

EC4: The study is written in a different language than English;

EC5: The study is directly related to another primary study of the same author; and

EC6: The study consists of a compilation of work, for instance, from a conference or workshop.

4) **Data extraction and synthesis methods**: In order to extract data, we plan to analyze each research question. This analysis must synthesize results aiming at facilitating to obtain conclusions. During the extraction process, the data of each primary study will be independently extracted by two reviewers. If disagreement occurs, discussion will be conducted.

B. Step 2: Conduction

In this step, we conducted the search of primary studies and obtained them according to previously established plan. This was achieved by performing a search in the selected databases using the previously established search string. In order to enable the correct results to be delivered, the

[5]http://www.ieeexplore.ieee.org
[6]http://www.portal.acm.org
[7]http://www.springer.com/lncs
[8]http://www.scirus.com/
[9]http://www.isiknowledge.com
[10]http://www.sciencedirect.com
[11]http://www.scopus.com

search string were adjusted for each specific database and its mechanisms. As a result, a total of 351 primary studies were found; also, 85 studies were duplicated, and thus we analyzed in fact 266 studies. Domain experts pointed out two other studies, which we also included in our analysis, expanding the number of studies to 268.

To support the organization and manipulation of the primary studies, we used JabRef[12], an open source reference manager system. It made possible to store information on the primary studies (including, for instance, title, authors and abstract), as well as the inclusion/exclusion criteria applied to select the primary studies.

Following, the inclusion/exclusion criteria were then applied to select the relevant primary studies. Table II summarizes the number of primary studies obtained in each database and those selected after applying the selection criteria. From a total of 353 primary studies previously identified, only six studies (i.e. 1.70 %) were selected as relevant. It is observed that Scirus was the most effective source, because although it returned the smallest result set (two studies), one was relevant for our systematic review. In Table III we presents the six selected primary studies (S1 to S6). From a total of six included primary studies, one was published in 2003, another in 2006 and four in 2008. This does not indicate statistically a trend, but a growth in the interest for that research area. It is important to note that only studies published until December/2010 were considered in our systematic review.

TABLE II
PARTIAL AND TOTAL AMOUNT OF PRIMARY STUDIES INCLUDED AND EXCLUDED

Source	Included	Excluded	Total Found
ACM Digital Library	0	5	5
IEEE Xplore	1	96	97
Scirus	1	2	3
Springer Link	1	103	104
ISI Web of Knowledge	0	3	3
ScienceDirect	0	132	132
Scopus	1	6	7
Domain Experts	2	0	2
Total	6	347	353

C. Step 3: Reporting

In this step, a qualitative and descriptive analysis was conducted on each selected primary study, enabling us to achieve more accurate conclusions. Linden et al. [17] (study S1) presented a case study using a reference architecture together with XP methodology in order to develop a software system for the medical area as a distributed, component-based and standard-based system with a multi-tier architecture. For this, a reference architecture, more specifically the Reference Model for Open Distributed Processing (RM-ODP)[13], was used. According to Linden et al. [17], this architecture made possible the use of software components; component reuse and refactoring saved development time and cost, at the same time assuring the software quality. This study does not explicitly address the benefits or disadvantages of using reference architecture together with XP; however, besides advantages provided by XP, such as possibility of managing volatility in requirements and TDD (Testing-Drive Development), the adoption of reference architecture has contributed in a positive way. Since this system is based on a reference architecture, according to the authors, it could be ported to another domain without much effort.

Other two studies – S3 [20] and S5 [19] – adopted agile methodologies in the context of Software Product Line (SPL) [22], that has the reference architecture (also referred as product line architecture) as one of the main basis elements to the software development. Specifically, Kakarontzas et al. [20] adopted an agile practice — the TDD — as a complementary activity to the software development using SPL. As main result, TDD made easier the evolution of the software components of a product line and enabled better reuse decisions. In spite of SPL presents sometimes a more rigid process, TDD can work harmoniously in this context [20]. Another work was conducted by Hanssen and Fægri [19] that presented a case study in the context of the software industry where they integrated SPL and practices from agile methodologies. This case study observed that SPL and its reference architecture creates a controlled environment and solid planning while agile practice exploits the creativity potential of the developers and constant releases. In other words, SPL is more responsible for the long-term strategy planning, while agile practice enables the short-term dynamism. In the same perspective of SPL, Carbon et al. [18] (study S2) combined practices of agile methodologies and Product Line Engineering (PLE). In particular, Carbon et al. proposed an iterative process inserting agile practices within a reuse-centric environment of PuLSE-I, an application engineering process that is a part of PuLSE (Product Line System and Software Engineering), a process for PLE [23]. Its objective is to establish a reference architecture that is flexible in respect to a class of anticipated changes, at the same time that agile practices accelerate the application engineering process and the maintenance of the reference architecture. As main conclusions, they pointed out that for projects in which changes happen with frequency and can be predicted with certain accuracy, flexible up-front design pays off in the long run, despite being more time consuming at first. On the other hand, for projects which will last for an unknown period and

[12]http://jabref.sourceforge.net/

[13]http://www.rm-odp.net/

TABLE III
SELECTED PRIMARY STUDIES

Study	Authors	Title	Publication Year	Source(Database)	IC
S1	Linden, H. van der, Boers, G., Tange H., Talmon, J. and Hasman, A. [17]	PropeR: a multi disciplinary EPR system	2003	ScienceDirect	IC1, IC2
S2	Carbon, R., Lidvall, M., Muthig, D. and Costa, P. [18]	Integrating Product Line Engineering and Agile Methods: Flexible Design Up-front vs. Incremental Design	2006	Domain Expert	IC1, IC2, IC3
S3	Hanssen, G. K. and Fægri, Tor E. [19]	Process fusion: An industrial case study on agile software product line engineering	2008	ScienceDirect	IC2, IC3
S4	Fernándes, J.D., Martínez-Prieto, M.A., De La Fuente, P., Vegas, J., Adiego, J. [10]	Agile DL: Building a DELOS-Conformed Digital Library Using Agile Software Development	2008	Scopus	IC1
S5	Kakarontzas, G., Stamelos, I. and Katsaros, P. [20]	Product Line Variability with Elastic Components and Test-Driven Development	2008	IEEE Xplore	IC1
S6	Hadar, E. and Silberman, G. M. [21]	Agile architecture methodology: long term strategy interleaved with short term tactics	2008	Domain Expert	IC1, IC2

for which changes can not be anticipated, it seems better to use incremental approaches. Moreover, it was identified that there are several potential benefits of the adoption of both PLE and agile methodologies, such as increase of agility of a product line organization, error reduction, and increased time-to-market. Nevertheless, this work stated also that more researches are needed on the combination of those approaches.

Fernándes et al. [10] (study S4) described a development team that used Scrum to implement a digital library system using a reference architecture of that domain. As main benefits, this study pointed out reduction in the development effort and less misunderstandings during the software development. The nature of Scrum allowed the delivery of several intermediate versions of the software. Scrum enabled the creativity in a controlled environment and constant feedback on the deliveries, where the reference architecture dictated the horizon.

Hadar et al. [21] (study S6) introduced the CA Agile Architecture (C3A) method, which aims at providing a view for both long-term strategic architectural evolution and short-term agile incremental development. The method proposes a reference architecture documentation granularity, with a set of one-page architecture component contracts, where each of them has a responsible which keeps the contract updated. The evolution of the reference architecture together with its implementation is modeled into a cyclic process, composed by evaluation and evolution. Thus, it is possible to have agile software development at the same time that the reference architecture is built and maintained.

Following the description of each primary study, we discuss each research question established previously:

RQ1: This question is related to the current state of the adoption of reference architectures in agile methodologies. We can observe that although agile methodologies have been increasingly adopted in the software industry [8], their adoption together with reference architectures has not been sufficiently explored as a research line up to this date;

RQ2: It refers to benefits and limitations provided by the adoption of reference architectures in agile methodologies. The most of the studies have pointed out on benefits of the adoption of reference architecture together with agile methodologies than limitations. Amongst the benefits, the most cited are the reduction of the overall error rate in the projects and also the reduction of the delivery time. On the other hand, there is a lack of more detailed analysis about the negative impacts of reference architecture in the context of agile methodologies; and

RQ3: It refers to agile methodologies that have adopted reference architectures. We have observed that Scrum and XP have been used in projects involving reference architectures. It is worth highlighting that those methodologies have been pointed out as one of the most used in the software industry and, according to [8], Scrum has been used in 10.9% and XP in 2.9% of the all companies. Thus, more case studies involving Scrum, XP and reference architecture could be interesting. Furthermore, some agile practices, such as TDD, have also been explored together with reference architectures.

IV. DISCUSSION

In spite of considerable relevance of reference architecture in the software development, as well as also positive impacts of agile methodologies in the software development, including in the software industry, exploring reference architecture and agile methodologies in the coordinated manner seems to be interesting. However, our systematic review found only six works in the published literature. This works are in general recent and are important initiatives in this direction.

Regarding research questions established for our systematic review, it is observed that all of them were answered. This suggests that, the general, knowledge about reference architectures in agile methodologies has been mapped. We

believe that results presented in this work are representative of the whole software engineering domain, since systematic review has provided mechanism to achieve it.

We have also observed that the included primary studies have been published in different vehicles (four in conference proceedings and two in journals, including of different subjects, such as related to medical area). In other words, they are not concentrated, for instance, in only software architecture or agile methodology conferences. In this perspective, the conduction of a systematic review seems to be an adequate choice, aiming at finding possibly all primary studies in this context.

In spite of the achieved results, our systematic review could be conducted again, aiming at inserting also primary studies published until now. For this the systematic review protocol has been established and divulged. Besides that, relevant primary studies written in other languages can have been ignored, since we considered only paper in English. Although the databases used in our systematic review are usually considered efficient sources to Software Engineering area, other databases, such as Compendex[14] and Google Scholar[15], could be included.

Considering knowledge arisen from this work, it is possible to identify interesting and new research lines, such as (i) conduction of qualitative and quantitative analysis regarding, for instance, cost/effort reduction and software quality improvement, of using reference architecture in specific methodologies, such XP and Scrum; and (ii) exploring reference architectures in other agile methodologies.

V. Conclusions

The main contribution of this work is to present an encompassed view about the adoption of reference architectures together with agile methodologies. For this, we applied systematically a set of steps provided by systematic review. As main result, we can conclude that more attention should be given for that topic, since both reference architecture and agile methodology have contributed with considerable advantages to software development. Furthermore, we intend that this view opens perspectives of new and important research lines, contributing to a more effective and successful software development.

Acknowledgments: This work is supported by Brazilian funding agencies: FAPESP, CNPq and Capes. The authors would like also to thank the anonymous reviewers for their valious comments and suggestions, which certainly contributed to improve this paper.

References

[1] P. Kruchten, H. Obbink, and J. Stafford, "The past, present, and future for software architecture," *IEEE Software*, vol. 23, no. 2, pp. 22–30, 2006.

[2] M. Shaw and P. Clements, "The golden age of software architecture," *IEEE Software*, vol. 23, no. 2, pp. 31–39, Mar/Apr 2006.

[3] S. Angelov, J. J. Trienekens, and P. Grefen, "Towards a method for the evaluation of reference architectures: Experiences from a case," in *ECSA'08*, Paphos, Cyprus, 2008, pp. 225–240.

[4] U. Eklund, Örjan Askerdal, J. Granholm, A. Alminger, and J. Axelsson, "Experience of introducing reference architectures in the development of automotive electronic systems," *SIGSOFT Softw. Eng. Notes*, vol. 30, no. 4, pp. 1–6, 2005.

[5] A. Grosskurth and M. W. Godfrey, "A reference architecture for web browsers," in *ICSM'05*, Budapest, Hungary, 2005, pp. 661–664.

[6] E. Y. Nakagawa, F. Ferrari, M. M. F. Sasaki, and J. C. Maldonado, "An aspect-oriented reference architecture for software engineering environments," *Journal of Systems and Software*, pp. 1–35, 2011.

[7] National Instruments, "Reference architecture for mobile robotics," [*On-line*], *World Wide Web*, 2011, available in http://zone.ni.com/devzone/cda/tut/p/id/10820 (Access 01/10/2011).

[8] D. West and T. Grant, *Agile Development: Mainstream Adoption Has Changed Agility*, 2010. [Online]. Available: http://www.forrester.com/rb/Research/agile_development_mainstream_adoption_has_changed_agility/q/id/56100/t/2(Access02/10/2011)

[9] C. Larman, *Agile and Iterative Development: A Manager's Guide*. Pearson Education, 2003.

[10] J. Fernández, M. Martínez-Prieto, P. De La Fuente, J. Vegas, and J. Adiego, "Agile DL: Building a DELOS-conformed digital library using agile software development," in *ECDL'08*, Aarhus, Denmark, 2008, pp. 398–399.

[11] B. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering," Keele Univ. and Durham Univ., Tech. Rep. EBSE 2007-001, 2007.

[12] V. R. Basili, G. Caldiera, and G. Cantone, "A reference architecture for the component factory," *ACM Trans. Softw. Eng. Methodol.*, vol. 1, no. 1, pp. 53–80, January 1992.

[13] T. Dybå and T. Dingsøyr, "Empirical studies of agile software development: A systematic review," *Inf. Softw. Technol.*, vol. 50, no. 9-10, pp. 833–859, August 2008.

[14] R. Farenhorst and R. C. Boer, *Software Architecture Knowledge Management*. Springer, 2009, ch. Knowledge Management in Software Architecture: State of the Art, pp. 21–38.

[15] L. B. R. Oliveira, K. Felizardo, D. Feitosa, and E. Y. Nakagawa, "Reference models and reference architectures based on service-oriented architecture: A systematic review," in *ECSA'2010*, Kopenhagen, Denmark, 2010, pp. 360–367.

[16] T. Dybå, T. Dingsoyr, and G. K. Hanssen, "Applying systematic reviews to diverse study types: An experience report," in *ESEM'07*, Los Alamitos, CA, USA, 2007, pp. 225–234.

[17] H. van der Linden, G. Boers, H. Tange, J. Talmon, and A. Hasman, "PropeR: a multi disciplinary EPR system," *Int. Journal of Medical Informatics*, vol. 70, no. 2-3, pp. 149 – 160, 2003.

[18] R. Carbon, M. Lidvall, D. Muthig, and P. Costa, "Integrating product line engineering and agile methods: Flexible design up-front vs. incremental design," in *APLE'06*, 2006, pp. 1–8.

[19] G. K. Hanssen and T. E. Fægri, "Process fusion: An industrial case study on agile software product line engineering," *Journal of Systems and Software*, vol. 81, no. 6, pp. 843–854, 2008.

[20] G. Kakarontzas, I. Stamelos, and P. Katsaros, "Product line variability with elastic components and test-driven development," in *CIMCA'08*, Vienna, Austria, Dec. 2008, pp. 146 –151.

[21] E. Hadar and G. M. Silberman, "Agile architecture methodology: long term strategy interleaved with short term tactics," in *OOPSLA'08 at 23rd ACM SIGPLAN*, 2008, pp. 641–652.

[22] P. Clements and L. Northrop, *Software Product Lines: Practices and Patterns*. Boston, MA: Addison-Wesley, 2001.

[23] J. Bayer, O. Flege, P. Knauber, R. Laqua, D. Muthig, K. Schmid, T. Widen, and J.-M. DeBaud, "PuLSE: A methodology to develop software product lines," in *SSR'99*, Los Angeles, USA, May 1999, pp. 223–234.

[14] http://www.engineeringvillage.com

[15] http://www.scholar.google.com

Neglecting Agile Principles and Practices: A Case Study

Patrícia Vilain
Departament de Informatics and Statistics (INE)
Federal University of Santa Catarina
Florianópolis, Brazil
vilain@inf.ufsc.br

Alexandre Jonatan B. Martins
NATTEX Sistemas
Florianópolis, Brazil
alexandre@nattex.com.br

***Abstract*—Agile processes are expected to follow a set of well known agile principles and practices. This paper analyses the utilization of an agile process in a project that imposes certain difficulties in meeting such principles and practices in their totality, particularly those that refer to daily face-to-face communication among team members and frequent delivery of new versions of the product in short periods of time within a predictable schedule. A process based on Scrum was adapted to be utilized in a project which had its members distributed in different locations and that required the utilization of emergent technologies not familiar to the developers.**

Keywords-Agile principles; agile methods; agile practices; Scrum

I. INTRODUCTION

Since the appearance of agile methods [1], several software development companies that refused to adopt traditional software processes (e.g. the Unified Process [2]), started to adopt agile methods. Such methods value "individuals and interactions over processes and tools" and "working software over comprehensive documentation". What can be observed in practice is that the development of low complexity applications, when performed by small teams, does not require a bureaucratic software process, which explains why agile methods seem like a good fit in those situations.

Agile methods are defined according to agile principles [1]. The processes described by such methods include the so called agile practices, that is, practices that help developers follow those principles accordingly. Some of these principles are focused on having people working together and meeting in a daily basis. In the agile method Scrum [3], for example, these principles are followed by the practice of daily meetings. Other agile principles state that working software must be delivered frequently, preferably in short periods of time within a predictable schedule. That can be accomplished in Scrum by simple practices such as having Sprints restricted to a specific duration. However, constraints eventually imposed to the development process may prevent the team from adopting even simple practices such as daily meetings or time-boxed Sprints.

This paper analyses the utilization of an agile process in a project (case study) that imposes certain difficulties in meeting agile principles and practices in their totality, particularly the ones mentioned above. The case study in question refers to the development of a web application for storing and managing electronic documents. One of the difficulties faced in this project was that the development team, though small, did not work every day at the same place, which prevented the adoption of daily meetings. Another difficulty was that the application being developed, though of relatively low complexity, involved the use of emerging technologies that the development team was not familiar with. This generated certain uncertainty around iteration planning in the sense that iterations might not be set to a fixed duration or offer the guarantee to produce working software at the end, thus neglecting the agile principles and practices already mentioned. This process utilized in the referred project was based in Scrum, adapted with practices that try to compensate for the difficulties in following agile principles and practices in their totality.

The paper is organized as follows. In Section 2, agile principles are revisited, along with the agile practices proposed in Scrum. Section 3 briefly describes the case study. Section 4 presents the constraints imposed to following all agile principles and practices for the case study, along with adaptations made to compensate for those constraints. Section 5 presents the process utilized in the case study in more detail. Section 6 contains a discussion about the adaptations necessary to the agile process. Section 7 concludes the paper.

II. AGILE METHODS AND SCRUM

Agile methods are software development methods which apply the iterative and evolutionary development, employ adaptive planning, promote incremental delivery, and include other values and practices that encourage agility. They are best suited for projects characterized by changing, speed and turbulence.

In beginning of 2001, a group of agile methods followers created the agile manifesto containing agile principles that all agile methods should follow [1]. These principles are reproduced below:

"1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

4. Business people and developers must work together daily throughout the project.

5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

7. Working software is the primary measure of progress.

8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

9. Continuous attention to technical excellence and good design enhances agility.

10. Simplicity--the art of maximizing the amount of work not done--is essential.

11. The best architectures, requirements, and designs emerge from self-organizing teams.

12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly."

A development method is considered agile if it provides agile practices that follow the principles listed above. Among the most common agile methods are: Scrum, XP, DSDM, Lean, Crystal, FDD and ASD [4]. The agile method Scrum is detailed next for being the basis of the agile process utilized in this work.

A. *Scrum*

The method Scrum was created by Ken Schwaber and Jeff Sutherland in the 90's and its purpose is to manage the system development process [3]. As it is generally devoted to process management, practices related to more specific software development activities such as requirements gathering, software design and programming are not detailed in Scrum. As observed in [5], those practices are usually borrowed from XP, and are out of the scope of this paper.

Scrum defines the following agile practices:

- *Scrum Master*. The person who is responsible for ensuring that Scrum values, practices, and rules are enacted and enforced.
- *Product backlog*. A repository containing all system requirements (features, functions, technologies, enhancements, and bug fixes). These requirements are organized in a priority list.
- *Scrum Team*. A group of people that will develop a set of items from the Product Backlog during an iteration (Sprint). It is recommended to include seven people, plus or minus two, but teams with 3 people are also allowed.
- *Daily Scrum Meetings*. 15-minute meetings that promote the face-to-face communication among Scrum Team members. During this meeting, members of the team are supposed to explain what they have accomplished since their last meeting, what they intend to do before the next meeting, and what difficulties they are facing.
- *Sprint*. This refers to one development iteration. The development must be divided into fixed sprints that take, each one, 30 days. Once project members have developed more experience with Scrum, adjustments can be made to the duration of the sprints.
- *Sprint Planning Meeting*. This is the meeting to plan the next sprint. At this meeting, the Scrum Team decides what functionalities and technologies will be developed during the sprint.
- *Sprint Review Meeting*. 4-hour meeting that happens in the last day of the sprint. This is the meeting for the team to present and review the product increment developed during the sprint. It must be coordinated by the Scrum Master.
- *Sprint Backlog*. A repository containing those requirements selected from the Product Backlog to be developed during the next sprint.

Fig. 1 briefly shows the Scrum process. Initially, all functionalities, features, and technologies are included as items in the Product Backlog. During the project, instead of being static, the Product Backlog evolves along with the product. In each sprint, the Scrum Team selects as many items from the Product Backlog as they think they can develop during the iteration, and these items are included in the Sprint Backlog as a list of tasks. Throughout the entire sprint, a short meeting conducted by the Scrum Master, called Daily Scrum Meeting, happens to review the progress and identify impediments. At the end of each sprint, an increment of the product including new functionalities is delivered; members of Scrum Team all gather for a meeting, the Sprint Review Meeting, where that product increment is inspected. Finally, the Product Backlog is reorganized and prioritized, and the items with higher priority are selected for the next sprint.

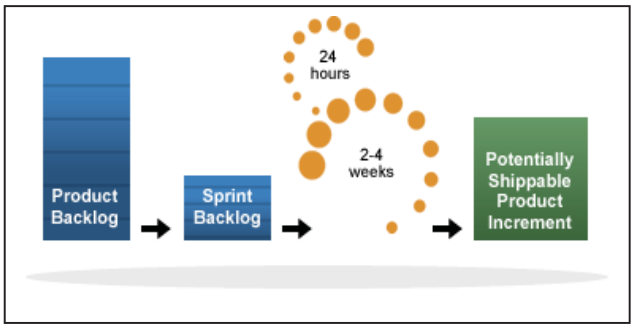


Figure 1. Scrum Process *(http://www.scrumalliance.org/learn_about_scrum)*

Scrum members can play the following roles:

- *Scrum Master*: the person responsible for the success of the project. As already mentioned, it is the Scrum Master who has to ensure that the values, practices and rules of Scrum are performed and enforced accordingly;
- *Product Owner*: the one person who is responsible for managing and controlling the product backlog;
- *Team Members*: those responsible for achieving the target goal set for a sprint.

III. CASE STUDY

The case study utilized in this paper refers to the development of a web application for storing and managing electronic documents. It consists essentially of a web-based user interface to a core document service already implemented. That core service was made available to team members as an API for the Java language. A non-functional requirement was that the implementation of such a web application should rely on the latest version of the JSF framework (i.e. Java Server Faces version 2.0) that had been recently released.

The team was composed by four people. All of project members had significant experience with Java. The Scrum Master had good experience in project management and could spend four hours a week in this project. The Product Owner also played the role of a team member (i.e. developer). This person had developed the code of the document service back end to be utilized in this project, thus having good knowledge about the application domain, which is why was assigned the role of Product Owner. This person could spend ten hours a week in this project: four hours as Product Owner, and six hours as a development team member. The remaining two people were regular team members and they were considered beginners in regard to the technology utilized in the implementation, particularly the JSF-2 framework. They could dedicate twenty hours a week to this project.

The entire project should not exceed four months due to budget restrictions, which is why adopting some sort of agile process was considered imperative by all project members.

IV. CONSTRAINTS TO THE AGILE PROCESS

The development process utilized in the case study had to be defined with the following constraints in mind:

- *Distributed Team.* Team members would not be working at the same place every day and face-to-face meetings could be held only once a week.
- *New Development Technologies.* Team members were not very familiar with some of the development technology utilized in the project.

It is important to note that, although all the participants did not work at the same place, they were all in the same city. In fact, they could meet once a week. They could also schedule sporadic face-to-face meetings in case some crucial issue had to be addressed. Therefore, this project did not have most of the difficulties commonly found in distributed agile development such as culture, time zone, communication, customer collaboration, trusting, training and technical issues [6].

The process utilized in the case study was based on Scrum. However, some of its practices had to be modified in order to cope with the constraints mentioned above. Therefore, the following decisions regarding the adaptations necessary to the process were made:

- *Utilize a web-based tool to support the process.* As project members would be working at different places in the same city, they decided to utilize a web-based tool to support the process, not only to allow all members to monitor the execution of sprints, but also to serve as a means of communication between them. In addition to providing support for managing the Product Backlog and monitoring the development of sprints, the tool in question should have some sort of wiki or blog associated with Product Backlog items (stories and their tasks) with the capability of notifying (preferably through e-mail) those involved about new entries in the wiki.
- *Eliminate daily meetings.* Despite daily meetings being one of the most important practices of Scrum [3], it was necessary to remove them from the software process because project members were unable to meet daily. The possibility of having daily meetings through video web-conferencing was considered, but there was the additional difficulty of team members having disparate working hours on certain days of the week. So, they decided to adopt only weekly meetings for monitoring the sprints. It would be up to the Scrum Master and other project members to monitor and update the entries in the wiki daily in order to identify and resolve any issues that might come up during the development of sprints.
- *Permit sprints with extendable length.* Most of the programming to be carried out in sprints was expect to utilize a new technology not yet dominated by developers. Initially, the sprints had their duration set to two weeks. However, some of the first sprints showed a delay between 10% and 25% compared to the estimated time. Project members realized that the main reason for the error in estimating sprints was the difficulty in predicting the time developers needed for learning the new technology. In a situation like that, it is recommended that the Scrum Master speak with people who understand the technology [3]. However, no one with that knowledge was available to contribute to the project. It was then decided that the Scrum Master would have the power to increase the duration of the sprints by 25% in case the delay had been caused by the lack of experience with the technology involved. Therefore, instead of being sent back to the product backlog in order to be incorporated into a new sprint (as recommended in [3]), the tasks that could not be completed by the end of a given sprint would remain in

that same sprint, which would have its length increased by up to 25% (the exact percentage should be set by the Scrum Master during the weekly meeting). The idea behind that decision was to prevent a newly created sprint from causing team members to lose their focus on the issues related to learning the new technology, and also to save the team from unnecessary frustration resulting from a significant number of sprints not being completed.

These adaptations introduced into the process were shown to be adequate to the project in question and to the profile of the project team. Table 1 summarizes the agile principles and practices that had to be neglected due to such adaptations and why they were considered necessary.

TABLE I. AGILE PRINCIPLES AND PRACTICES NEGLECTED

Principles/Practices	Constraint	Adaptation
Agile Principle: The most efficient and effective method of conveying information to and within a development team is face-to-face conversation. Scrum Practice: Daily Scrum Meeting	Team members do not work at the same time or place and are not capable of meeting every day	Longer weekly meetings to monitor the sprint Web based tool to support the process and communication between all project members
Agile Principle: Business people and developers must work together daily throughout the project	Project Owners are not available for meeting with the development team every day	Web based tool to support the process and communication between all project members
Scrum Practice: Time Boxed Sprints	It is not always possible to make a precise estimate of the time taken by sprints because of new technologies involved	Extendable Sprint that can last up to 25% beyond its initially estimated length (according to the judgment of the Product Owner at the Sprint Meeting)

V. DEFINING A NEW AGILE PROCESS

The agile process that resulted from the modifications mentioned above is generically described by the steps below. Note that the roles played by project team members are the same ones defined in Scrum: Scrum Master, Product Owner and Team Member.

A. *Definition of the Product Backlog*

This step aims to define an overview of the application, and it consists of a list with an initial description of the stories and a list of non-functional requirements. The stories are included in the Product Backlog, just like in Scrum, with the exception that, in this case, it assumes the utilization of a workgroup tool (preferably web-based) that provides support to creating and managing the backlog and monitoring the development process. As new stories are identified, they are included into the Product Backlog.

B. *Sprint Planning*

The duration of each sprint should be 1 to 3 weeks. The sprints can have their time length increased by 25% in case of error in the estimate due to the lack of experience with new technologies. Sprints of 4 weeks are not permitted because a 25% increase would result in a sprint of five weeks, inconsistent with the recommendations of Scrum.

The stories of the Product Backlog that will be part of the sprint should be selected according to the time estimated for its development, following the method Planning Poker [7] with the participation of all project members. However, considering that team members may not be familiar with the technology to be utilized in application development, or difficulties imposed by the distance between team members, it is acceptable to have only the Scrum Master and Product Owner participate in Planning Poker time estimations.

After stories have been allocated to the Spring Backlog, it is necessary to verify that the description of each selected story is detailed enough to be understood by team members. If necessary, the description of the story should be more detailed. These details could include the sketch of a screen layout, an operating example, or even part of the programming code. This is important to avoid fundamental questions about the story descriptions that would normally be resolved during the daily meetings.

Once reviewed, each story is divided into tasks. The tasks, in turn, may also be detailed, and each task is assigned an estimate of effort and time for execution. It is not required that a story is sub-divided into tasks, but it may be necessary to define a task within the story if the tool utilized to support the process requires it.

Finally, tasks are distributed to developers. This distribution is so that developers can benefit from the knowledge acquired in previous tasks regarding the technology (or component) necessary to perform those tasks.

C. *Daily Notification Monitoring*

As this process does not include the daily meetings originally proposed in Scrum, it is necessary to monitor notifications triggered by the Scrum Master or other team members. This assumes that the Scrum Master and team members are not always at the same place, and the execution of this step requires the utilization of a workgroup tool that allows team members to submit their questions and complaints to the Scrum Master.

D. *Sprint Design and Coding*

This step is devoted to applying any design techniques considered appropriate for a given project, such as sequence diagrams and design patterns, and writing the corresponding implementation code to be compiled and run in the target platform.

E. *Weekly Meeting*

Participants must schedule a weekly meeting to be held by all project members. Whenever possible, members should meet in person, otherwise through video conference. If the meeting is held at the end of the sprint (Sprint Meeting), it may last up to three hours. In such a Sprint meeting, the Scrum Master can evaluate how the stories that were not completed will be handled and whether or not the sprint will have its duration augmented. If it is just a regular meeting to monitor the sprint, the Sprint Backlog must be partially reviewed and discussed, and it can last no longer than 2 hours.

F. *Code Delivery*

After each sprint, the code must be made available to both the Product Owner and the Scrum Master so they can verify the functionality implemented in that sprint.

VI. DISCUSSION

The development of an application that utilizes emerging technologies not well known by the developers brought up some important aspects related to utilizing agile methods for that task, which are discussed next.

When a new technology is being utilized, the time spent to learn this technology has to be computed in the time allocated for the development of the stories selected in a sprint. However, it is often quite difficult to determine the time required to learn that technology, which normally leads to errors in estimation. One solution to avoid that problem is to allocate a period of training for developers prior to development. However, this can cause an unnecessary delay to the start of the project as developers can be more efficient to learn about the new technology while they are trying to address specific issues related to the tasks of each sprint. For this reason, the process described here assumed that the time required to learning that technology should be estimated as part of each sprint.

Another solution to the problem of incorrectly estimating the time allocated to each sprint would be to assign the stories and respective unfinished tasks back to the Product Backlog, as suggested originally in Scrum [3]. This solution was not adopted here to prevent team members from being held responsible for unfinished sprints. Project team members found better to allow the length of the sprint to be increased in up to 25%, provided that the delay was related to technologies that developers were not familiar with.

It is important to point out that as developers become familiar with new technology, it is no longer considered new and therefore the time that was once required for learning that technology can no longer be included in the sprint time estimation.

In the case study described earlier, the time originally allocated for development was 3 months (13 weeks), divided in 8 sprints. As a considerable number of sprints have had their duration time increased, the development ended up taking 16 weeks to complete. Fig. 2 presents a graphic showing the estimated time and the actual time spent in each sprint. Note that this project involved the development of a graphical user interface (UI). Some agile processes suggest that the design of all UI must be done prior to regular iterations [8] and some suggest that UI design is performed in parallel to product development [9]. Since the project in question could not count on a UI specialist in usability, as suggested by [10], and considering that the UI was relatively simple, it was up to the Product Owner to sketch some of the UI screens, which were utilized to complement the description of the stories.

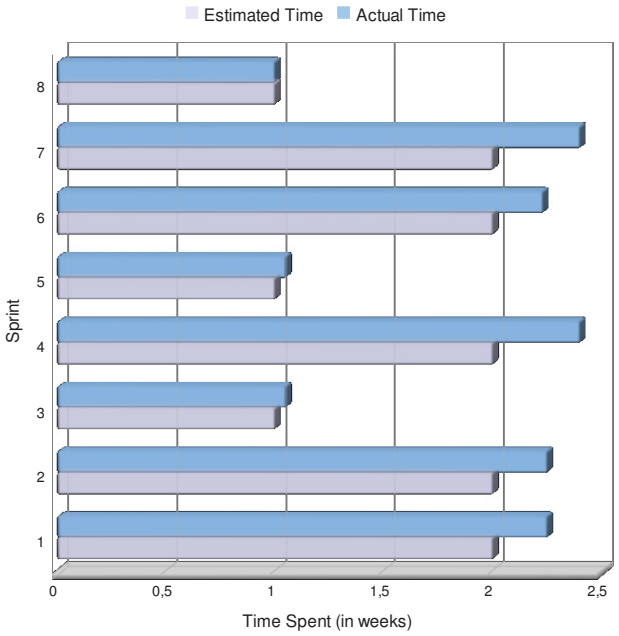


Figure 2. Duration of each Sprint

Another point of discussion is the amount of description included in each story and task. As suggested in [11], the stories can be detailed to make them more understandable for the developers. Usually, this description should be concise. However, when developers have no prior knowledge of a technology being utilized or lack a complete understanding of application requirements, detailing the stories and tasks increases their productivity in development. That also helps in situations where project members are physically apart.

As suggested in [12], it is important to modify or replace a practice, rather than simply delete it. Regarding daily meetings, an agile practice defined in Scrum, it seems possible to replace them by electronic communication without large apparent losses, provided that an appropriate tool is utilized. When there is no possibility of having daily meetings, the communication through a tool seems to be the best option to keep track of problems that are occurring daily as opposed to address these problems only after the weekly meeting.

A common practice of Scrum is to use a task board. Since the development team was not working at the same place, the use of a task board to record tasks not started, in progress and completed seemed useless. However, that was compensated by the web tool selected, which allowed to have tasks assigned to each team member and keep track of the status of each task (not started, in progress and completed).

VII. CONCLUSIONS

This paper presented the development of an application using an agile process. This process was based on the agile method Scrum, but it was adapted according to some development and project team constraints. Although neglecting some of the most known agile principles and practices, the resulting process was successfully applied to a software project without affecting the agility of its development.

The agile process defined in this work neglects two agile principles defined in the Agile Manifesto [1], particularly those that say that development teams must work together daily and that information should circulate among team members in face to face conversation. However, it was possible to observe that agile processes can also be applied to projects that do not allow such agile principles and practices to be completely followed, provided these are compensated by appropriate practices or tools. In the case study presented here the daily meeting practice had to be replaced by a weekly meeting. The lack of a daily face to face communication was compensated by the adoption of a web based tool that notifies project members about questions or issues faced throughout the sprint.

One important adaptation incorporated to the development process described here was to allow extendable sprints. Instead of having time-boxed sprints, the time allocated for each sprint could increase by 25% provided that the delay was caused by errors in estimating the time required for the developers to become familiar with the new technologies utilized in development.

REFERENCES

[1] Agile Manifesto, "Manifesto for Agile Software Development", http://www.agilemanifesto.org/, March 2011.

[2] I. Jacobson, G. Booch, and J. Rumbaugh, The Unified Software Development Process. Addison-Wesley, p. 463, 1999.

[3] K. Schwaber, M. Beedle, Agile Software Development with SCRUM. Prentice Hall, p.158, 2002.

[4] J. Highsmith, Agile Software Development Ecosystems, Addison Wesley, p.404, 2002.

[5] P. Abrahamsson, J. Warsta, M. T. Siponen, and J. Ronkainen, "New Directions on Agile Methods: A Comparative Analysis". 25th International Conference on Software Engineering (ICSE'03), 2003.

[6] M. Kajko-Mattsson, G. Azizyan, M.K. Magarian, "Classes of Distributed Agile Development Problems", Agile 2010, pp. 51-58, 2010.

[7] Planning Poker, "Play. Estimate.Plan.", http://www.planningpoker.com/, March 2011.

[8] J. Ungar, "The Design Studio: Interface Design for Agile Teams", Agile 2008, pp. 519-524, 2008.

[9] J. Ferreira, J. Noble, R. Biddle, "Agile Development Iterations and UI Design", Agile 2007, pp. 50-58, 2007.

[10] M. Singh, "U-Scrum: An Agile Methodology for Promoting Usability", Agile 2008, pp. 555-560, 2008.

[11] J. Sutherland, A. Viktorov, J. Blount, N. Puntikov, "Distributed Scrum: Agile Project Management with Outsourced Development Teams", 40th Annual Hawaii International Conference on System Sciences (HICSS'07), pp.274, 2007.

[12] S. V. Shrivastava, H. Date, "Distributed Agile Software Development: A Review", Journal of Computer Science and Engineering, vol. 1, pp. 10-17, May 2010.

Simulations of Risks for Monitoring and Prevention

MariaGrazia Fugini, Filippo Ramoni
Politecnico di Milano, Dip. di Elettronica e Informazione, Piazza L. Da Vinci, 32, 20126 Milan, Italy
fugini@elet.polimi.it, filippo.ramoni@yahoo.it

Ronald Israels
Quint Wellington Redwood, De Oude Molen 1, 1184 VW Amstelveen, The Netherlands
r.israels@quintgroup.com

Claudia Raibulet
Università degli Studi di Milano-Bicocca, Dipartimento di Informatica, Sistemistica e Comunicazione, Viale Sarca, 336, U14, 20126, Milan, Italy
raibulet@disco.unimib.it

Ovidiu Constantin
Oracle Italia, Viale Fulvio Testi, 136, Cinisello Balsamo, Milan, Italy
ovidiu.constantin@oracle.com

***Abstract*—This paper deals with a tool developed to support training and education in the field of security and safety in work areas, plants, or industrial environments. The tool comprises the dashboards enabling administrators and users to simulate risks and accidents to be ready for interventions in case of occurrence. Based on a computational model of risks, able to identify the causes and the signs which precede accidents, to enact prevention mechanisms, and to execute repair actions in order to avoid accidents, this paper presents a Risk Simulation Management (*RSM*) system and its functioning on some examples.**

Keywords-risk; simulation of risks; preventive strategies.

I. INTRODUCTION

Most accidents in work areas (e.g., plants or construction) are announced by events signalling risky situations, which may be identified and managed through preventive strategies before they evolve into actual accidents. In other cases, an accident determines an emergency situation that should be addressed through corrective strategies. The advances available in IT, service-based software, telecommunications and sensors networks, and safety garments may be exploited successfully in the identification of the risky situation which usually precedes and announces accidents, and their proper and timely management in order to avoid accidents or limit their damages.

In this paper, we describe our approach to risks in work areas and a prototype aimed at promoting education and risk-awareness through the simulation of risks. The idea behind our prototype is that risks are explicitly modelled, computed, and managed. To achieve this objective, we propose a prototype based on a risk computational model which identifies the environmental entities meaningful for the representation and computation of risks.

The contribution of this paper consists in the explicit modelling and computation of risks and their management functions. We define thresholds to identify operating modes (e.g., normal, risky and emergency), and propose a *probabilistic* computational model to handle risks in work areas and to put in places preventive strategies. When the situation is beyond the emergency threshold, a *deterministic* approach is applied and the system provides corrective strategies to handle the emergency. Software and hardware components for risk detection (e.g., sensors and detection devices) are simulated, so that we are able to embed simulated features of the available advances in areas such as location-awareness through antennas or tagging through RFID technologies. A risk is treated in a person-centred way, in that we handle a risk only when it occurs to persons during their interaction with tools, during the movement of machineries, or in relation to their protection from risky events. Currently, we deal with single risks only. The issues concerning compositions of risks are part of future developments.

The paper focuses on the architecture for implementing a *Risk Simulation Management (RSM)* system and introduces a scenario to train people to work in a risky environment by using our *RSM*. In particular, in previous work [9] we have explored how technologies can provide information about the individual who wears the computer sensors and services and the world which this individual interacts with. We also assume a Gaussian distribution for each risk source [14].

This paper is organized as follows. Section II introduces the risk management and assesment issues. Section III presents our approach to risk modeling. Section IV describes the simulation prototype. Section V addresses related work. Section VI contains conclusions and further work.

II. RISK MANAGEMENT AND ASSESSMENT ISSUES

The purpose of risk management is to obtain a balance between success, challenge, and failure. Risk management is aiming to foresee risks and prevent failures. But as the risk has its price, the prevention is not costless. For this reason, the central element of an enhanced risk management method is to make risks and the mitigating actions financially opportune. This makes it possible to establish the wanted balance.

A risk is the product of the possible effect of a threat and the chance that this threat occurs. In this paper we present our approach to risk identification and management based on the *Monitoring, Analyzing, Planning, Executing (MAPE)* loop [6].

The main elements in risk management are to identify and classify threats and to define the mitigating actions. For the identification of threats, there are two basic methods. The first is by brainstorming and collecting possible hazards to the desired situation (can be a steady state or project). A more

systematic method is to use predefined lists of hazards. Those lists are industry related e.g., banking and insurance, chemical industry, ICT [8], aerospace and circumstance related e.g., ICT outsourcing in banking and insurance [1], projects (e.g., the Prince2 project management method).

In Figure 1, the overall model for risk management with all affecting factors is shown. We see that a risk has an interaction with Actors, is of different Types, has various Effects and receives different Responses. Such modeling framework tackles risks from all viewpoints: technical, organizational, operational/tactical/strategic, and human-related issues.

For the classification of a risk effect, the simplest approach is to evaluate the effort in *financial terms*. Every ultimate effect of a threat is that it costs money to remove this effect. However, there are more possibilities to classify the effects of a risk. In the banking and insurance industry these are 'reputation' and the 'license to operate'. In other industries (like healthcare and the chemical industry) the amount of human damage will be a more important class of effects. In the end it helps to align all the classes to make it possible to count the balance between risk and remedy.

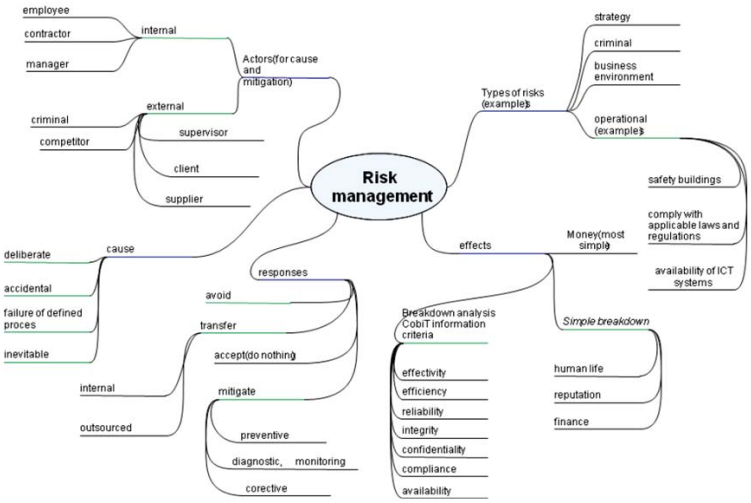


Figure 1. Comprehensive Risk Management Model

The mitigating actions consist out of three classes which are *prevention* (making the hazard cannot occur and/or the effect of the hazard minimal), *monitoring* (checking if the hazard does not occur) and *correction* (minimizing and removing damages when the threat takes place). *Simulation* is an example of a preventive risk management method. It makes it possible to check the completeness of mitigating actions and to train how to monitor and to act when a threat takes place.

III. RISK MODELING

Our approach *to risk modelling and simulation* starts with the technical points of the Risk Model in Figure 1. Using the MAPE loop, we observe an environment, detect the anomalies by analyzing the data collected through monitoring, decide if a risky situation occurred and if intervention/change is needed (and of what type), and put in place (execute) the planned modifications in the environment. Such a loop implements a self-healing [6] *RSM* able to detect and manage risky situations through *preventive* and/or emergencies through *corrective* actions. Specific preventive actions are put in place when the risk is between the *Risk Threshold* and the *Emergency Threshold.* Since some monitored parameters (e.g., the temperature) can reveal arising risks, the idea of preventive actions is that parameters can be evaluated using a *probabilistic* approach that computes the probability for the out-of-range parameter to evolve into a dangerous situation. *Preventive actions* are then suggested by the system to take the abnormal parameter back to normal values before the risk evolves further. Such actions range from checking a source of gas loss to sending preventive alarms to persons thorough their PDAs. Beyond the Emergency Threshold, there is an emergency and we move to a deterministic approach, using off-the-shelf, real-time *corrective actions* that patch the situation immediately.

A risk (or risky situation) occurs when one or more parameters through which the environment is monitored are out of their ordinary operation range. The causes (or sources of risks) may be various: loss of substances, machinery movements, or the interaction between the workers and tools. The computation of the risk (or emergency level) is supported by the pre-computed risk map related to the environment. The risk map is determined considering the location of risk sources and protection elements (including emergency exits).

The remaining of this section introduces the environment model and the risk computation through an example.

A. Entities Modeling

The overall modelling at the basis of Risk Simulation is centred on the *Environment*, the *Persons*, the *Tools* and the *Machineries*. These elements are both subject to risks and generate risk. *Informative Devices* (e.g. sensors) and *Protection Elements* (e.g., security garments) allow risks to be identified, avoided and/or faced. An entity can be either *static* (its location is fixed within the work area: a tool for example) or *mobile* (its location changes along time: persons or machineries).

A *plant* is a *ComplexEnvironment*, which may include open and closed areas. The topology of a plant is defined by the physical location of each area, its dimension attribute, and the proximity relationship. Each area has at least one access point (for entrance and exit), which may be used as emergency exit. We indicate the plant elements which may either be the source of risks or influence the computation of risks and their level, namely: access points (in/out openings); location of water and gas pipes and electrical wires; and windows, doors, emergency exits, and other unprotected open spaces.

A *person* represents a worker in the environment with an associated profile, containing: (1) *skills*: related to ability in use of tools and machineries; (2) *experience*: related to knowledge of security procedures and work organization; and (3) *roles*: worker, team head, visitor, administrator.

Persons use *tools* (e.g., hammers, saws, drills, diggers) or *machineries* (e.g., trucks). Both *tools and machineries* are characterized by the functionalities they offer and their fabrication date (an indicator of the risk in using them).

Informative devices model the technological elements needed for surveillance of the environments. They collect information from the environment and convey it to the *RSM* and receive information from a surveillance centre, allowing persons to be notified about the current situation in the area.

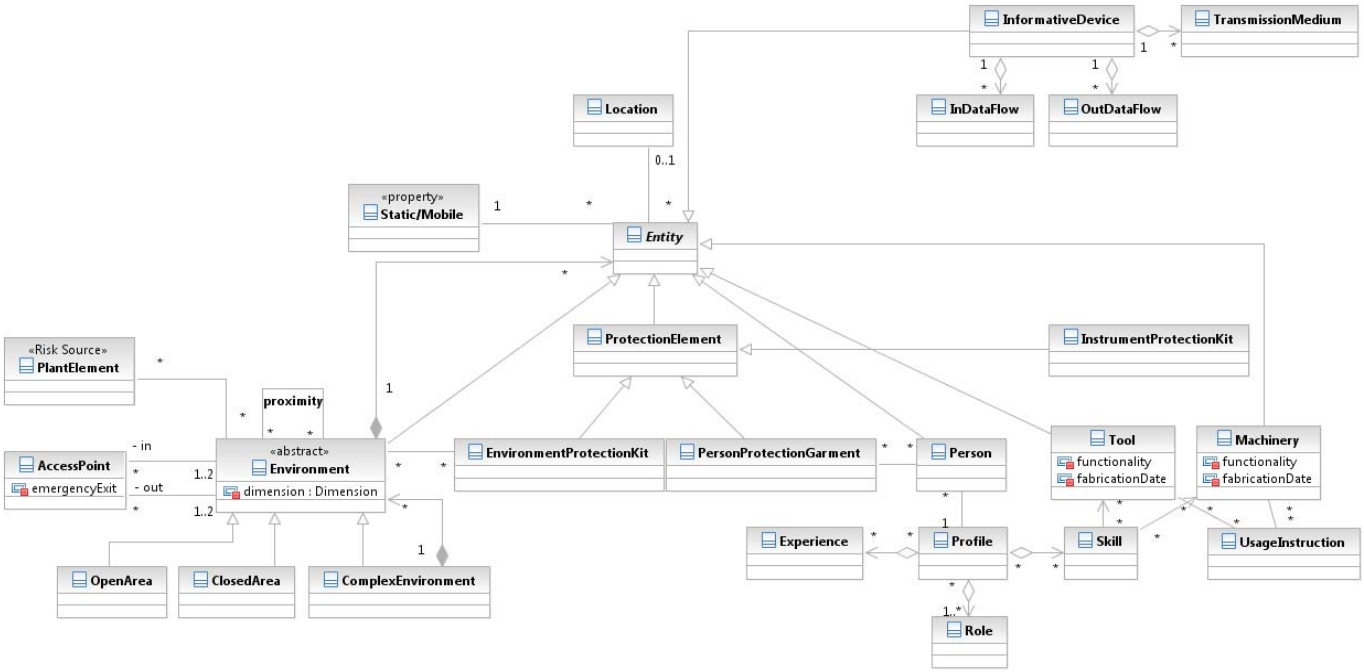


Figure 2. Modeling Entities for Risk Simulation

Protection elements are entities whose role is to prevent risks/emergencies by providing protection to persons, tools, and machineries. They are of three types (see Figure 2): (1) *person protection garments*: objects providing physical protection to persons (i.e. helmet, boots, jacket); (2) *environment protection kits*, which help persons to protect the environment in case of difficulty, or to escape quickly if necessary (e.g., fire protection kits and evacuation kits); and (3) *instrument protection kits*, which provide physical protection to devices, tools, and machineries.

B. Risk Computation

Suppose the environment structure is composed of five areas (see Figure 3). For each area, a risk map is defined. Assume that area A has a gas pipe loosing gas slowly and a team is present with tools and one machine (e.g., a truck).

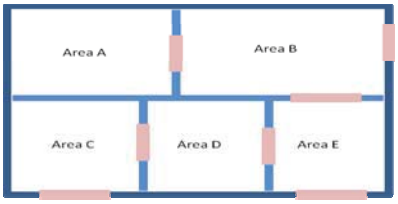


Figure 3. Map of a Work Environment

The *Risk Map* of Area A is defined as follows.

RiskMap (A) {
Fixed_Part = {Access Point (*Xap*,*Yap*), Window (*Xw*,*Yw*), GasPipe(*Xgp*,*Ygp*), DetectionDevice},
Dynamic_Part = {Team [(person,role)], [Tools], Moving Machine, Activity}
}

where *X* and *Y* are coordinates of access points, windows, gas pipe and informative devices. The fixed part describes the area structure while the dynamic part describes who is in the area doing which activity and is computed through functions illustrated in what follows. Detectors signal a situation of increasing gas saturation and a risk event is reported. The risk in Area A is computed by the function:

Evaluation_Value=REF(Risk Map, parameters)

where *REF* denotes a *Risk Evaluation Function,* the Risk Map is as given above and *parameters* identifies all elements monitored in the environment not included in the Risk Map which are meaningful to determine the risk (e.g., gas). *Evaluation_Value* is a value expressing the risk for each parameter. If the value is beyond a given value (fixed for that particular parameter), a potential risk is signalled.

The *RSM* contains the definitions of the *REFs* which may be set through a dashboard: the system administrator sets an evaluation function for each parameter, which associates an evaluation to each value detected by the informative devices.

Each person has an associated *Personal Risk Level (PRL)* which has to be combined with the Risk Map of the area. The *PRL* depends on the: (1) physical location in the environment, (2) role of the person; (3) tools and machineries used; (4) physical state of the person.

Formally:

PRL=f (location, role, tools and machinery)

Role, tools, and machinery are evaluated using an opportune evaluation function, while the location is evaluated using the risk map; *f* is a suitable evaluation function. A possible choice is given by the following function (where eval stands for evaluation), namely:

PRL= Σ (eval(location), eval (role), eval (tool,FabricationDate), eval(machinery,FabricationDate))

Each person has also an associated *Personal Protection Level (PPL)* dependent on the: (1) role and experience; (2) protection garment, that protect persons; and (3) device (e.g., a PDA) enabling a person to communicate with other persons.

Formally, we have:

$$PPL=f(role, experience, protection, device)$$

Role, experience, protection garment and device are evaluated using an opportune evaluation function; *f* is a suitable evaluation function. A possible choice for *f* is:

$$PPL = \Sigma(eval(role), eval(experience), eval(protection), eval(device))$$

where *eval* is a function defined to evaluate the various components of the *PPL*. *PRL* and *PPL* are continuously computed for each person, using the evaluation functions onto the values measured from the environment. Using the environmental risk map, the risk related to the location of workers can be retrieved, and using the information on tools/machinery used and protection garment the risk level of the worker is computed. When a person moves around the environment, his *PRL* is compared instantaneously with the *PPL*, defining the *Personal Risk Index (PRI)* as follows:

$$PRI=PRL-PPL$$

This is an indicator of the existing risk for the person at a given moment determined as a statistic measure computed without considering how the person influences the environment, but rather how the environment creates risky situation for a person. If the computed *PRI* exceeds the risk threshold, the *RSM* tries to reduce the *PRI* by choosing the best strategy. Risk Facing Rules are defined as sets of strategies to face a risk (gas leak in our example). A *Strategy* is an ordered list of actions to be executed in the indicated sequence to mitigate the risk or to handle the emergency. A simple *Risk Facing Rule* putting in *or* two strategies for "gas leakage" risk are reported in Table 1. In order to choose the most appropriate Risk Facing Rule and strategy, the root causes of the high value of *PRI* have to be identified. By knowing the *f()* used to compute the *PRL*, the inverse function f^{-1} *(PRL, Parameter)* returns the main parameter causing the risky situation. Based on this information, the suitable preventive strategy can be chosen.

Here, the *Risk Facing Rule* has two alternative strategies: *Strat1* is a "softer" preventive strategy, where actions allow a team to plan their leaving from the dangerous area, while *Strat2* is a preventive strategy whose set of actions implies evacuation. Strategies contain actions made of a message part (denoted under the *to* part) directed to a tool, a team and even to a single person, such as Team Leader in *Strat1* and Worker n. 7 in *Strat2*. For instance, in Table 1, Worker 7 might have special health requirements and hence is directly notified, while in *Strat1* both the Team leader and the specific Team (Team A) are specified as targets of the alarms.

TABLE I. RULES AND STRATEGIES FOR A GAS LEAKAGE RISK

Strategy	Action	
	To	***Message***
Strat1	Gas Sensor 1	Check gas concentration level
	Team leader	Send sms
	Team A	Recover work tools
	Team A, Team leader	Leave area A and go in area B
Strat2	Gas Sensor 1	Check gas concentration level
	Sound alarm 1	Activate
	Worker 7	Group people in area D

IV. THE SIMULATOR

The *RSM* prototype is composed of the following elements [15]. The Data Collector receives the current values of the monitored parameters related to the environment from the informative devices (e.g., sensors, RFID, antennas), which are located in the environment to monitor its status and to generate alarms. Data can arrive also from users. The Data Collector forwards these data to the Environment Data Manager, which stores them in the Environment Database. The System Database contains data regarding the observed system (e.g., topology, user profiles workers) which do not change frequently over time. The main task of the Data Analyzer Manager is to check the current values of parameters (i.e., if they are out of the admitted range). When it detects parameters beyond of a given threshold, it notifies the Risk & Emergencies Manager, which is composed of a Risk Detector, an Emergency Detector and a Strategy Manager. While the first two identify the current risk/emergency based on the parameters values received from the Data Analyzer module, the Strategy Manager identifies the appropriate strategy to be put in place to address the current situation, based on a set of risk facing rules available in the Risk & Emergency Database containing the description of all the risks/emergencies which can be addressed by the system. Once the strategy has been identified, the Risk & Emergencies Manager contacts the Execution Manager to apply the strategies. The messages to be sent to the environment elements (persons, informative devices or protection elements) are managed by the Communication Manager, which chooses the appropriate format for the messages (e.g., sound, alarm, sms). A Business Intelligence Module analyzes the risks and emergencies (e.g., the frequency of their occurrence, reaction time, number of involved persons) to improve the future results of the *RSM*.

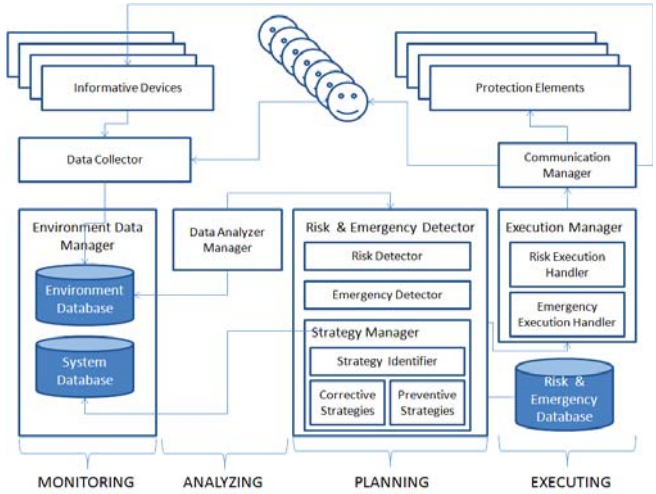


Figure 4. *RSM* Prototype Architecture (taken from [16])

The aim of our simulation is: (1) to make available to the *RSM* the data that would be collected by sensors and devices (sensor networks, RFIDs, antennas, tags), (2) to simulate the sensor communication techniques, and (3) to simulate the risk reaction process in terms of alarms, actions, and strategies.

To wrap the environment in a simulation, each entity of the environment has a corresponding simulated element, so that a

1:1 correspondence exists between the object instantiated in the simulation and the physical object/person to be controlled. The simulation is centred around the user's (Worker) actions, to focus on the Human-Environment interactions. One basic case is the Simulation of Worker's Actions. These actions are described in external XML files containing the sequence of actions and their duration. The actions currently implemented in the prototype concern movement actions (*goto*), use of tools (e.g., *usehammer*), use of transport means (e.g., *ontruck*, *offtruck*), and health-related actions (e.g., faint). An action has a duration, a maximal execution time to be completed, and an attribute denoting the number of times it has to be repeated.

Our simulator models the behaviour of environment elements which provoke risk events. A selected set of resources have been implemented: (1) *Gas Pipe*: in one use case of our simulation, one pipe breaks due to hammer misuse; this is modelled as a buffer filled one unit at a time corresponding to each hammer blow; and (2) *Gas*: this is simulated through a linear function that augments the radius of a circle centred in the pipe breakpoint, and covering an area equal to that of the area interested by the expanding gas.

The simulator shows the communication and protocols of instruments and devices in order to provide data from sensors and tags to the *RSM*. Such communication occurs as the value change of some variables: each simulated object that transmits data has and associated modelled object. To modify these parameters (e.g., a value read by a sensor) the sensor uses the reference to the object of System and that, via suitable methods, modifies the values of the attributes. Some simulated objects have no reference to objects of the environment. For example, a gas pipe in the simulation is used only to model the interactions with other elements of the *RSM* to show what occurs in the environment, but is not an active technological component and has no direct link to the *RSM*. Such object has a software component counterpart in the *RSM*, necessary for the localization services, in case, for instance, during a gas leak, the identification of the leak source is needed.

A screenshot of our prototype shows a sample Risk Map (see Figure 5): the third box of the first row reports the risk map of the working area showed in the second box. On the upper side of the room, an emergency exit is located, while on the right side a source of risk - a gas pipe - is located. The gas pipe creates a risk zone, identified by a red and yellow area, while the emergency exit decreases the risk in its vicinity. The workers present in the area also contribute to the risk map. The current prototype implements a set of simple prevention and correction strategies and allows one to modify the description of the environment by allowing the upload of a Risk Map definition in an external XML file.

V. RELATED WORK

Risks in work environments are usually addressed technically through sensors, ad-hoc devices (wearable devices [9]) and suitable software for empowering the individuals' lives. A general approach to risk management has to consider also social, knowledge management, and decision-making approaches. In [12], emphasis is on key decision structuring steps and analytical tools to help ensure the systematic treatment of fact-based and value-based risk knowledge claims. The introduction of intelligent capabilities in work protective equipment through ICT technologies is one of the priorities of several international safety and health organizations [22]. The overall problem of risks and safety, with particular attention to the way of communicating risks to people and companies is discussed in [20]. From the technical perspective, sensors and sensor networks [3], [4], [13], [18] are popular in the development of software for various application domains (i.e., scientific contexts [21], habitat monitoring [2], structural health monitoring [7], augmented reality [17]). [11] considers that sensors provide some of the basic input data for risk management of natural and man-made hazards. A current problem pointed out in the literature [11] is achieving interoperability between different sensor networks. In practice, each remote data source may use distinct encodings, formats and even communication protocols [5] leading to a lack of standardization and interoperability problems when multiple sensor networks are combined. From the software viewpoint, interesting approaches are service-oriented [3], [4] due to their modularity, standardization

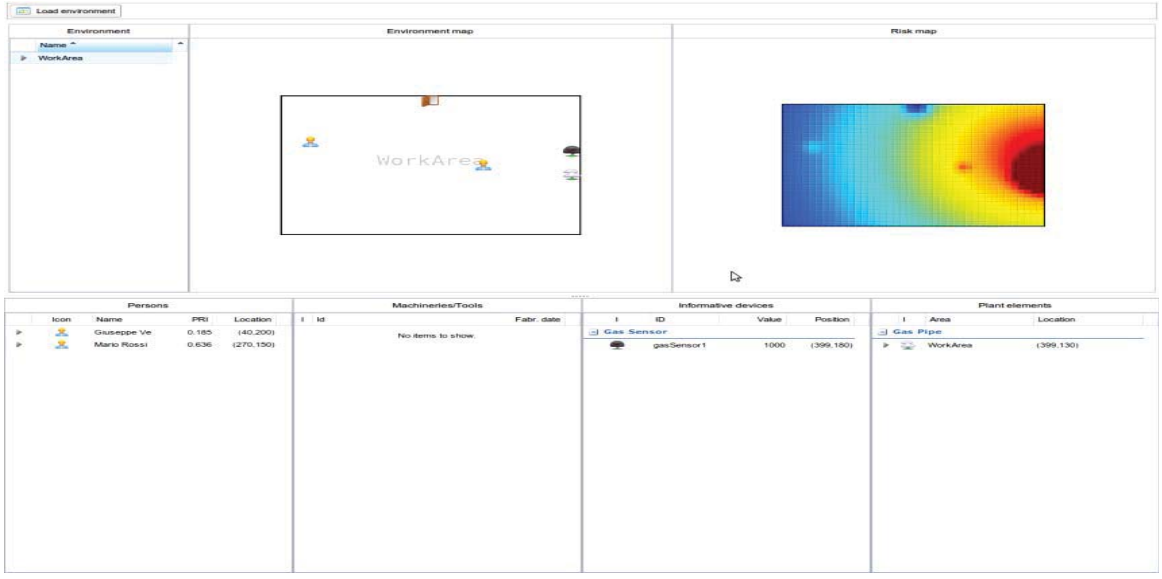


Figure 5. Risk Map of a Work Area

possibilities, support to self-healing approaches and adaptation. In [19], environmental risk assessment and decision-making strategies use comparative risk assessment and multi-criteria decision analysis for the incorporation of project stakeholders' opinions in the ranking of alternatives.

The European Commission (EC) has funded a number of Integrated Projects concerned with the accessibility of data and services for risk management (e.g., ORCHESTRA, http://www.eu-orchestra.org/ and SANY, http://sany-ip.eu/).

VI. Conclusions and Further Work

This paper has presented an approach to simulate the monitoring and prevention of risks in work areas. The definition of risk is related to the characteristics of the environment, the persons working in it, the work machineries and the tools used in the environment and of the work actions performed by the persons inside this environment. Design problems arise when an attempt is made to identify the relations among these entities in order to identify the risk. We have therefore introduced a model relating people, tools, machinery moving in a space, and elements and locations which are potential risky elements. We have described risk sources generating and subjected to risk and ways to prevent, monitor and correct risky situations. Technology devices are modelled in that they offer a protection (e.g., sensors, antennas, RFID, alarms) as well as protection garments (helmets, protection shoes and so on), which embed no IT elements but are a must for accessing the work areas. Finally the behaviour of persons must be modelled. This is one the hardest modelling tasks, since it implies the determination of what a worker can do at each moment, the teams he is participating in (more than one at a time, in general), and other run-time aspects. There are further approaches we plan to consider in the next future such as the PDCA (Plan-Do-Check-Act) or Deming cycle [10]. Risk management starts with analysis of the actual plans, in- and external rules and threats. For serious risks (relative high chance and large effects) responses (mitigating actions) are planned. Responses must be controlled and further, the actual risk determined. Our current work regards the implementation of a full- fledged simulator endowed with the administrator's dashboard of the *RSM*. We are also working on refinement of the choice of suitable risk evaluation functions related to the personal risk in real environments, and on criteria for the selection of best strategies to face risk and the evaluation of a strategy application in terms of risk mitigation.

Acknowledgment

This research has been partially financed by the Italian TEKNE Project. We thank G. Tomasino for implementation of the simulator.

References

[1] Basel Committee, On Banking Supervision, Outsourcing in Financial Services, http://www.bis.org/bcbs/

[2] Bitsch Link, J. Á., Bretgeld, T., Goliath, A., and Wehrle, K. RatMote: A Sensor Platform for Animal Habitat Monitoring. In Proceedings of the 9th ACM/IEEE international Conference on Information Processing in Sensor Networks, ACM, pp. 432-433 (2010)

[3] Botts, M., Percivall, G., Reed, C., and Davidson, J. Sensor Web Enablement: Overview and High Level Architecture, OGC® Sensor Web Enablement: Overview and High Level Architecture. In GeoSensor Networks, Lecture Notes in Computer Science Publisher Springer Berlin, Heidelberg, Vol. 4540/2008, pp. 175-190 (2008)

[4] Chen N., Di L., Yu G. and Min M., A Flexible Geospatial Sensor Observation Service for Diverse Sensor Data-based on Web Service. ISPRS Journal of Photogrammetry and Remote Sensing Vol. 64, pp.234–242 (2009)

[5] Chen C. and Helal S. Sifting Through the Jungle of Sensor Standards. Pervasive Computing Vol. 7, Issue 4, pp. 84–88 (2008)

[6] Cheng, B. H. C., de Lemos, R., Giese, H., Inverardi, P., and Magee, J. Software Engineering for Self-Adaptive Systems. LNCS 5525, Springer (2009)

[7] Chintalapudi K., Paek J., Gnawali O., Fu T. S., Dantu K., Caffrey J., Govindan R., Johnson E., and Masri S. Structural Damage Detection and Localization using NETSHM. In Proceedings of the 5th Intl. Conf. on Information Processing in Sensor Networks (2006)

[8] COBIT, Enterprise Risk, http://www.cioindex.com/it_governance/articleid/975/cobit-as-a-risk-management-framework.aspx.

[9] Conti, G.M., Rizzo, F., Fugini, M.G., Raibulet, C., Ubezio, L. Wearable Services in Risk Management. In IEEE/WIC/ACM International Joint Conferences on Web Intelligence and Intelligent Agent Technologies, Web2Touch Workshop, pp. 563-566, IEEE Press, Milan, Italy (2009)

[10] Deming, W.E. Out of the Crisis, MIT Center for Advanced Engineering Study, MIT Press (1986)

[11] Douglas J, Usländer T, Schimak G, Esteban J F, and Denzer R.,. An Open Distributed Architecture for Sensor Networks for Risk Management, Sensors 8, pp. 1755-1773 (2008)

[12] Failing, L., Gregory, R., Harstone, M. Integrating Science and Local Knowledge in Environmental Risk Management: A Decision-Focused Approach. In Ecological Economics, Vol. 64, Issue 1, pp. 47-60 (2007)

[13] Franceschini, F., Galetto, M., Maisano, D., Mastrogiacomo, L. A Review of Localization Algorithms for Distributed Wireless Sensor Networks in Manufacturing, International Journal of Computer Integrated Manufacturing, Vol. 22, Issue 7, July, pp. 698-716 (2009)

[14] Fugini, M.G., Raibulet, C., Ubezio, L. A Web-Service Architectural Perspective on Risk Management in Work Environments. In Proceedings of the 2010 Workshop Living Experience through the Web (Web2Touch), in conjunction with NOTERE'2010, pp. 327-332, IEEE Press, Tozeur, Tunisia (2010)

[15] Fugini, M.G., Raibulet, C., Ramoni, F. A Prototype for Risk Prevention and Management in Working Environments. In Information Technology and Innovation Trends in Organizations, to appear in LNCS, (2011)

[16] Fugini, M.G., Raibulet, C. A Self-Healing Approach to Risk Management in Work Environments, ERCIM News 84, 38-39 (2011)

[17] Kealy A, and Scott-Young S. A Technology Fusion Approach for Augmented Reality Applications. Transactions in GIS 10, pp. 279–300 (2006)

[18] Laguna, M. A., Finat, J., and González, J. A. Mobile Health Monitoring and Smart Sensors: a Product Line Approach. In Proceedings of the 2009 Euro American Conference on Telematics and information Systems: New Opportunities To increase Digital Citizenship, Prague, Czech Republic, ACM Press, New York, 1-8 (2009)

[19] Linkov I., Satterstrom F.K., Kiker G., Batchelor C., Bridges T., Ferguson E. From Comparative Risk Assessment to Multi-Criteria Decision Analysis and Adaptive Management: Recent Developments and Applications, Environment International, Vol. 32, Issue 8, Environmental Risk Management - the State of the Art, pp. 1072-1093 (2006)

[20] Lundgren, R., E., and McMakin, A.H. Risk Communication. A Handbook for Communicating Environmental, Safety, and Health Risks, 4th Edition, Wiley Publ. (2009)

[21] Martinez K., Hart J. K., and Ong R. Environmental sensor networks. IEEE Computer Vol. 37, Issue 8, pp. 50–6 (2004)

[22] Rubio, J., Padillo, I., Espina, J., Verástegui, J., López-de-Ipiña, J. RFID in Workplace Safety Solutions. In Proceedings RFID Systech 2010, European Workshop on Smart Objects: Systems, Technologies and Applications, Ciudad, Spain, (2010)

Flexible Support for Adaptable Software and Systems Engineering Processes

Richard Mordinyi, Thomas Moser and Stefan Biffl
Christian Doppler Laboratory "Software Engineering Integration for Flexible Automation Systems"
Vienna University of Technology
Vienna, Austria
{*firstname.lastname*}@tuwien.ac.at

Deepak Dhungana
Siemens AG
Vienna, Austria
deepak.dhungana@siemens.com

***Abstract*—Typical complex software and systems engineering projects involve a set of different engineering disciplines and therefore heavily rely on systems integration approaches. Service-oriented architecture and Enterprise Service Bus provide a valuable basis for systems integration; however often depend on tedious manual work or invasive and therefore inflexible adaptations in case of even minimal changes to engineering processes or engineering tools. In this paper we describe the Engineering Service Bus (EngSB), an integration platform which not only integrates tools and systems, but also provides a bridge between engineering processes and existing engineering tools. We evaluate the EngSB by implementing a standard software engineering process for multi-engineering environments. Major results were that the EngSB allows for non-invasive and flexible integration of well-established tools in multi-engineering environments, while enabling the integration to take place on the level of engineering processes.**

Keywords-(software+) engineering, non-invasive integration, component-based architectures.

I. INTRODUCTION

Todays real-world and large-scale software and systems engineering projects such as the engineering of water power plants or steel mills typically involve a wide range of different engineering disciplines such as mechanical engineering, electrical engineering and software engineering [1]. Each of the disciplines already provides its specific engineering tools and engineering systems to manage specific engineering processes. Each of these processes heavily relies on different technical platforms and heterogeneous data models. Well-established software engineering methods and approaches (e.g., iterative development processes, issue tracking systems etc.) could provide additional value for building and managing such information systems engineering projects, but require careful integration and seamless collaboration with other engineering fields to achieve industrial acceptance. Therefore, this kind of cooperation can be considered as "(software+) engineering" [2].

There is already individual tool-support available for each engineering process step, but the integration of those tools to form a single integration solution heavily relies on manual work and invasive changes to tools, and thus does not provide an automated and non-invasive solution including support for the integration of entire engineering processes. Furthermore, current rigid integration solutions introduce complete tool sets supporting the whole engineering process, but are inflexible regarding even minimal changes in standard engineering processes or tool sets. However, typical real-life engineering processes very often comprise specific and highly specialized tools and also often alter existing standard engineering processes according to their needs. Therefore, large-scale engineering projects seek for a flexible framework that facilitates a non-invasive support of multi-disciplinary engineering processes in case of tool exchangeability, as well as reusability and adaptation of process descriptions without changing existing tools.

In this paper, we describe in detail the Engineering Service Bus (EngSB) integration platform we have successfully used in several industrial projects for the integration of heterogeneous automation systems engineering tools. The EngSB is an integration platform based on the ESB and not only integrates different tools and systems, but also provides a bridge between engineering processes and existing engineering tools [1, 3]. The so-called engineering workflow is used as basis for the non-invasive engineering tool integration and to automate the integration; this workflow is directly related to the original engineering process and represents the implementation of the engineering process in the EngSB platform. The EngSB addresses requirements such as the capability to integrate a mix of user-centered tools and backend systems, mobile work stations that may go offline, and flexible and efficient configuration of new project environments and SE processes.

In this work we focus on the application of the EngSB to multi-engineering environments and describe the detailed architecture of the relevant EngSB integration platform components. Furthermore, we discuss the EngSB concept as a technical platform for the integration of different engineering tools and for bridging engineering processes and engineering tools. Additionally, we evaluate the EngSB by applying it to a standard software engineering process for the information systems engineering domain, the Continuous Integration and Test (CI&T) process [4], and comparing the results to other related approaches, like with the general Application lifecycle management (ALM) approach and as a specific implementation of

the process with Hudson[1]. The specific configuration of the EngSB with respect to the given process results in the concrete OpenCIT[2] (a continuous integration and test server) implementation.

The remainder of this paper is structured as follows: Section 2 summarizes related work on aspects of technical integration and related implementations of research groups or industrial projects. Section 3 describes the use case scenario. Section 4 details the EngSB implementation, which is evaluated and discussed in section 5. Finally, section 6 concludes the paper, and identifies further work.

II. Related Work

This section summarizes related work on current methods, concepts and approaches to integrate distributed environments, architecture concepts for tool integration, and integration framework specifications.

A. Tool Integration in Distributed Environments

The most common integration patterns have been summarized in [5] whereas messaging pattern results in a huge number of middleware frameworks available [6], mostly with the Enterprise Service Bus (ESB) concept [7]. The ESB has been successfully applied as agile integration platform for back-end services in distributed heterogeneous business software environments [7]. Key strengths of the ESB [8] are providing distributed integration and separating the description of the business logic from the integration logic in contrast to design patterns such as client-server architecture [9] or Messaging. Nevertheless it has to be taken into consideration that too much abstraction does not help software engineers integrate their environments [10, 11] as the reusability of the different components is limited. ESBs provide components which can be reused to integrate different functionality or protocols, but only little research is done regarding tool and process abstraction, requiring system integrators to start again nearly from the scratch for each new project.

The ESB is also used as the underlying infrastructure for Service-oriented Architectures (SOA) [11] which intend to support service composition and application evolution [12] aiming at higher reusability, shorter time to market and lower costs. Although the concepts are well adapted [11, 13], the large number of different standards, specifications, technologies and high requirements in analyzing the organization make SOA hard to understand and complex to implement. Additionally, SOA itself does not provide any concepts for business process event handling. SOA notifications are basically implemented by defining services which directly call other services. This requires rewriting the service, notifying the system, for each logical change. With the help of service orchestration the logic can be extracted from the services themselves into a meta layer. However, this only moves the problem into a higher level of abstraction as the meta-layer has to be rewritten every time an additional service has to be notified.

B. Integration Environments

In general, analyzing the different concepts for integration developed over time, three approaches were able to become defacto standards for integration in software engineering:

Script based integration approaches describe the build process of software only. Modern approaches like Maven [14] also integrate test-, deployment-, announcement- and documentation environments to cover the complexities of current software engineering projects. Although Maven is one of the most used build systems for software development in the Java world, its reusability and flexibility is limited: first, the Maven workflow is hard coded into its core. It can be extended by attending plug-ins to the different lifecycle goals, but it cannot be changed completely and adapted for other uses.

Another approach of tool integration is to focus on the developer within its **Integrated Development Environment** (IDE) [15] like in Eclipse[3] or Microsoft Visual Studio[4]. They provide integration modules like Mylyn which supports a complete, task centered integration of issues retrieved from a wide range of different issue trackers. Although an IDE brings the entire engineering environment to the developer the logic and process models have to be implemented with each environment. Furthermore, they do not support automation of complex processes over multiple, different IDEs.

Application lifecycle management (ALM) based integration approaches usually comprise source control system and integration workflows, deployment, monitoring, project management and testing, while at the same time covering the entire project lifecycle [16]. An ALM is the Eclipse Application Lifecycle Framework (ALF) [17] which handles integration challenges by introducing a central negotiator that manages interactions between applications. By using an intermediate communication format the event manager prevents to integrate applications several times with several other applications. It allows a single integration on the central node, which carries out communication with other ALM applications through orchestration of their corresponding web services. However, ALF is no longer supported by the Eclipse Foundation [18].

III. Use Case

This section presents a (software+) engineering industrial use case from signal engineering that has been retrieved from an industrial partner developing, creating, and maintaining hydro power plants. The standard mo del of the Continuous Integration and Test (CI&T) process known from software engineering consists of a set of activities: checking out source code artifacts from a source code management system, building the system, running tests, performing analyses regarding the outcome of these tests, and finally reporting test results to interested roles. The CI&T use case shows a key feature of an iterative software development process: if parts of a system or engineering model get changed, the system has to be rebuilt and tested in order to identify defects early and to provide fast feedback on the implementation progress to the project manager and the owners of the changed system parts.

[1] http://hudson-ci.org
[2] http://opencit.openengsb.org
[3] http://www.eclipse.org/
[4] http://msdn.microsoft.com/en-us/vstudio/default.aspx

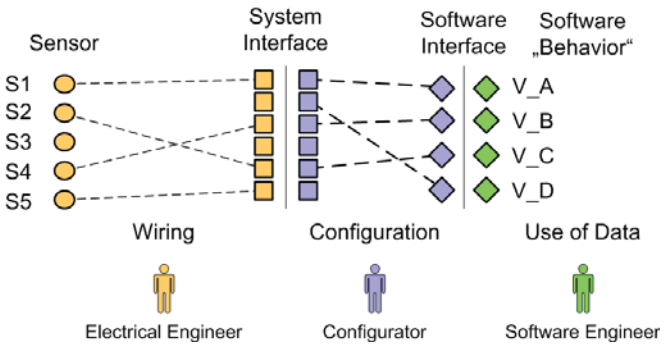


Figure 1: Overview End-to-End Analysis [19, 20].

In distributed multi-engineering environments (see Figure 1), typically a set of different tools and data models are used along the engineering chain. In difference to single system models, where changes of model values have direct impacts on other model elements, model value changes in cross-domain modeling require additional transformation and checks before the actual impact of a value change can be estimated or measured. To cooperate, the engineers have to exchange relevant parts of the data structures in their tools with each other with the goal of a consistent overall view on certain aspects in the project, e.g., when producing a specification for a subcontractor. This requires exhaustive communication between engineers to keep models consistent and thus introduces a variety of error sources as well. Currently, every role uses organization-, domain-, and tool-specific data formats and terms, thus the data exchange also takes considerable expert knowledge on the receiving end to make sense of the incoming data, typically as large PDF document or tool-specific import file. Applying the automated continuous integration and testing process in such an environment increased consistency and minimize the number of potential error sources.

IV. EngSB Implementation

This section gives a description of the components (Figure 2) of the Engineering Service Bus (EngSB) architecture:

Core Components. The concept of Core Components contains additional developed components providing services which could not be provided by external tools or which are not usable from external tools as required. The two most important components for the architectural need of the EngSB are the registry and the workflow component. Since different projects may be running in parallel on the EngSB, each project requires carrying a context which contains at least the project identifier allowing the ***registry*** to map onto endpoints for each accessed tool. The ***workflow component*** is responsible for engineering rule, process and event management in the EngSB. The Drools[5] business logic integration platform builds the backbone for the workflow management system since it provides a consistent solution for BPM, knowledge and event management. It includes Drools Expert as rule engine, Drools Fusion for complex event processing, Drools Flow for business process management and finally Drools Guvnor as a centralized knowledge repository to manage processes and rules centralized. The important part of the workflow component is its indirection layer between engineering rules and processes and the EngSB Tool Domains.

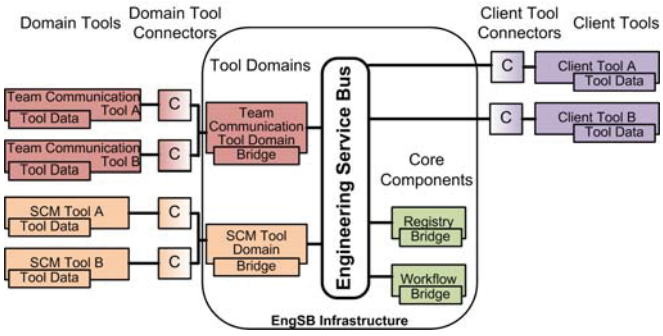


Figure 2: Components of the Engineering Service Bus.

Tool Domains. Although each tool provider gives a personal touch to its product their design is driven by a specific purpose. For example, there are many different issue tracker available, each having its own advantages and disadvantages, but all of them can create issues, assign and delete them. Tool Domains [3] are based on this idea and distill the common functionality for such a group of tools into one Tool Domain interface. They can contain code, workflows, additional logic and data, but they are useless without a concrete implementation.

Bridge. JBI does not allow components that are not deployed in the JBI infrastructure to directly interact with services. This means that tools deployed to the EngSB can directly access neither external components nor the other way round. The bridge provides a connection between these two worlds.

Client Tools. Client Tools in the EngSB concept are tools which do not provide any services, but consume services provided by Tool Domains and Core Components instead.

Domain Tools. Domain Tools, compared to Client Tools, denote the other extreme of only providing services. Classically, single purpose server tools, like issue tracker or chat server, match the category of Domain Tools best.

Domain and Client Tool Connectors. Although most (back-end) tools provide interfaces to integrate and automate them in distributed environments, they do not directly fit the needs of the EngSB. Instead, they have to be wrapped up to allow external, distributed tools to access them via the EngSB. These required bridges are called Tool Connectors. Tool Connectors wrap tools as Domain Tool Connectors to provide their services to accommodate the relevant Tool Domain with the expected interface. As Client Tool Connectors they provide a Client Tool with an access to the EngSB services. Tools can be integrated with more than one connector to act in many different domains.

V. Evaluation and Discussion

In this section we evaluate the capabilities of the EngSB related to flexibility and non-invasive support for integrating software engineering processes by reporting on conducted feasibility studies. The evaluation is based on the OpenCIT framework by applying the standard software engineering

[5] http://www.jboss.org/drools/

CI&T process (see left hand side of Figure 3) to multi-engineering environments.

With respect to the CI&T process, the right hand side of Figure 3, shows OpenCIT integrating common tools via five different domains, namely Source Code Management, Build, Test, Analysis and Notification. For each of these domains, a concrete tool instance from each used engineering environment (e.g., electrical engineering and software engineering) is connected using a tool connector. In addition, the core Engineering Workflow Rules component is used to configure the defined CI&T process as a workflow on the EngSB and to orchestrate the specific runtime events of the individual tools accordingly.

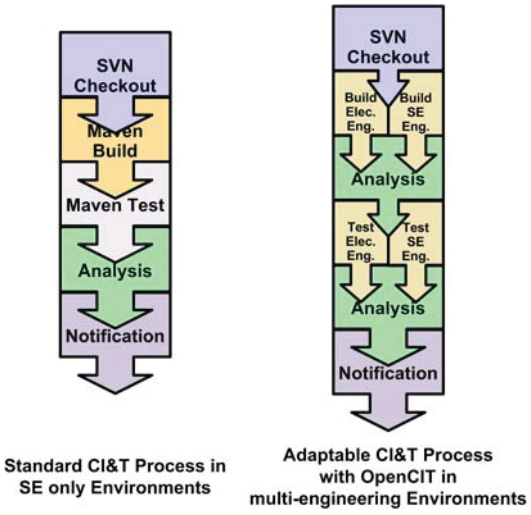


Figure 3. Overview CI&T and OpenCIT.

The evaluation has been conducted with two different tools in two phases. First, we compare OpenCIT with a highly popular CI&T tool suite (Hudson) to demonstrate how EngSB can address standard software engineering integration problems. Second, we compare EngSB with a general purpose integration framework (Eclipse ALF), which is a set of tools designed to support an enterprise application from its initial inception through its deployment and system optimization. The high level goal is to analyze the effort of adopting a new CI&T tool suite, from the viewpoint of software administrators, building tool connectors and defining/customizing the process.

Phase 1. In order to compare the effort required for integrating new tools to the frameworks, we compared the effort required for developing a connector and the mechanisms available for extending the core in Hudson and EngSB. We examine initial adoption effort and the possibility to make changes to standard CI&T workflow. Hudson provides extension points, where new plugins can be added to extend the application logic. This mechanism is used to integrating new tools to the framework. EngSB provides interfaces, which can also be extended. The extensions provide the functionality via JMS, or REST, or Web Services, allowing programming language independent integration facilities. As Hudson is meant for use in software engineering, event processing is specialized and hardcoded and may be customized only via tools integrated using extensions points. Workflows are hardcoded, which means that the core has to be extended for this purpose. In EngSB events can be designed for domains but also completely independent, which allows one to decide how tight the tools should be integrated to the workflows. Workflows are basically defined in drools.

Phase 2. We investigated the key characteristics of Eclipse ALF, a system that allows reusable process definition and adaption regardless of low-level technical details. We analyzed the flexibility regarding event processing, effort needed to modify workflows, and tool exchangeability capabilities. In Eclipse ALF events are SOAP messages which are defined in the core. New tools may react to the SOAP messages to achieve workflow modifications. In EngSB events can be designed for domains but also completely independent, which allows one to decide how tight the tools should be integrated to the workflows. In Eclipse ALF workflows are defined in BPEL leading to a heavy-weight solution which has to be defined programmatically and cannot be modified or extended at runtime. In EngSB workflows are defined in Drools. While in Eclipse ALF exchanging tools requires redefinition of the processes and event handling, EngSB supports tools to be freely exchangeable, as the workflow is defined on the level of tool domains.

While CI&T suites such as Hudson enable the integration of software engineering tools, there is no CI&T suite available for enabling CI&T processes for multi-engineering environments such as the scenario presented in Figure 1. As shown in the right hand side of Figure 3, OpenCIT can be used for migrating the CI&T process, which originally comes from the software engineering domain, to multi-engineering environments. This we define as (software+) engineering, as software engineering provides additional value to systems engineering. The tool domain concept of the EngSB allows for a non-invasive exchange of single tool instances, since tool domains define and provide the domain-specific functionality usually provided by single engineering tools. Therefore, OpenCIT has no restrictions regarding the tool sets to be used, but relies on the tool domain concept of the EngSB to specify and integrate already existing tool functionalities.

In order to enable the CI&T process for multi-engineering environments, the standard CI&T process is adapted as shown in the right hand side of Figure 3. In comparison to the standard software engineering CI&T process, the multi-engineering CI&T implementation of OpenCIT allows the definition of multiple tool domains (often from different engineering disciplines) to define the same CI&T process step (e.g., the *build* process step). When this process step is executed, OpenCIT triggers the functionalities of all tool domains that are linked to this specific process step. Furthermore, different strategies can be implemented related to the order of the triggering, e.g., the workflow engine of the EngSB can be used to implement and enforce project-specific reporting strategies regarding the outcome of a specific process step (e.g., a build failure notification). The tool domain concept of the EngSB can additionally be used to simulate engineering tool instances on the level of an engineering process. This feature can be used to test engineering process (e.g., whether they work as planned), without the need to have individual tool instances available and connected to the EngSB.

Additionally, EngSB showed more flexibility regarding the continued use of favored tools by engineers, whereas standard tool suites allows the integration of some standard software engineering tools, which may or may not be used by the engineers in a certain organizational setting. The possibility to use known tools (thorough the possibility of integrating arbitrary tools) results in minimal training effort for project participants, since the used engineering tools are already familiar to them.

VI. CONCLUSION

Today's large-scale information systems engineering projects typically involve a range of different engineering disciplines with their specific engineering tools. However, for building and organizing information systems engineering projects to manage specific engineering processes a cooperation of each discipline is required to form an integrated engineering system. Although, there is already individual tool-support available for each engineering process step, the integration of those tools to form a single integration solution heavily relies on manual work and invasive changes to tools. However, typical real-life engineering processes very often comprise specific and highly specialized tools and also often alter existing standard engineering processes according to their needs. In this paper, we presented the Engineering Service Bus (EngSB) as an integration platform based on the ESB that does not only integrate different tools and systems, but also provides a bridge between engineering processes and existing engineering tools. We demonstrated the use of the EngSB to migrate a standard software engineering process - Continuous Integration and Test (CI&T) – to multi-engineering environments and compared it with popular tools, like Hudson or Eclipse ALF. The comparison showed that OpenCIT, the multi-engineering environment integration of CI&T using the EngSB, can replace an integration framework like Hudson for projects involving multiple engineering disciplines. Adaptations to the standard workflow are more flexible in the case of EngSB because it allows arbitrary extensions through flexible event handling and exchangeability of tools. Mutual restrictions between tools and processes can be minimized through the use of workflow rules that are formulated on the level of tool domains. Furthermore, we compared EngSB with Eclipse ALF to demonstrate the EngSB provides higher abstraction level for tool integration, which makes it easier to adopt For future work we intend to perform empirical evaluations in case studies and implementations as well as creating domain-specific process definition tools to support non-experts in defining information systems engineering processes to be deployed in the EngSB.

ACKNOWLEDGMENTS

This work has been supported by the Christian Doppler Forschungsgesellschaft and the BMWFJ, Austria. We are thankful to Alexander Schatten and Andreas Pieber for their valuable input in several discussions.

REFERENCES

[1] Biffl, S., and Schatten, A.: 'A Platform for Service-Oriented Integration of Software Engineering Environments'. Proc. Eight Conf. on New Trends in Software Methodologies, Tools and Techniques, 2009

[2] Biffl, S., Pieber, A., and Schatten, A.: 'Service-Oriented Integration of Heterogeneous Software Engineering Environments', TechnicalReport QSE, ISIS, Vienna University of Technology, 2009

[3] Biffl, S., Schatten, A., and Zoitl, A.: 'Integration of Heterogeneous Engineering Environments for the Automation Systems Lifecycle', Proc. IEEE Industrial Informatics (IndIn) Conference 2009

[4] Duvall, P., Matyas, S., and Glover, A.: 'Continuous Integration: Improving Software Quality and Reducing Risk' (Addison-Wesley, 2007. 2007)

[5] Hohpe, G., and Woolf, B.: 'Enterprise Integration Patterns : designing, building, and deploying messaging solutions' (Addison-Wesly) 2004

[6] Du, Y., Peng, W., and Zhou, L.: 'Enterprise Application Integration: An Overview'. In Proc. of the 2008 Int. Symposium on intelligent information Technology Application Workshops, Washington, DC, USA2008 pp. Pages

[7] Chappell, D.: 'Enterprise Service Bus' (O'Reilly Media, Inc.), 2004

[8] Chappell, D.: 'ESB Myth Busters: 10 Enterprise Service Bus Myths Debunked', (Last visited February 22, 2010, online: http://soa.syscon.com/node/48035)

[9] Berson, A.: 'Client/server architecture (2nd ed.)' (McGraw-Hill, Inc.), 1996

[10] Woolf, B.: 'ESB-oriented architecture: The wrong approach to adopting SOA', (Last visited February 22, 2010, online: http://www.ibm.com/developerworks/webservices/library/ws-soa-esbarch/)

[11] Engels, G., and Assmann, M.: 'Service-Oriented Enterprise Architectures: Evolution of Concepts and Methods'. Proc. of the 2008 12th Int. IEEE Enterprise Distributed Object Computing Conference, Washington, DC, USA, 2008

[12] Yin, J., Chen, H., Deng, S., Wu, Z., and Pu, C.: 'A Dependable ESB Framework for Service Integration', IEEE Internet Computing, 2009, 13, (2), pp. 26--34

[13] Mike, P.P., and Willem-Jan, H.: 'Service oriented architectures: approaches, technologies and research issues', The VLDB Journal, 2007, 16, (3), pp. 389-415

[14] Sonatype: 'Maven: the definitive guide, 1st edition' (O'Reilly & Associates, Inc., 2008. 2008)

[15] Cheng, L.-T., de Souza, C.R.B., Hupfer, S., Patterson, J., and Ross, S.: 'Building Collaboration into IDEs', Queue, 2004, 1, (9), pp. 40--50

[16] Chappell, D.: 'What is Application Lifecycle Management?' (Last visited March 22, 2011, online: http://www.davidchappell.com/WhatIsALM--Chappell.pdf)

[17] Buss, T., and Caroll, B.: 'ALF Architecture - Draft: A Platform for ALM Tools Integration', 2005

[18] Manchester, P.: 'Eclipse kills open-source SOA projects', (Last visited February 26, 2010, online: http://www.theregister.co.uk/2008/11/07/eclipse_kills_soa_projects/)

[19] Biffl, S., Moser, T., and Winkler, D.: 'Risk Assessment In Multi-Disciplinary (Software+) Engineering Projects', Int. Journal of Software Engineering and Knowledge Engineering, Special Issue: Software Risk Assessment, 2011, 21, (2), pp. 1-25

[20] Moser, T.: 'Semantic Integration of Engineering Environments Using an Engineering Knowledge Base'. PhD thesis, Vienna University of Technology, 2009

Automated Detection of Likely Design Flaws in Layered Architectures

Aditya Budi, Lucia, David Lo, Lingxiao Jiang, and Shaowei Wang
School of Information Systems, Singapore Management University
{adityabudi,lucia.2009,davidlo,lxjiang,shaoweiwang.2010}@smu.edu.sg

Abstract

Layered architecture prescribes a good principle for separating concerns to make systems more maintainable. One example of such layered architectures is the separation of classes into three groups: Boundary, Control, and Entity, which are referred to as the three analysis class stereotypes in UML. Classes of different stereotypes are interacting with one another, when properly designed, the overall interaction would be maintainable, flexible, and robust. On the other hand, poor design would result in less maintainable system that is prone to errors. In many software projects, the stereotypes of classes are often missing, thus detection of design flaws becomes non-trivial. In this paper, we provide a framework that automatically labels classes as Boundary, Control, or Entity, and detects design flaws of the rules associated with each stereotype. Our evaluation with programs developed by both novice and expert developers show that our technique is able to detect many design flaws accurately.

1 Introduction

Layered architecture is a recommended industry practice as it promotes separation of various concerns into layers [17]. By using this architecture, when requirements change, most of the changes could be localized to a limited number of classes in a particular layer. Thus, no changes would be needed for classes in unrelated layers as long as the interfaces between the layers remain the same. As software evolves over time, layered architectures are more likely to have better reusability, improve comprehension and traceability, and ease maintenance and evolution tasks than single-tier architectures.

One commonly used layered architecture is the separation of classes into three stereotypes, namely: Boundary, Control, and Entity, following the Unified Modeling Language (UML) and its suggested objectory process [15, 14]. *Boundary* classes are responsible to interface with external systems or users. *Control* classes are responsible to realize particular functionalities or use cases by coordinating the activities of various other classes. *Entity* classes are responsible to model various domain concepts and store and manage system data.

Class stereotypes are not just symbols; they come with design rules governing their behaviors and responsibilities. If the nature of a class is not apparent to developers or its behaviors do not match its stereotype label, developers are prone to make mistakes, violating the rules, especially as the code evolves over time. There are two common rule variants: *robustness rules* [15], and *well-formedness rules* [14].

Unfortunately, many software projects, during development or maintenance, have little documentation. Many design documents, including those specifying stereotype labels of the classes in a program, are often missing. Such information is often not obvious in the source code either due to poor variable and class names, code changes, etc. Also, keeping design documents and stereotypes up-to-date manually could be time-consuming and error-prone.

To address the above issues, we propose a framework that can automatically reverse engineer class stereotypes and detect violations of design rules associated with them.

We empirically evaluated our proposed system on a number of student projects and a real software system. Our preliminary experiments are promising. Compared with manually stereotyped labels, our approach achieves on average 77% of accuracy. Design defects resulted from violations of robustness and well-formedness rules could be detected with up to 75% precision and 79% recall.

The rest of the paper is organized as follows. Section 2 describes related work. Section 3 presents the concept of class stereotypes and their associated design rules. Section 4 describes our design flaw detection framework. Section 5 presents evaluation results. We discuss limitations and applicability in Section 6 and conclude in Section 7.

2 Related Work

There are a number of studies on the characteristics of class stereotypes [2, 8, 1, 12]. Andriyevska et al. study the effect of stereotypes on program comprehension [1]. Kuz-

niarz et al. also study the effects of stereotypes on program comprehension but focus on user-defined stereotypes rather than the standard three (i.e., Boundary Control, and Entity) [12]. Atkinson et al. propose different de facto ways in which stereotypes are used [2]. Gogolla and Henderson-Sellers analyze the part of the UML metamodel that deals with stereotypes and provide recommendations for improving the definitions and uses of stereotypes [8]. Dragan et al. investigated an automated way to infer class and method stereotypes [6, 7]. To the best of our knowledge, we are the first to propose an automated way to automatically detect likely violations of design rules governing stereotypes based on automatically identified stereotype labels.

Various studies also address the problem of detecting and correcting design flaws and code smells [9, 11, 19, 13, 18]. Guéhéneuc et al. [9] find code segments that do not conform to a particular design pattern and transform them accordingly. Khomh et al. use Bayesian Belief Networks to detect code and design smell [11]. Vaucher et al. study *god* classes and propose an approach to distinguish good *god* classes from bad ones [19]. Moha et al. extract concepts from text descriptions and establish formal specifications of code smells so that they can detect code smells automatically [13]. Trifu and Reupke also detect structural flaws in object oriented programs and use optimization-based techniques to automatically restructure programs [18]. Our work in this paper focuses on detecting class stereotypes and checking violations of design rules involving stereotypes, which enriches the type of design information and defects detected by past studies and helps users to reverse engineer their designs.

3 Design Rules of Class Stereotypes

This paper provides a mechanism to identify class stereotypes automatically and detect design flaws in programs. The class stereotypes and design rules associated with the stereotypes are described in the following subsections.

3.1 Class Stereotypes

Identifying class stereotypesis an important step for designing, analyzing, understanding, and maintaining a software system. In particular, this paper focuses on automated identification of three class stereotypes, i.e., *Entity*, *Control*, and *Boundary*, which was introduced as an extension to the standard UML [14, 3]. The UML extension describes the responsibilities of classes belonging to each stereotype. It promotes separation of different concerns into different class stereotypes, and thus software changes related to one concern would only affect one particular stereotype involving a limited number of classes [17].

Classes with the *Entity* stereotype store and manage information in a system. In this paper, we further distinguish *Regular Entity* from a special kind of entity called *–Data Manager*, which is used to persist to storage systems (e.g., databases, file systems, etc). As an example, *Course* is a possible entity class in a University Management System (UMS) and *CourseStore* is a possible data manager class that stores and retrieves course data from databases.

Classes with the *Boundary* stereotype serve as an interface between a system and external systems interacting with it. External systems, represented as *Actors*, could be other computing systems or the users of the system. These interface classes would be the ones affected if the behavior of external systems change. In a typical UMS, *CourseManagementUI* is a possible boundary class.

Classes with the *Control* stereotype act as a glue among entity and boundary classes, and control the activities of other classes for particular tasks. For example, *CourseRegistration* class is a possible control class that interacts with a user interface class and related entity classes, e.g. *Course*.

3.2 Design Rules

Class stereotypes, based on their supposed responsibilities and the principle of separation of concern, should follow certain design rules, such as, an *Entity* class cannot call a *Boundary* class directly. Regulating the interactions among different classes of various stereotypes can help to ensure the understandability and maintainability of a software system.

Our work provides an automated mechanism for checking compliance of design rules governing the interactions among class stereotypes. In particular, we instantiate the checking against two sets of rules which reflect various architectural styles, namely *robustness rules* and *well-formedness rules*. Our checking mechanism is designed to be flexible enough to take various rules for checking.

3.2.1 Robustness Rules

Robustness analysis, as described in Rosenberg and Scott's UML book [15], provides a set of rules that indicate all valid and invalid interactions among different class stereotypes. The rules are paraphrased as follows, where *Actors* represent users of a system which could be humans or classes/objects outside the system under analysis

- R1 Actors can only call boundary objects.
- R2 Boundary objects can only call controllers or actors.
- R3 Entity objects can only call controllers.
- R4 Controllers can call entities, other controllers, and boundaries, but not actors.

3.2.2 Well-Formedness Rules

The *well-formedness rules* are defined in the UML extension [14], and rephrased as follows: [1]

[1]The complete set of rules in the UML extension also allows the subscriber-publisher style of interaction. This paper considers only interactions via direct calls and thus omits a part of the rules governing subscribe-publish interactions.

Caller	Callee							
	Actor		Entity		Control		Boundary	
Actor							R1	W1
Entity				W3	R3			
Control			R4	W4	R4	W4	R4	W4
Boundary	R2	W2		W2	R2	W2		W2

Table 1. Robustness and Well-Formedness Rules

W1 Actors can only call boundary objects.
W2 Boundary objects can call entities, controllers, other boundaries, and actors.
W3 Entity objects can only call other entities.
W4 Controllers can call entities, other controllers, and boundaries, but not actors.

The two sets of rules can also be represented as a matrix shown in Table 1. Each entry indicates a rule that validates the corresponding caller-callee relation. Unmarked entries signify bad caller-callee relations that violate the rules.

4 Design Flaw Detection Framework

Our framework are shown as a flowchart in Figure 1. We use a classification framework that has two phases, namely training and violation detection. In the training phase, we build a statistical model using a machine learning technique that can discriminate the class stereotypes based on a set of classes with given stereotypes, i.e., training data labeled with Regular Entity, Data Manager, Control, and Boundary stereotypes. In the violation detection phase, given a class or a program containing more than one class with no stereotype labels (i.e., test data), we first predict the corresponding stereotype for each class based on the model. We then detect design defects by using the inferred stereotypes verified against either robustness or well-formedness rules.

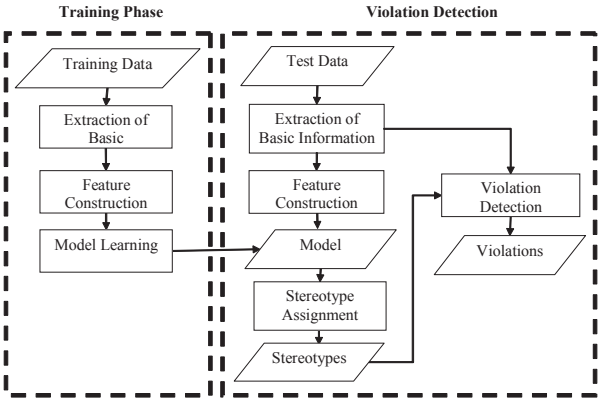


Figure 1. Design Flaw Detection Framework

There are five processes in the framework: extraction of basic information, feature construction, model learning, stereotype assignment, and violation detection. The following paragraphs describe these processes in more detail.

Extraction of Basic Information. In this process, we extract information about the classes in each Java program from its source code. The basic information we extract includes all the methods in each class, all the instructions contained in each method, the call-relations among classes (represented as call graphs), and the classes that contain operations related to I/O or database operations. We built our information extractor upon WALA [20].

Feature	Description
Size	Number of instructions that a class has
NOM	Number of methods that a class has
ASize	Average size of all the methods in a class
Fan-out	Number of other classes that a class calls
Fan-in	Number of other classes that call a class
GetCnt	Number of getters method in a class
SetCnt	Number of setters method in a class
CRUD	Number of methods performing create, read, update, or delete to data sources in a class

Table 2. Features Used in Our Statistical Model.

Feature Construction. Based on the basic information, we form features that could help in differentiating the training classes belonging to each of the four given stereotypes. In this work, we compute the set of features shown in Table 2 for each class. Instead of using absolute values for the features, we normalize their values to be in the $[0, 10]$ range.

Model Learning. In this process, we take the training data with its features and learn a model that could discriminate the four stereotype labels: Regular Entity, Data Manager, Boundary, and Controller. We use Support Vector Machine (SVM) [5] for this task since it is a well-known machine learning technique that has been shown to have good accuracies in many application domains. Regular SVMs learn models that only discriminate between two labels. We use an SVM extension handling multiple class labels [4]. Implementation-wise, we use the publicly available $SVM^{multiclass}$ [16].

Stereotype Assignment. We use the model learned in the training phase and the features extracted from the test data to assign stereotype labels to each class in the test data. We use the classification capability of $SVM^{multiclass}$.

Procedure Violation Checking
Inputs:
R : A set of rules (e.g., robustness, well-formedness)
C: A set of classes
L: The corresponding stereotypes for the classes
Output: V: A set of violations against R
Method:
1: Let V = {}
2: Let $RMap$ = Process R and represent it as a pair $\langle Caller, \{Callee\}\rangle$, where $\{Callee\}$ is the set of stereotypes that can be called by the stereotype $Caller$ as expressed in R.
3: **For each** class c in C
4: Let $Caller$ = c's stereotype
5: Let $\{Callee\}$ = $Caller$'s information in $RMap$
6: Let C' = All other classes that are called by c
7: **For each** class c' in C'
8: **If** c''s stereotype $\notin \{Callee\}$
9: $V \leftarrow V + \{c'\}$ // A violation is found
10: **OUTPUT** V

Figure 2. Violation Detection

Violation Detection. After the stereotypes are inferred, we can check for violations against a set of class design rules by

leveraging the caller-callee relations extracted from code, and the inferred stereotypes. In this paper, we consider *robustness rules* [15] and *well-formedness rules* [14]. For a set of rules, the automated rule checker performs the steps shown in Figure 2 to search for violations. At line 1, we initialize the output set. At line 2, we represent a set of rules as a set of pairs of valid interactions from a stereotype (i.e., classes with this stereotype) to other stereotypes. At line 3-10, we visit each class and for each, we extract its stereotype (line 4), other valid stereotypes that it could call (line 5), and the set of other classes called by it (line 6). At line 7-8, we check if any of the called classes has a stereotype that violate the rule (i.e., not in the set $\{Callee\}$). If this is the case, we record this violation at line 9. We finally report all violations found (line 10). For example, for the robustness rule R3, given a class with stereotype *Entity*, any call from the class to other Entity or Boundary classes will be reported as a violation. As another example, for the well-formedness rule W4, any call originated from a Control class is valid.

5 Empirical Evaluation

In this section, we describe our dataset and evaluate the accuracies of our approach in inferring stereotypes and detecting design flaws.

5.1 Dataset

We perform our evaluation on 15 Java projects developed by students of an object-oriented application development (OOAD) course. The projects are all about a single player hunting game. The number of Java classes per project ranges from 36 to 67 with an average of 45. Each project has 3431 to 9220 lines of code (including comments and blank lines), with an average of 5168. We also perform experiment using a real open source software namely OpenHospital, which is a hospital management system. The system consists of 233 classes, with 59,087 lines of code.

For each project, we manually labeled the classes with either *Boundary*, *Control*, *(Regular) Entity*, and *Data Manager*. The manual labels provide us valid classes stereotypes for the training phase and an oracle to measure the accuracy of our approach in the testing phase.

5.2 Accuracy of Stereotype Inference

We employ ten-fold cross validation to evaluate our approach. It divides all data points (i.e. classes in a project) into ten disjoint subsets of (approximately) equal size. To obtain a representative training data, the classes of the same stereotype are distributed over the subsets. Then, one subset is used as test data, while the others are used as training data. This process is repeated ten times (iterations); each iteration uses a different subset as test data.

We evaluate the accuracy of a trained model in inferring stereotypes as a ratio of number of correctly inferred class

Real Vs. Inferred Label	Number and Proportion of Predicted Classes			
	Boundary	Control	Entity	Data Man.
Boundary	81.35%	9.84%	1.04%	7.77%
Control	10.37%	58.54%	23.78%	7.32%
Entity	3.21%	2.88%	92.31%	1.60%
Data Manager	18.33%	6.11%	12.22%	63.30%

Table 3. Confusion Matrix of the Stereotypes Inferred

```
1  public class InventoryController{
2  private TrapDataManager trapDM;
3  private BaitDataManager baitDM;
4  private PlayerDataManager playerDM;
5  public InventoryController(){
6  {    trapDM = TrapDataManager.getInstance();
7       baitDM = BaitDataManager.getInstance();
8       playerDM = PlayerDataManager.getInstance(); }
9  public void setTrap(Player p, int trapID)
10 {   trapDM.setTrap(p, trapID); }
11 public ArrayList<Trap> retrieveAllTraps(String username)
12 {   return trapDM.retrieveAllTraps(username); }
13 public void setBait(Player p, int baitID)
14 {   baitDM.setBait(p, baitID); }
15 public ArrayList<Bait> retrieveAllBaits(String username)
16 {   return baitDM.retrieveAllBaits(username); }
17 public ArrayList<InventoryItem> retrieveAllInventory(String username)
18 {   return playerDM.retrieveAllPlayerInventory(username); }
19 public void readPlayerChoice(Player p, String choice)
20 {    …
21      if (tOrBChoice == 'T')
22      {    InventoryUI inventoryUI = new InventoryUI(); …}
23      else if (tOrBChoice == 'C'){
24      {          InventoryUI inventoryUI = new InventoryUI(); …}
25      else{      InventoryUI inventoryUI = new InventoryUI();
26              System.out.println("Please enter a VALID item ID > "); }}
27 public String getBaitInUse()
28 {   return baitDM.getBaitInUse(); } }
```

Figure 3. Example of a Wrongly Labeled Control Class

stereotypes with number of classes in test data. We compute the accuracy for each iteration for each project and average them as the accuracy of each project. The accuracy of our approach is then computed by taking the average of the accuracies of all projects, which is 77%.

We draw a *confusion matrix* to evaluate the accuracy of each stereotype prediction produced by the trained model [10]. A confusion matrix is a table with rows corresponding to real labels and columns corresponding to inferred labels. A cell (X,Y) in the matrix corresponds to the number of test data points with real label X that are assigned label Y by a classifier/model. Table 3 shows the accuracy of the inferred stereotypes in percentages.

Considering the diagonal entries of the matrix, we notice that boundary and entity classes can be detected with very good accuracies of more than 80%. However, it is less accurate when assigning labels to control classes. Control classes are often confused with entity classes. Upon inspection, we find that many students implement their control classes poorly. For example, consider the control class named *InventoryController* in Figure 3. It is assigned an entity stereotype by our approach. This is the case, as all of

the methods in this class except *readPlayerChoice* method perform either get data operation (e.g., *retrieveAllTraps*) or set data operation (e.g., *setTrap*). The control class simply delegates the execution of these operations to the respective data manager classes.

5.3 Accuracy of Design Flaw Detection

After the labels are inferred, we can detect design flaws as violations of the robustness and well-formedness rules. In this subsection, we show sample detected violations and analyze the quality of our violation detection mechanism.

Sample Violations. Figure 4 shows an example where violations occur in a Boundary, a Control, and an Entity.

According to the robustness rule R2, a boundary can only call controllers or actors. We detected a violation of R2 in Code-1: the boundary class named `RegistrationPage` calls an entity class named `RegistrationManager` (lines 8 and 13). Note that this is not a violation when we check it against the well-formedness rule W2.

Both robustness and well-formedness rules allow a controller to interact with any class but not actors. In Code-2, we detect a violation of the rules: the controller class named `SendingController` calls `System.out.println` (lines 6, 8, and 12) to display a message directly to a user (i.e., an actor) and uses `Scanner(System.in)` (line 16) to elicit inputs directly from the user.[2]

Robustness and well-formedness rules deal differently with entity classes. The robustness rule R3 allows an entity to call only controllers, while the well-formedness rule W3 allows an entity to call only entities. Code-3 of Figure 4 shows that an entity class named `Player` violates both of the rules: `Player` uses another entity `Inventory` (line 4) and thus violates R3; It also uses a controller `StarbugsController` (line 8) and thus violates W3. In addition, this class interacts with an actor directly via `System.out.println` (line 15), violating R3 and W3.

Quality of Detected Violations. With correct stereotype labels, our checking mechanism will detect all violations perfectly (i.e., no false positives or negatives) as both robustness and well-formedness rules are well specified. However, since stereotypes inferred by our approach could be wrong, we may detect wrong violations (false positives) or miss some violations (false negatives).

To measure false positives and negatives, we can simply compare the violations detected with inferred labels against those detected with correct labels: The size of the intersection of the two sets relative to the sizes of the two sets are indicative of false positive rates and negative rates, which can be measured by the notion of precision and recall. Precision is the ratio of inferred violations that are true and recall is the ratio of true violations that are inferred.

[2] We have (manually) predefined a list of classes and functions that send or receive messages to users which are treated as actors'.

	Class & Type		Class Only	
Evaluation	**Rob.**	**Well.**	**Rob.**	**Well.**
Precision	61.2%	74.6%	68.6%	69.9%
Recall	68.7%	61.8%	78.8%	59.8%

Table 4. Precision and Recall of Detected Anomalies for Robustness Analysis (Rob.) and Well-Formedness Analysis (Well.)

$$Precision = \frac{\|\{Inferred\,Violations\} \cap \{True\,Violations\}\|}{\|\{Inferred\,Violations\}\|}.$$

$$Recall = \frac{\|\{Inferred\,Violations\} \cap \{True\,Violations\}\|}{\|\{True\,Violations\}\|}.$$

When comparing violations during the intersection operation, we consider two equivalence criteria. One criterion considers two violations are matched only if both the violating class (i.e., the class where the violation occurs) and the type of the violation are matched (Class & Type). The other considers only the violating class (Class Only).

The total numbers of true violations of robustness and well-formedness rules are 244 and 138 respectively. The total numbers of inferred robustness and well-formedness violations are 255 and 112 respectively. The overall precision and recall of our approach is shown in Table 4. The precision and recall values are aggregated averages across many model building and testing iterations. We calculate them using the two equivalence criteria.

Table 4 shows that violating classes (Class Only) can be detected with precision and recall of 68.7% and 78.8% (robustness), and 69.9% and 59.8% (well-formedness). When considering both violating classes and violation types (Class & Type), the precision and recall are reduced by 7.5% and 10.1% (robustness), and increased by 4.7% and 2% (well-formedness), due to some violations are detected with correct violating classes but wrong violation types.

6 Discussion

Effects of Features Used. We used eight code features in our experiments. It would be interesting to consider other features, such as code complexity metrics, which might help to improve the accuracy of the stereotype inference further. The confusion matrix shown in Section 5.2 particularly suggests more features related to controllers should be used to reduce the number of confusion occurrences.

Effects of Dataset Investigated. We perform experiments on software systems written by novice programmers and one real medium-sized software system. These systems are chosen based on the availability of the class labels. None of the processes in our framework is expensive: basic information extraction, feature construction, model learning, and stereotype assignment make use of an inexpensive static analysis technique and a scalable classification engine, i.e., SVM. We believe the framework is able to process larger programs. We plan to analyze larger systems in the future.

Effects of Design Rules Checked. We detected violations against only robustness and well-formedness rules. There

Code - 1

```
1  public class RegistrationPage{
2    ...
3    private RegistrationController RC=new
                                RegistrationController();
4    private RegistrationManager RM=RC.getRM();

5    public RegistrationPage()
6    {   Scanner sc= new Scanner(System.in); ....
7      username= sc.next();
8      if(RM.usernameIsAvailable(username)) {
9        ...
10       String password =
11       String confirmation
12       if(confirmation.equals(password)) { ....
13           RM.processRegistration(username,password);
14         }else { ... }
15       }else{
16     System.out.println(username+" is already in use");
17 }}}
```

Code - 2

```
1  public class SendingController
2  {
3    public void displayPlayer()
4    {   ArrayList<Player> playerList =
                PlayerDataManager.retrievePlayers(); ...
5      if(playerList.size() < 2)  {
6        System.out.println("There is no other
7      } else {
8        System.out.println("Which user ...");
9        for (int i = 0; i < playerList.size(); i++)
10       {  Player aPlayer = playerList.get(i);
11         if(
12         { System.out.println(... aPlayer.getName());
13           playerID++;  } }}}
14   public void displayGiftOption(Player recipient)
15   {  ...
16     Scanner sc = new Scanner(System.in); ...
17   }
```

Code - 3

```
1  public class Player {
2  ...
3  public Player(String username, String password, String rank, String
     region, int experience, int gold)
4  {  ...  inventory = new Inventory();  }

5  public int getGold()
6  {  try
7    {  Player p = this;
8      StarbugsController starbugsController = new
9      ArrayList<BaitLineItem> alBli = new ArrayList<BaitLineItem>();
10     alBli = p.getInventory().getBaits();
11     BaitLineItem currentBait = null;
12     currentBait = p.getInventory().getBait();
13     if (alBli.isEmpty() && !(currentBait != null)) {
14       if (this.gold < 50) {
15         ... System.out.println("Your gold  ...");
16       } } catch (Exception e) {}
17   return this.gold; }
```

Figure 4. Violations in Boundary, Controller, and Entity Stereotypes

are other design rules, for example, some design rules relax the robustness rules by allowing direct interactions between Entity objects. Our approach can be easily extended to handle such variants. However, if the design rules involve constraints such as conditional call-relations, our violation checking mechanism would then need further improvements, such as, taking control-flow conditions embedded in inter-procedural call graphs into consideration.

Threats to Validity. To reduce the threats to construct validity, we used standard evaluation metrics, namely accuracy, precision, and recall, which are commonly used in data mining and information retrieval tasks. However, it remains a question what is the effect of inaccuracies on program comprehension. To answer this question, a user study would be needed and is left as future work. To reduce the threats to internal validity and selection bias, the 15 projects used in the experiment are chosen randomly from a pool of 93 student projects. However, as aforementioned, there are still threats to external validity on the generalizability of our results. We plan to evaluate our framework on various types of software systems of various sizes to alleviate the threats.

7 Conclusion & Future Work

In this paper, we present a framework that detects likely design flaws in layered object-oriented architecture with classes belonging to various stereotypes, which should follow certain design rules. Also, to accommodate to systems without sufficient stereotype annotations, our framework learns a statistical model to distinguish various class stereotypes available from a training set. This model in turn is used to give labels to unannotated classes. Likely design flaws are later detected by finding violations of well-known design rules. We have evaluated our approach on Java projects developed by novice and expert developers. The results show that our approach can identify class stereotypes with 77% accuracy on average and can detect violations of the design rules associated with each stereotype with up to 75% accuracy and up to 79% recall.

In the future, we plan to further investigate more useful features for the inference of class stereotypes and more software systems. We believe our framework is general and can be adapted for reverse engineering other kinds of domain-specific stereotypes.

Acknowledgement. We would like to thank Yeow-Leong Lee for providing some stereotype labels.

References

[1] O. Andriyevska, N. Dragan, B. Simoes, and J. Maletic. Evaluating uml class diagram layout based on architectural importance. In *VISSOFT*, 2005.

[2] C. Atkinson, T. Kuhne, and B. Henderson-Sellers. Systematic stereotype usage. *Software and Systems Modelling*, 2:153–163, 2003.

[3] G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley, 1999.

[4] K. Crammer and Y. Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *JMLR*, 2002.

[5] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines: And Other Kernel-Based Learning Methods*. Cambridge, 2000.

[6] N. Dragan, M. Collard, and J. Maletic. Reverse engineering method stereotypes. In *ICSM*, 2006.

[7] N. Dragan, M. Collard, and J. Maletic. Automatic identification of class stereotypes. In *ICSM*, 2010.

[8] M. Gogolla and B. Henderson-Sellers. Analysis of uml stereotypes in the uml metamodel. In *UML*, 2002.

[9] Y.-G. Guéhéneuc and H. Albin-Amiot. Using design patterns and constraints to automate the detection and correction of inter-class design defects. In *TOOLS USA*, 2001.

[10] J. Han and K. Micheline. *Data Mining Concepts and Techniques*. Morgan Kaufmann, 2006.

[11] F. Khomh, S. Vaucher, Y.-G. Guéhéneuc, and H. Sahraoui. A bayesian approach for the detection of code and design smells. In *QSIC*, 2009.

[12] L. Kuzniarz, M. Staron, and C. Wohlin. An empirical study on using stereotypes to improve understanding of UML models. In *IWPC*, 2004.

[13] N. Moha, Y.-G. Guéhéneuc, L. Duchien, and A.-F. L. Meur. DECOR: A method for the specification and detection of code and design smells. *IEEE TSE*, 36:20–36, 2010.

[14] Rational Software et al. *UML Extension for Objectory Process for Software Engineering ver. 1.1*, 1997.

[15] D. Rosenberg and K. Scott. *Use case driven object modeling with UML: a practical approach*. Addison-Wesley, 1999.

[16] `http://svmlight.joachims.org/svm_multiclass.html`.

[17] P. Tarr, H. Ossher, W. Harrison, and S. S. Jr. N degrees of separation: Multi-dimensional separation of concerns. In *ICSE*, 1999.

[18] A. Trifu and U. Reupke. Towards automated restructuring of object oriented systems. In *CSMR*, 2007.

[19] S. Vaucher, F. Khomh, N. Moha, and Y.-G. Guéhéneuc. Tracking design smells: Lessons from a study of god classes. In *WCRE*, 2009.

[20] `http://wala.sourceforge.net`.

Exploiting Computational Redundancy for Efficient Recovery from Soft Errors in Sensor Nodes

Aly Farahat
Department of Computer Science
Michigan Technological University
Houghton MI 49931, U.S.A.
Email: anfaraha@mtu.edu

Ali Ebnenasir
Department of Computer Science
Michigan Technological University
Houghton MI 49931, U.S.A.
Email: aebnenas@mtu.edu

***Abstract*—Most existing techniques for the design and implementation of fault tolerance use resource redundancy. As such, due to scarcity of resources, it is difficult to directly apply them for adding fault tolerance to sensor nodes in Wireless Sensor Networks (WSNs). Thus, it is desirable to develop techniques that implement fault tolerance under the constraints of memory and processing power of sensor nodes. We present a novel method for designing recovery from transient faults that cause non-deterministic bit-flips in the task queue of the scheduler of TinyOS, which is the operating system of choice for sensor nodes. Specifically, our approach exploits computational redundancy for the design of recovery instead of using resource redundancy. The presented fault-tolerant task queue recovers from bit-flips with significantly lower space/time overhead compared with the Error Correction Codes.**

I. Introduction

Wireless Sensor Networks (WSNs) are increasingly used in mission-critical applications (e.g., body sensor networks, habitat monitoring, flood forecasting, etc.), where they have to be deployed in harsh environments (e.g., volcano, forest, battle field, etc.). On one hand, WSNs must exhibit a high degree of service dependability due to application requirements, and on the other hand, unexpected environmental events, i.e., *faults*, may negatively affect their quality of service. For example, *transient faults* may cause non-deterministic bit-flips in the main memory of sensor nodes (a.k.a. *motes*), thereby perturbing the state of the running program to an arbitrary state in its state space. Since the quality of the service provided by the entire sensor network heavily relies on the dependability of the controlling software of motes, fault tolerance techniques should be applied to improve the dependability of motes. Nonetheless, due to their limited computational (e.g., memory and processing power) and energy resources, it is impractical to apply the traditional fault tolerance methods (e.g., Error Correction Code (ECC) [9]) to motes. This paper proposes a novel method that exploits computational redundancy for the addition of recovery to transient bit-flips in the task queue of TinyOS [14].[1]

Most existing techniques [12], [4], [10], [1], [13] present solutions for the design of fault-tolerant protocols for WSNs rather than focusing on the fault tolerance of individual sensor nodes. For instance, techniques for reliable transmission are mostly based on redundant and/or multi-path retransmission [10]. Several methods exist for (i) designing self-stabilizing WSN communication protocols [1] that ensure a correct synchronization among sensor nodes starting from an arbitrary non-synchronized state, and (ii) providing recovery for data dissemination in WSNs [13]. ECC methods (e.g., Hamming code [9]) often require extensive memory redundancy for storing extra parity bits in code words. Moreover, decoding/coding algorithms in these methods are computationally expensive.

We propose a novel approach that enables space/time-efficient recovery to transient Bit-Flips (BFs) in motes. The proposed approach is based on the detection of the violations of invariance conditions that must always be true and dynamic corrections of such violations. Specifically, we focus on the task queue of the TinyOS as it is one of the most critical components of the kernel of TinyOS and its structure is heavily sensitive to BFs. We first define conditions under which the task queue has a valid structure, called the *structural invariant*. Then, before and after the addition/removal of a task to/from the task queue, we check whether the structural invariant holds. In case of the violation of the invariant, we identify different failure scenarios created due to the occurrence of BFs and systematically correct them, thereby recovering to the structural invariant. The proposed approach enables the detection and correction of multiple BFs in a single-byte variable in a space/time-efficient fashion. Compared with the Hamming Code (HC) [9], our approach needs at least 20% less memory and performs at least twice as fast as HC. The time complexity of our approach is linear in the size of the task queue. We also note that HC cannot correct multiple BFs whereas our approach enables the correction of multiple BFs as long as they occur in the same variable. Furthermore, for some special cases, the proposed approach corrects BFs in multiple variables as well.

Organization. Section II illustrates the structure of the TinyOS task queue. Then, Section III presents our approach for the detection and correction of transient bit flips in the TinyOS task queue. We also demonstrate the superiority of the space/time efficiency of the proposed approach compared with the ECC methods. Section IV makes concluding remarks and outlines future research directions.

[1] TinyOS is the operating system of choice for sensor nodes in WSNs.

II. Structural Invariance of TinyOS Task Queue

In this section, we define what constitutes a valid structure of TinyOS's task queue. In Tiny OS version 2.x, the task queue is a linked list of task identifiers implemented as a statically allocated array of 256 entries (see Figure 1). Figure 2 illustrates the implementation of the task queue in nesC [8], which is a component-based variant of the C programming language used for application development on TinyOS. Each identifier (ID) is an integer between 0 and 255 inclusive. The set of variables of interest are `m_head`, `m_tail` and `m_next[256]`. `m_head` holds the index of the oldest ID in the queue, and `m_tail` holds the ID of the most recent task inserted in the queue. Every value in `m_next` is an ID for the next task to be executed and an index (i.e., pointer) to the successor entry in `m_next`. A distinguished task has the identifier `NO_TASK` $= 255$. `NO_TASK` is the value of `m_next[m_tail]` and it is the successor of all non requesting identifiers. For example, as depicted in Figure 1, a queue state s_1 consists of `m_head`=12, `m_next[12]`=3, `m_next[3]`=255, `m_tail`=3, and $\forall j : (j \neq 12) :$ `m_next[j]`=255.

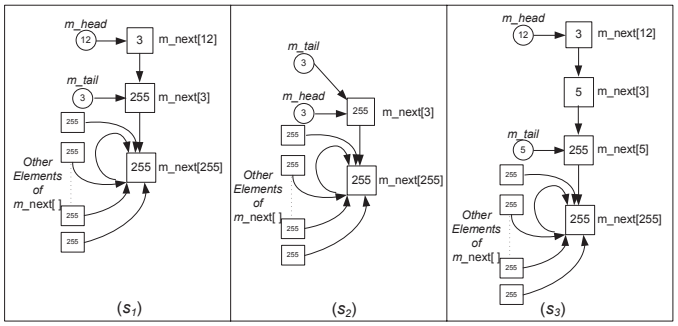


Fig. 1. Example states of the task queue.

The state s_1 represents a task queue having only two pending tasks of identifiers 12 and 3 respectively. The effect of `popTask()` on s_1 is a transition to state s_2 (see Figures 2 and 1). In state s_2, `m_head`=3, `m_next[3]`=255, `m_tail`=3, and $\forall j : (0 \leq j \leq 255) :$ `m_next[j]`=255. The effect of `pushTask(5)` on s_1 is a transition to state s_3 (see Figures 2 and 1), where `m_head`=12, `m_next[12]`=3, `m_next[3]`=5, `m_next[5]`=255, `m_tail`=5, and $\forall j : (j \neq 12) \wedge (j \neq 3) \wedge (0 \leq j \leq 255)$: `m_next[j]`=255.

Structural Invariant. A *valid* state of the task queue is a state where the queue has a linear structure with its head (`m_head`) pointing to its beginning and its tail (`m_tail`) pointing to the most recently added identifier to the task queue. Each element of `m_next` with a non-255 ID is reachable from the head. Each entry of `m_next` that is not in the queue holds the value of `NO_TASK`, and `m_next[m_tail]` is equal to `NO_TASK`. Moreover, the task IDs belong to the interval $0 \leq ID \leq 255$. Figure 3-(a) illustrates a sample valid state of the task queue. Furthermore, any operation performed on the queue should remove an element from the head (i.e., `popTask()`), add an element to the tail `pushTask()` or leave the structure of the queue and the task IDs unchanged.

```
inline uint8_t popTask()
{
 if( m_head != NO_TASK ) {
   uint8_t id = m_head;
   m_head = m_next[m_head];
   if( m_head == NO_TASK )   m_tail = NO_TASK;
   m_next[id] = NO_TASK;
   return id;   }
  else       return NO_TASK;
  }

bool isWaiting( uint8_t id )
 { return (m_next[id] != NO_TASK) || (m_tail == id); }

bool pushTask( uint8_t id )  {
 if( !isWaiting(id) )  {
   if( m_head == NO_TASK ) { m_head = id;  m_tail = id; }
   else  {  m_next[m_tail] = id;  m_tail = id;   }
   return TRUE;
 } else   return FALSE;      }
```

Fig. 2. Excerpt of the Tiny OS Scheduler.

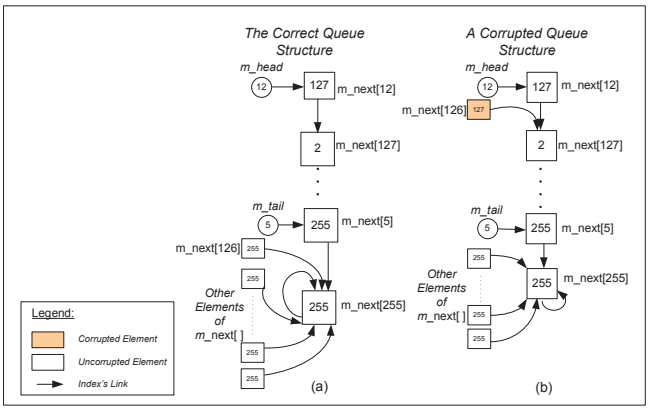


Fig. 3. Valid and invalid task queue structures.

Transient faults. Transient faults may toggle multiple bits in a single variable; i.e., `m_head`, `m_tail` or a memory cell of `m_next[]`. The case of multi-variable corruption is the subject of our current investigation. Bit-flips may perturb a task ID and the structure of the task queue to an invalid state. For example, Figure 3 demonstrates how resetting the most significant bit of `m_next[126]` could change its content from 255 to 127, thereby pointing to `m_next[127]` instead of pointing to `NO_TASK`.

III. Addition of Recovery

Section III-A analyzes the memory and time requirements of correcting BFs with the Hamming code. Section III-B illustrates how our approach enables recovery from BFs by detecting invalid queue structures and correcting them.

A. Correcting Bit-Flips with ECC

One approach for recovery from transient faults that cause bit-flips is to use error detection and correction codes such as the Hamming Code (HC) [9]. However, due to high memory/CPU cost of the encoding/decoding algorithms these approaches seem impractical in the context of WSNs. For example, there are two ways to deal with bit-flips in the task queue using HC; consider either individual memory cells of the `m_next[]` array as separate data words, or the entire 256 bytes of the task queue as one data word.

In the first case, each cell of the `m_next[]` array should be encoded before storing a value and it should be decoded before reading its contents. To encode 8 bits of data with HC, we need 4 extra parity bits, which results in a code word with 12 bits in the following format: $p_1p_2d_1p_3d_2d_3d_4p_4d_5d_6d_7d_8$, where d_j denotes data bits for $1 \leq j \leq 8$, and p_i represents the parity bits for $1 \leq i \leq 4$. The encoding algorithm of HC determines the 12-bit code word by multiplying a 12×8 matrix by a vector made of the data bits. Such a matrix multiplication takes 96 multiplications and 84 additions; i.e., totally 180 *basic operations* in addition to one read and write operation on each memory cell, where a basic operation includes arithmetic and logical operations as well as comparisons and load/store. The decoding algorithm also multiplies a 4×12 matrix by a vector containing the 12-bit code word, which results in a 4-bit syndrome vector representing the position of the corrupted bit. (Thus, each decoding takes 48 multiplications and 44 additions, totally 92 basic operations.) Notice that for each byte allocated in `m_next[]` 4 extra bits should be considered for parity. That is, $256/2 = 128$ extra bytes should be allotted along with the 256 bytes allocated for `m_next[]`. Besides, every time a task ID is stored/retrieved to/from a memory cell in `m_next[]`, the encoding/decoding algorithm must be executed. That is, for one round of detection and correction, $256 \times (180 + 92) = 69632$ basic operations should be performed.

In the second case, the queue comprises a bit pattern with $256 \times 8 = 2024$ bits, for which $1 + log\ 2024 = 12$ parity bits are needed in HC. Thus, the size of the code word is equal to $2024 + 12 = 2036$ bits. The encoding takes $2024 \times 2036 = 4120864$ multiplications and $2023 \times 2036 = 4118828$ additions; i.e., totally 8239692 basic operations. For decoding, we will need 12×2036 multiplications and 12×2035 additions resulting in 48852 basic operations. This analysis clearly illustrates the impracticality of using HC on sensor nodes with a small memory and a limited processing power. While other ECC methods (e.g., , Forward Error Correction (FEC) [3] and Reed-Solomon (RS) [15]) can correct cases where multiple variables are corrupted, they are even more expensive than HC in terms of either space or time. For example, the RS method needs t bits for the correction of $\lfloor t/2 \rfloor$ bits and $\mathcal{O}(t^2)$ is its time complexity.

B. Adding Recovery to Task Queue

This section illustrates how we enable recovery to a valid queue structure from transient BFs. Figure 4 depicts a state machine that demonstrates the impact of transient faults and how recovery should be achieved. Since the task queue is a centralized program running on a single CPU, we can benefit from a high atomicity model in which a set of instructions can be performed atomically. In fact, the nesC language provides atomic blocks that capture a sequence of statements that are supposed to be executed without interruption. The essence of the addition of recovery in high atomicity [6] is based on detecting the violation of the invariant due to the occurrence of faults, and providing recovery from every invalid state to the invariant. Thus, we present the function `DetectCorrect()` (see Figure 5) that we add to the Tiny OS scheduler to enable the detection and correction of BFs before and after any push/pop operations on the task queue. The function `DetectCorrect()` should be invoked in an atomic block (i.e., `atomic{DetectCorrect()}`) to ensure that detection and correction are not interrupted during execution. Depending on the harshness of the environment where the motes are deployed, the period of invoking `DetectCorrect()` could be changed by the developers; i.e., `DetectCorrect()` can be invoked in an adaptive fashion by the scheduler of TinyOS in order to enable a tradeoff between the degree of dependability and the energy cost of providing recovery. Next, we explain different parts of `DetectCorrect()` to illustrate how detection and correction are achieved.

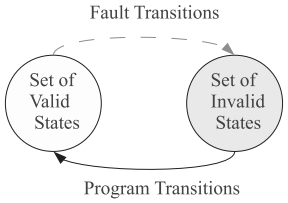


Fig. 4. Adding recovery to the task queue.

Data structures. To detect and correct corruptions of the task queue, `DetectCorrect()` gathers some information about the structure of the queue and stores them in these data structures (see Figure 5):

```
DetectCorrect(int q_size) {
 uint8_t previous;
 uint8_t current=m_head;
 uint8_t Index=0;
 uint8_t dangleElem=0;
 uint8_t non255 =0;       // Number of non-255 elements
 uint8_t qLength =0;      // Number of elements in the queue
 uint8_t cyclePoint =0;   // The corrupted element that
                          // points back and creates a cycle

 component visited = new BitVectorC(256);
 visited.clearAll();
 component pointedTo = new BitVectorC(256);
 pointedTo.clearAll();
 component pointsToNoTask = new BitVectorC(256);
 pointsToNoTask.clearAll();

 bool cyclic = FALSE;
 bool pointedByHead = FALSE;
```

Fig. 5. Data structures.

(1) `previous, current, dangleElem` and `cyclePoint` are pointers that are used during the traversal of the queue; (2) `non255` keeps the number of memory cells in `m_next` that contain non-255 values; (3) `qLength` stores the number of non-255 elements reachable from the head of the queue (`m_head`); (4) the `visited` bit vector allocates one bit corresponding to each element of `m_next` illustrating whether or not it has been visited previously in a queue traversal for cycle detection; (5) the `pointedTo` bit vector keeps a bit for each element of `m_next` demonstrating whether or not that element is being pointed to by some other element; (6) the `pointsToNoTask` bit vector allocates a bit corresponding to each memory cell of `m_next` that

contains NO_TASK; (7) the cyclic flag is set if a cycle is detected in the structure of the task queue, and (8) the pointedByHead flag is true if and only if there is an element in the queue that is pointed by both m_head and another element in the queue. We use the pointedByHead flag in detecting/correcting the corruption of m_head. Notice that, we allocate 96 bytes for the bit vectors and 7 bytes for other variables (i.e., 103 bytes totally) capturing local variables; i.e., when DetectCorrect() returns this memory is released.

Initialization. In this step, we first count the total number of elements in m_next that contain non-255 values. Then, we initialize the pointedTo and pointsToNoTask bit vectors. This step incurs $256 \times 15 = 3840$ basic operations on our solution.

```
// Count the number of non-255 elements in array m_next
 for(Index=0; Index<NO_TASK; ++Index)
    if (m_next[Index] != NO_TASK) non255++;
// Determine the elements that are being pointed to
 for(Index=0; Index<NO_TASK; ++Index)
    pointedTo.set(m_next[Index]);
// Determine the elements that point to NO_TASK
 for(Index=0; Index<NO_TASK; ++Index)
    if(m_next[Index] == NO_TASK) pointsToNoTask.set(Index);
```

Detection and correction of m_head. Since the traversal of the queue for subsequent processing is performed using the m_head pointer, we first ensure that m_head is corrected. If m_head points to NO_TASK (see Figure 6), then we set m_head to the index of the element to which no other element points, and exit (because, by assumption, our focus is on single-variable corruption). Otherwise, we detect whether m_head points to another non-255 element in the queue. (Please see Figure 7 and the first for-loop in the else part of Figure 6.) If so, then we set m_head to the index of the non-255 element to which no other element points. (See the second for-loop in the else part of Figure 6.) Figure 7 illustrates a case where the value of m_head has been corrupted from 12 to 4.

```
if (m_head == NO_TASK)  {
  for(Index=0; Index<NO_TASK; ++Index)
    if(!pointedTo.get(Index) && m_next[Index] != NO_TASK) {
            m_head = Index;  return; }      }
else {
  for(Index=0; Index<NO_TASK; ++Index)
    if(pointedTo.get(Index) && Index == m_head) {
                    pointedByHead = TRUE; break; }
  if (pointedByHead)
   for(Index=0; Index<NO_TASK; ++Index)
    if(!pointedTo.get(Index) && m_next[Index] != NO_TASK) {
                  m_head = Index;  return; }
      }
```

Fig. 6. Detect and correct m_head.

The time complexity (and energy consumption) of this step is proportional to the maximum number of basic operations. If m_head = NO_TASK, the for-loop in the if part of Figure 6 will be executed, which has one comparison and one increment for the loop counter in each iteration. Moreover, the if-statement inside the for-loop performs two load operations, one comparison and two logical operations per iteration. Thus, in the worst case, we have 7 basic operations in each iteration of this for-loop, which results in $256 \times 7 = 1792$ basic operations if m_head = NO_TASK. A similar reasoning illustrates that, in the worst case, we perform $256 \times 15 = 3840$ basic operations if m_head $\neq$ NO_TASK. Therefore, since either the if part or the else part is executed in Figure 6, the correction of m_head takes at most 3840 basic operations.

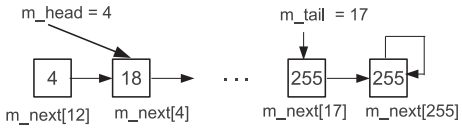


Fig. 7. Corruption of m_head.

Detection and correction of cyclic structures. The do-while loop in the below code uses the visited bit pattern to determine whether there is a cycle in the queue. This loop also stores the number of elements in the queue that are reachable from m_head in the qLength variable.

```
// Detect cycles
do {
   if(!visited.get(current))       visited.set(current);
   else { cyclic = TRUE;
          cyclePoint = previous;     break;  }
   previous=current;
   current=m_next[previous];
   qLength++;
     } while(current != NO_TASK && m_tail!=previous);

// Correct cycles
if (cyclic && (cyclePoint == m_tail)) {
          m_next[cyclePoint] =  NO_TASK; return; }
if (cyclic && cyclePoint != m_tail)
  for(Index=0; Index<NO_TASK; ++Index)
    if(!pointedTo.get(Index) &&
          (m_next[Index] != NO_TASK) && (Index != m_head)) {
          m_next[cyclePoint] = Index;  return; }
```

A cycle could be formed in two ways: either the tail points back to some element including itself (see Figure 8-(a)), or another element points back to some element including itself (see Figure 8-(b)). If a cycle is detected, then the cyclic flag is set and the index of the element pointing back is stored in cyclePoint. In case cyclePoint is equal to m_tail, then that means the tail of the queue is pointing back to some element instead of pointing to NO_TASK. Otherwise, to fix the cycle, we set the contents of m_next[cyclePoint] to the index of the element that has become *dangled* due to the cycle creation; i.e., the element to which no element points, does not point to NO_TASK, and is not equal to m_head. This correction will take at most $256 \times 26 = 6656$ basic operations.

Detection and correction of queue size and non-255 elements. To detect discrepancies in the size of the task queue, we add a new variable q_size to the TinyOS scheduler to store the size of the queue outside the DetectCorrect function. Nonetheless, q_size could be perturbed by transient faults. The first if statement in Figure 9 corrects q_size. Notice that DetectCorrect can simultaneously correct q_size and m_head, which is a special case of correcting MBFs in multiple variables. Moreover, faults may change the value of an array element from 255 to some other value. This means that that element points to some queue element. Such a link is not part of the task queue and should be eliminated.

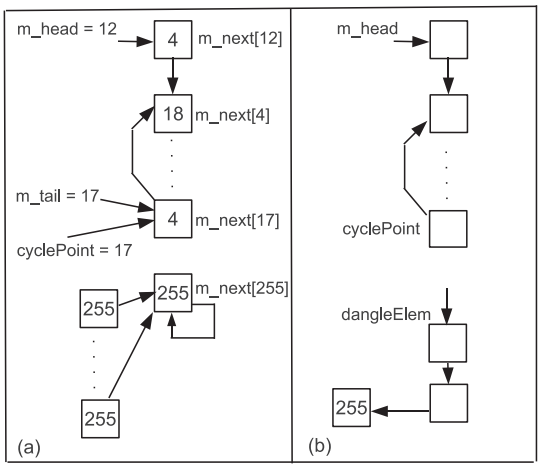


Fig. 8. Cyclic corruption of the task queue.

To this end, we assign 255 to an element to which no other element points, points to a non-255 element and is not equal to `m_head`. The `for-loop` in the second `if statement` could take at most $256 \times 11 = 2816$ basic operations.

```
if (qLength == non255) {
    if (q_size != qLength)  { q_size = qLength; return; }
        else return;  // Task queue is NOT corrupted.
         }

if (non255 > q_size) // some 255 element has become non255
   for(Index=0; Index<NO_TASK; ++Index)
     if (!pointedTo.get(Index) &&
        (m_next[Index] != NO_TASK) && (Index != m_head)) {
                m_next[Index] = NO_TASK;  return;  }
```

Fig. 9. Detect and correct queue size.

Detection and correction of `m_tail`. To detect and correct the corruptions of `m_tail`, we set `m_tail` to the index of the first element whose contents point to `NO_TASK` (see below). For example, in Figure 10, `m_tail` is set to 17. The `for-loop` in the below code performs at most 2560 basic operations.

```
for(Index=0; Index<NO_TASK; ++Index) {
    if ((!pointsToNoTask.get(Index)) &&
        (m_next[m_next[Index]] == NO_TASK) &&
                (m_tail != m_next[Index])) {
               m_tail = m_next[Index];  return;  } }
```

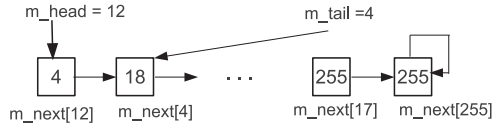


Fig. 10. Corruption of `m_tail`.

Detection and correction of corrupted acyclic structures. If faults corrupt a non-255 element so it points to one of its successors, then a structure similar to Figure 11 could be created. In this example, the contents of `m_next[4]` is changed from 18 to 25 and `m_next[18]` becomes unreachable from head; i.e., a *dangling* element. One way to detect this case is to simply compare `qLength` with the number of non-255 elements; if `qLength` $\neq$ `non255`, then either this case has occurred or the corruption of `m_head`. Nonetheless, if the code of the `DetectCorrect()` routine reaches this point, then it means that `m_head` has the correct value.

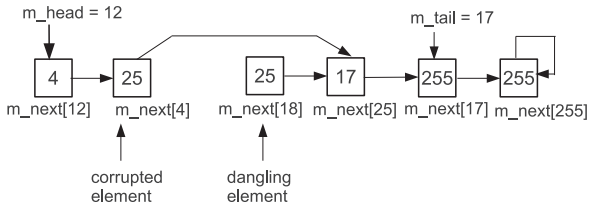


Fig. 11. Corrupted acyclic structure.

The identification of the `corrupted element` in Figure 11 is not straightforward. Our strategy is to determine the index of the element in the queue that is pointed by two internal elements of the queue (see `m_next[25]` in Figure 11). Such an element must be in the fragment of the queue that starts with the dangling element. Thus, we first find the index of the dangling element by the first `for-loop` in Figure 12. If there is such a dangling element, then we reset the `pointedTo` bit vector. Then, in the first `do-while` in Figure 12, we start setting the bits of `pointedTo` corresponding to the fragment of the queue that starts with the dangling element. In the second `do-while`, we search the first fragment of the queue (starting from `m_head`) for the element that points to an element whose corresponding bit is already set in the `pointedTo` vector. Once we find such an element, we set its content to the index of the dangling element, and the queue is corrected. This step includes $27 \times 256 = 6912$ basic operations in the worst case.

```
dangleElem = NO_TASK;
for(Index=0; Index<NO_TASK; ++Index)
   if(!pointedTo.get(Index) &&
       m_next[Index] != NO_TASK &&
       Index != m_head) {
                  dangleElem = Index; break; }

if (dangleElem == NO_TASK)  return;

pointedTo.clearAll();
current = dangleElem;

do {
     pointedTo.set(m_next[current]);
     previous=current;
     current=m_next[previous];
     } while(current != NO_TASK && m_tail!=previous);

current = m_head;

do {
    if (pointedTo.get(m_next[current]) {
              m_next[current] = dangleElem; return; }
    previous=current;
    current=m_next[previous];
     } while(current != NO_TASK && m_tail!=previous);
   }
```

Fig. 12. Detect and correct acyclic structures.

Time complexity of `DetectCorrect()`. Since the code of `DetectCorrect()` does not include nested for-loops, its time complexity is linear in the size of the task queue. Figure 13 presents a comparison of the time/space cost of the proposed method of this paper with two scenarios of using

the Hamming code for correction of BFs: HC1 represents the case where each element of `m_next` is encoded with HC, and HC2 denotes the case where the entire `m_next` is encoded as a single word. Notice that, our approach outperforms HC1 in terms of both time and space efficiency, respectively by a factor of 20% and 60%. More importantly, the required memory (i.e., 103 bytes) is temporary; i.e., when `DetectCorrect()` returns this memory is released. The HC2 method seems impractical due to expensive computing requirements.

Approach	Memory Cost	# of Operations
Hamming Code for each element of `m_next` (HC1)	128 Bytes	$\simeq$ 70000
Hamming Code for the entire `m_next` (HC2)	12 bits	$\simeq$ 4.17 million
Proposed Method	103 Bytes	28000

Fig. 13. Space/Time cost of correction of BFs.

Scope of correction. The scope of correction in these three methods is different. Figure 14 demonstrates that our approach can correct multiple bit-flips in a single variable, which cannot be achieved by HC1 and HC2. However, HC1 can correct single bit-flips in multiple variables, which we do not currently have a solution for it.

Approach	Corrects SBFs	Corrects MBFs in a Variable	Corrects SBFs in Multi Vars	Corrects MBFs in Multi Vars
HC1	Yes	No	Yes	No
HC2	Yes	No	No	No
Proposed Method	Yes	Yes	No	No

Fig. 14. Scope of correction for Single Bit-Flips (SBFs) and Multiple Bit-Flips (MBFs).

Fault tolerance of `DetectCorrect()`. In case transient faults perturb the local variables and/or the control flow of `DetectCorrect()`, the current round of execution of `DetectCorrect()` may not recover the structure of the task queue. However, since `DetectCorrect()` is executed repeatedly and transient faults eventually stop occurring, `DetectCorrect()` will eventually provide recovery.

IV. Conclusions and Future Work

We presented a novel method for the detection and correction of transient Bit-Flip (BF) in the task queue of the TinyOS, which is the operating system of choice for sensor nodes. Since motes have limited computational and energy resources, instead of using resource redundancy, the proposed approach exploits computational redundancy to efficiently recover from transient BFs that corrupt the contents and the structure of the task queue. The essence of our approach is based on the detection of invalid structures of the queue that might be created due to transient faults. Upon reaching an invalid structure, we analyze the structure of the task queue to determine which failure scenario has occurred and recover to a valid state. Using this method, we can correct Multiple BFs (MBFs) in single-byte variables. We illustrate that the proposed approach can provide a better time/space efficiency with respect to Error Correction Codes such as the Hamming code [9] (see Figures 13 and 14).

Several techniques exist for designing fault-tolerant data structures. Aumann *et al.* [2] present alternative implementations for pointer-based data structures by adding redundant links. Finocchi *et al.* [7] provide resilient search trees through periodic checkpoints. Jørgensen *et al.* [11] devise a method that ensures the resilience of priority queues by storing pointers in resilient memory locations. By contrast, our approach continuously monitors a structural invariance and provides recovery if the invariant is violated.

Future/ongoing work focuses on techniques for the correction of MBFs in multiple variables. Moreover, we would like to leverage our previous work [5] on automated addition of fault tolerance for the addition of recovery to data structures. Specifically, our previous work models a state as a unique valuation of variables of primitive types (e.g., Boolean and integer). Nonetheless, we need to create richer models that capture the topological state of complex data structures. We will also work on models where ECC methods and our approach are used in a hybrid fashion.

References

[1] M. Arumugam and S. Kulkarni. Self-stabilizing deterministic TDMA for sensor networks. *Distributed Computing and Internet Technology*, pages 69–81, 2005.

[2] Y. Aumann and M. Bender. Fault tolerant data structures. In *focs*, page 580. Published by the IEEE Computer Society, 1996.

[3] G. C. Clark and J. B. Cain. *Error-Correction Coding for Digital Communications*. Springer, 1981.

[4] T. Clouqueur, P. Ramanathan, K. Saluja, and K. Wang. Value-fusion versus decision-fusion for fault-tolerance in collaborative target detection in sensor networks. In *Proceedings of Fourth International Conference on Information Fusion*. Citeseer, 2001.

[5] A. Ebnenasir. *Automatic synthesis of fault-tolerance*. PhD thesis, Michigan State University, 2005.

[6] A. Ebnenasir and A. Farahat. A lightweight method for automated design of convergence. In *To appear in the proceedigns of 25th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2011.

[7] I. Finocchi, F. Grandoni, and G. Italiano. Resilient search trees. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 547–553. Society for Industrial and Applied Mathematics, 2007.

[8] D. Gay, P. Levis, R. Von Behren, M. Welsh, E. Brewer, and D. Culler. The nesC language: A holistic approach to networked embedded systems. In *Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation*, page 11. ACM, 2003.

[9] R. W. Hamming. *Coding and Information Theory*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1986.

[10] W. Heinzelman, J. Kulik, and H. Balakrishnan. Adaptive protocols for information dissemination in wireless sensor networks. In *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, pages 174–185. ACM, 1999.

[11] A. Jørgensen, G. Moruz, and T. Mølhave. Priority queues resilient to memory faults. *Algorithms and Data Structures*, pages 127–138, 2007.

[12] F. Koushanfar, M. Potkonjak, and A. Sangiovanni-Vincentelli. Fault tolerance techniques for wireless ad hoc sensor networks. *sensors*, pages 1491–1496, 2002.

[13] S. Kulkarni and M. Arumugam. Infuse: A TDMA based data dissemination protocol for sensor networks. *International Journal of Distributed Sensor Networks*, 2(1):55–78, 2006.

[14] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, et al. TinyOS: An operating system for sensor networks. *Ambient Intelligence*, pages 115–148, 2005.

[15] I. S. Reed and G. Solomon. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, 8:300–304, 1960.

A Web Service Reliability Model Based on Birth-Death Process

Chunli Xie[1,2,3], Bixin Li[1,2], Xifeng Wang[1,2,4]
[1]School of Computer Science and Engineering, Southeast University, Nanjing, China
[2]Key Lab of Computer Network and Information Integration(Southeast University),Ministry of Education
[3]School of Computer Science and Technology, Xuzhou Normal University, Xuzhou, China
[4] School of Computer Science, Anhui University of Technology, Ma'anshan, China
{xiechunti@xznu.edu.cn,bx.li@seu.edu.cn}

Abstract

Fault tolerance by redundancy is a key technique for improving web services reliability. But, the reliability and fault tolerance are not well supported by Service-oriented architecture (SOA) conceptual model.Most of the existing analytical models to predict the reliability of web services assume their structures are static which do not conform to reality and the estimated reliability precision is lower. In this paper, first, an extended SOA model is proposed for improving the reliability of web services. Then, a reliability model is presented to evaluate the reliability of web services based on the birth-death process. Last, the reliability of web services is evaluated using simulation approach and a case study is designed, implemented and analyzed to support our model. The results of experiment show the feasibility, validity of our reliability model.

***Keywords*-Web Service Reliability; Birth-Death Process; SOA Conceptual Model**

I. Introduction

Fault tolerance is considered to be an effective way to improve system reliability. In traditional software development, fault tolerance is expensive and only some crucial module can be designed by redundancy. However, at the present time, there are many function-equivalent web services in the Internet which are often provided by different service providers. So, it is practicable to upgrade the reliability of web services by redundancy.

Web services reliability is more hard to predict than general software system. There are several fault-tolerant reliability analysis approaches which have been proposed in [1][2][3][4] and etc, most of their reliability models are based on architecture, mainly considering the static structure and transition probability, without taking into account their dynamic change with time, so the precision is not higher. In fact, the reliabilities of services are influenced by many factors in the Internet environment and a failure can occur at any arbitrary time. The reliability of a service is not a constant but a function of time[5]. Fault tolerance is not well supported by SOA conceptual model, hence, in this paper, we extend the SOA conceptual model by redundancy, propose a fault-tolerant conceptual model and present a reliability model for estimating fault-tolerant web services based on birth-death process.

The rest sections of the paper are organized as follows: Section II describes a fault-tolerant reliability model. Section III describes the reliability prediction algorithm. Section IV does some experiments to verify our model. Section V discusses the conclusion and our future works.

This work is supported partially by the National Natural Science Foundation of China under Grant No. 60773105 and no. 60973149, and partially supported by Doctoral Fund of Ministry of Education of China under Grant No. 20100092110022, and partially by National High Technology Research and Development Program under Grant No.2008AA01Z113, and partially by the National Nature Science Foundation of Xuzhou Normal University under Grant No.10XLA12,and partially by Young Teachers Foundation of Anhui University of Technology under Grant No.QZ201015

Correspondence to: bx.li@seu.edu.cn

II. Fault-Tolerant Reliability Model

A. Extended SOA Conceptual Model

The ESOACM (extended SOA conceptual model) is shown in Figure 1. The web services manager is responsible for all web services. Web services are defined as $WS = \{S, S_1, S_2, \ldots, S_n\}$, S is the primary service and $S_1, \ldots, S_n$ is the backup of S. The web service manager will choose the best service as the primary service. When service requesters send a request for WS, the UDDI

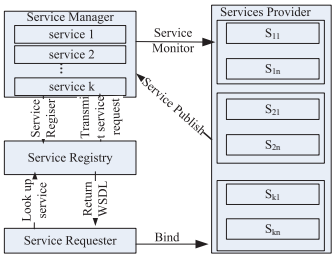


Fig. 1. Extended SOA Conceptual Model

registry will transmit the request to web service manager to look up which service S_i is available in sequence. First, it check S, if S is available, the service manager will return its address to the UDDI registry, then the UDDI registry will return its address to the service requesters. When S fails, the service manager will substitute backup service S_1 for S and notify service provider to repair it and so on.

B. Reliability Model

Now, we discuss how to estimate the reliability of web service in the extended SOA conceptual model. In the extended model, when a service fails, the next backup will replace it. The number of failed web services can be counted by a stochastic process. Stochastic process $\{X(t), t \geq 0\}$ denotes the number of failed web service which have occurred by time t and the state space is $\{0, 1, 2, .., n\}$. The stochastic process can be described by a birth-death process which satisfies some properties:

- Failed, repaired probability for service i in an interval $(t, t + \Delta t)$ is $\lambda_i \Delta t$, $\mu_i \Delta t$.
- Probability of two failures occurred at the same time is zero.
- State can only be transferred between two adjacent state.

State i means the number of failed web services is i and so on. When the process is in state n, the system means failure and no service is available. When a web service occurs failure, the system will go to the next state and when is repaired, it will go back to the previous state. The failure and repair rate of the web service S_i is denoted by $\lambda_i(t)$,μ_i. If failure rate of a web service is a constant, then $\lambda_i(t) = \lambda_i$. The state transition diagram of our stochastic failure model is shown in figure 2.

III. Reliability Prediction Algorithm

In this section, we discuss how to design algorithm. The failure behavior of web services can be denoted by

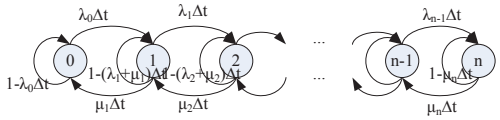


Fig. 2. State Transition Diagram

TABLE I. Notations of our algorithm

Notations	Explanation
r_i	reliability of service i;
R	reliability of the Redundant web services;
λ_i	failure rate of web service i;
μ_i	repair rate of web service i ;
$time$	a variable of $[0, total\_time]$;
dt	time step;
$flag$	failed flag of the process.
$fault\_count$	total number of faults;
run_count	total number of runs;
$total\_time$	the total execution time during a run;

reliability, failure rate. First, we define some notations which are shown in table I.

A. Reliability

Algorithm 1 simulates how to calculate the reliability if the failure behavior is denoted by reliability. It accepts as input the reliability of every service, the number of services including the primary and back-up services and returns the reliability of web services with fault tolerance. The run_count represents the total run times. For each run, if $r_i < x \ and \ i < n$, the procedure will check next service. The service is failed if and only if $r_i < x$ and the state is n. The variable $fault\_count$ records the number of faults. This procedure repeats run_count times and the reliability can be computed just like in algorithm 1.

Algorithm 1 Getreliability1(r, n)

```
Initialize some variables run_count, run, fault_count;
while run ≤ run_count do
    for i = 1 to n do
        Generate a random number x in the range of [0, 1];
        if r_i < x then
            if arrive the state n then
                the model failed and increase fault_count by 1;
            else
                transferred to next state;
            end if
        end if
        increase run by 1;
    end for
    R = 1 − fault_count/run_count;
end while
return R;
```

B. Constant Failure Rate

When failure behavior is denoted by constant failure rate. The stochastic failure procedure is described by a pure-birth process. Algorithm 2 simulates how to calculate the reliability of our model. For each time step dt, we compare $\lambda_i * dt$ with a floating random number x between 0 and 1. If $\lambda_i * dt > x$, we say the ith service fails and turn to check the $(i+1)th$ service by the same way until the nth service. Simulate the above procedure run_count times to obtain the reliability of web services with fault tolerance which can be computed by the expression $R = 1 - fault\_count/run\_count$. The $total\_time$ can be varied and the reliability with time can be computed just like in algorithm 2. From these, we can analysis the change trend of reliability with time.

Algorithm 2 GetReliability2(λ, n)

initialize $run\_count, dt, t, runflag, fault\_count$;
for all $total\_time$ in a range **do**
  for $run = 1$ *to* run_count **do**
    $time = 0, i = 1, flag = false$;
    while $time < total\_time$ *and* $flag$ is $false$ **do**
      $time = time + dt$;
      generate a random number x in range of $[0, 1]$;
      if $\lambda_i * dt > x$ **then**
        if arrive the state n **then**
          $fault\_count[total\_time] + +$;
          $flag = true$;
        else
          transferred to next state;
        end if
      end if
    end while
  end for
  $R(total\_time) =$
    $1 - fault\_count(total\_time)/run\_count$;
end for
return array R;

C. Constant Failure Rate and Repair Rate

When failure behavior is denoted by failure rate λ_i and repair rate μ_i, the procedure is a birth-death process. We presume the repaired time follows an exponential distribution with a parameter μ_i. The simulation procedure is shown in Algorithm 3. Firstly, sample a time t from exponential distribution with parameter $(\lambda_i + \mu_i)$; Secondly, the transition probability from state i to state $i+1$ and to $i-1$ is $p_i = \lambda_i/(\lambda_i + \mu_i)$ and $1 - p_i$. Generate a random number x from $\{0, 1\}$ with probability p_i, the state is transferred to $i+1$ when x is 1 and to state $i-1$ when x is 0. Repeat the above procedure for run_count times and reliability can be computed just like in algorithm 3.

Algorithm 3 GetReliability3(λ, μ, n)

initialize $run\_count, run, flag$ and array $fault\_count$;
for all $total\_time$ in a range **do**
  for $run = 1$ *to* run_count **do**
    $time = 0, i = 1, flag = false$;
    while $time < total\_time$ *and* $flag$ is false **do**
      sample time t from an exponential distribution with a parameter $(\lambda_i + \mu_{i-1})$;
      generate a random number x from $\{0, 1\}$ with probability p_i;
      if x is 1 **then**
        the state is transferred to i+1;
      else
        the state is transferred to i-1;
      end if
      $time = time + t$
      if arrive the state n **then**
        $fault\_count[total\_time] + +$;
        $flag = true$;
      end if
    end while
  end for
  $R(total\_time) =$
    $1 - fault\_count(total\_time)/run\_count$;
end for
return array R;

D. Parameter Estimation

We discuss how to estimate the parameters in our algorithm. The reliability,failure rate and repair rate presented by service provider are not enough to be convincing. So, we reform it by collecting failure and repair information to improve accuracy. In our ESOACM, the server manager use log file to record services behavior information. For parameter reliability r_i, we assume that:

1) an initial value $r_{i,1}$ presented by service provider ;
2) service invoked times n and failure occurred times f in an interval $(0, t)$;
3) tested times N provided by service provider;
4) reliability estimated from log files $r_{i,2}$.

then the revised reliability is calculated by

$$r_i = \frac{N}{N+n} r_{i,1} + \frac{n}{N+n} r_{i,2} \tag{1}$$

where $r_{i,2} = 1 - \frac{f}{n}$. When the service is never called, the invoked times $n = 0$, the reliability $r_i = r_{i,1}$. If the service provider doesn't present the initial reliability $r_{i,1}$, the reliability $r_i = r_{i,2}$. So, the revised reliability represents the static reliability information and the dynamic reliability information. Similar methods can be used for the estimation of λ_i and μ_i which can be expressed by

$$\lambda_i = \frac{N}{N+n} \lambda_{i,1} + \frac{n}{N+n} \lambda_{i,2}. \tag{2}$$

where $\lambda_2 = \frac{f}{t}$. and

$$\mu = \frac{M}{M+k}\mu_{i,1} + \frac{k}{M+k}\mu_{i,2} \quad (3)$$

where $\mu_{i,2} = \frac{k}{M}$ and k is the repaired times in an interval $(0, t)$, M is total repaired times.

IV. Experiments

A. Experiment 1

For algorithm III-A, we set the reliability of services $r = [0.5, 0.5, 0.6, 0.3, 0.6], n = 5$. The run times run_count varies from 100 to 1000 step by 100. Here,we set the time spent on a run is one time unit. So, the number of total run is identify to the times of total run. The results of our experiment are shown in Table II. Column 'Exp1' is the reliability obtained from algorithm III-A and column 'Zheng' is reliability according to Zheng's[3] method. From the table, we can discover that the reliability is very close to Zheng's result.

TABLE II. Results of Experiment 1

Run times	Exp1	Zheng	Run times	Exp1	Zheng
100	0.970	0.972	600	0.973	0.972
200	0.980	0.972	700	0.970	0.972
300	0.973	0.972	800	0.979	0.972
400	0.967	0.972	900	0.970	0.972
500	0.970	0.972	1000	0.967	0.972

B. Experiment 2

For algorithm III-B, we randomly set failure rate $\lambda = [0.0025, 0.003, 0.02, 0.02, 0.02], n = 5$. The run_count is set to 1000 at every specified $total\_time$. We also present the reliability of web services without redundancy. The reliability of single service is calculated by expression $r_i = e^{-\lambda_i t_i}, i = 1, 2, \ldots, n$ λ_i is the failure rate of service i and the t_i is the execution time spent in service i during a run[6]. The reliability with fault-tolerant and the max reliability of single services(MRSS) without fault-tolerant is shown in table III. From the table we can say that the reliability of services by redundancy is higher than a single service and the reliability will approach to zero with the time increased.

C. Experiment 3

To verify the validity of our model and to analyze simply in math, for algorithm III-C, we consider two services with same failure rate and repair rate, the parameter

TABLE III. Results of Experiment 2

Run times	Exp2	MRSS	Run times	Exp2	MRSS
100	0.997	0.779	600	0.654	0.223
200	0.972	0.607	700	0.539	0.174
300	0.918	0.472	800	0.456	0.135
400	0.814	0.3678	900	0.402	0.105
500	0.743	0.287	1000	0.307	0.082

$\lambda = [0.003, 0.003], \mu = [0.5, 0.5]$. The results obtained from our experiment and from the system of differential equations(SDE) of stochastic process are shown in table IV. From the table, we can say that the results of two methods are very close.

TABLE IV. Results of Experiment 3

Run times	Exp3	SDE	Run times	Exp3	SDE
100	0.996	0.998	600	0.992	0.989
200	0.995	0.997	700	0.992	0.989
300	0.996	0.995	800	0.990	0.986
400	0.990	0.993	900	0.991	0.984
500	0.989	0.991	1000	0.986	0.982

V. Conclusions

An extended SOA conceptual model for reliability enhancement by redundancy is presented in this paper and a birth-death process to estimate the reliability of our model is proposed. The birth-death process model is more precision because the information from service provider and log files are both considered. But, there are some details that we have not taken into account. So, our future work will consider other factors that cause composite services failure based on other architecture.

References

[1] Pat.P.W.Chan and M. R. Lyu, "Reliable web services: Methodology, experiment and modeling," in *International Conference on Web Services(ICWS)*, 2007.

[2] C.-L. Fang, D. Liang, F. Lin, and C.-C. Lin, "Fault tolerant web services," *J. Syst. Archit.*, vol. 53, no. 1, pp. 21–38, 2007.

[3] Z. Zheng and M. R. Lyu, "Ws-dream: A distributed reliability assessment mechanism for web services," in *International Conference on Dependable Systems and Networks*, 2008, pp. 392–397.

[4] H. Yu, G. Fan, L. Chen, and D. Liu, "A fault-tolerant strategy for improving the reliability of service composition," in *The 10th International Conference on Quality Software (QSIC)*, 2010, pp. 312–317.

[5] Z. Zheng and M. R. Lyu, "Collaborative reliability prediction of service-oriented systems," in *ICSE '10: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering*, 2010, pp. 35–44.

[6] T. K. Gokhale S.S., "Analytical models for architecture-based software reliability prediction: A unification framework," *IEEE Transactions on Reliability*, vol. 55, no. 4, pp. 578–590, 2006.

Architecture-based Reliability Analysis With Uncertain Parameters

Derek Doran, Matthew Tran, Lance Fiondella, and Swapna S. Gokhale
Dept. of Computer Science & Engineering
University of Connecticut, Storrs, CT, 06269
{derek.doran,lfiondella,matthew.tran,ssg}@engr.uconn.edu

Abstract

Architecture-based reliability analysis has gained prominence in the recent years as a way to predict the reliability of a software application during the design phase, before an investment is made in any implementation. To apply this analysis, the parameters comprising the architectural model must be estimated using the limited data and knowledge available during the design phase. These estimates, as a result, are inherently uncertain. Contemporary approaches, however, do not consider these uncertainties, and hence, may produce inaccurate reliability results. This paper presents a Bayesian approach to systematically consider parametric uncertainties in architecture-based analysis. The novelty of this approach lies in determining credible intervals for the model parameters as a function of their posterior distributions. By leveraging these intervals, we illustrate how to: (i) quantify the impact of uncertainty in a specific parameter on the system reliability estimate; (ii) evaluate when a sufficient amount of data has been collected to reduce the uncertainty to an acceptable level; and (iii) assess the impact of prior knowledge regarding the parameters in improving the system reliability estimate.

1 Introduction

Architecture-based analysis [8, 9] is a powerful approach to assess the reliability of a software application[1] earlier in its lifecycle, and to offer guidance for cost-effective reliability improvement. Such guidance often consists of identifying critical components from the perspective of application reliability, so that additional resources can be allocated to these components to improve their reliability. Architecture-based analysis is thus especially useful in the context of modern, complex software systems which are composed from a variety of components; picked off-the-shelf, built contractually, and developed in-house [13].

In the architecture-based approach, application reliability is expressed in terms of the failure characteristics of the components and the application architecture. The parameters comprising the architectural model thus include the transition probabilities among the components and the reliabilities of the components comprising the application. Thus, the number of parameters (transition probabilities and component reliabilities) that need to be estimated increases with the size of the application. Additionally, these parameters must be estimated based on the limited data or scant knowledge that may be available earlier in the lifecycle before the application is implemented and can be tested extensively. Because of such limited available information, architecture-based analysis must be performed with uncertain parameters. These parametric uncertainties then propagate to the system-level reliability results, which include the reliability estimate and the ranks of the components based on their criticality. Most of the contemporary architecture-based analysis approaches, however, do not consider these parametric uncertainties [2]. Thus, in the best case, the system reliability estimate that these approaches produce underestimate the true reliability, which may result in dedicating more than necessary resources for the system to reach its reliability target. In the worst case, however, the system will be deployed with an optimistic expectation of its reliability [3]. To avoid the pitfalls associated with over and under-estimation, it is necessary to systematically and quantitatively incorporate the impact of uncertain parameters into architecture-based reliability analysis.

In this paper, we present a Bayesian approach to incorporate parametric uncertainties into architecture-based software reliability analysis. The proposed approach constructs joint credible intervals using the posterior distributions of the model parameters as a function of limited information. We illustrate how these credible intervals could be used to explore the influence of uncertain parameters on application reliability. We also demonstrate how the Bayesian approach could be used to quantify the impact of additional data or prior knowledge in reducing the uncertainty in the system-level reliability results.

The paper is organized as follows: Section 2 presents the Bayesian approach. Section 3 illustrates the approach. Section 4 summarizes related research. Section 5 offers concluding remarks and directions for future research.

[1] The terms application, software, and system are used interchangeably throughout this paper.

2 Bayesian analysis methodology

In this section, we introduce a preliminary model for architecture-based reliability analysis that ignores uncertainties in the model parameters. We then describe how this preliminary model can be extended to incorporate parametric uncertainties using the Bayesian approach.

2.1 Preliminary model

We consider a terminating application with n components. We assume that the transfer of control between components is independent and that the failure of a single component causes the entire application to fail. The architecture of the application is represented as a probabilistic control flow graph. This graph is mapped to a discrete time Markov chain (DTMC) which is represented by an $n \times n$ matrix $\mathbf{P}$. The $(i,j)^{th}$ element of $\mathbf{P}$ is the expected value of the random variable $\mathcal{P}_{i,j}$, denoted $p_{i,j}$, and represents the expected probability that control is transferred from component i to component j. Thus, there exists a one-to-one mapping between the application components and the states of the DTMC. Without loss of generality, we assume that the application starts execution with component one and terminates after component n executes, so the n^{th} row of $\mathbf{P}$ contains all zeros.

We let R_i denote the expected reliability of component i. Component reliability is binary, that is, when control transitions to component i, the probability that it executes successfully is represented by the random variable Λ_i whose expected value is R_i. If a component does not execute successfully, the entire system fails. The expected application reliability can be computed using the following algorithm:

1. Define $\mathbf{D}$ as a diagonal matrix where $\mathbf{D}_{i,i} = R_i$
2. Let $\mathbf{Q} = \mathbf{D} \times \mathbf{P}$
3. Solve $\mathbf{S} = \sum_{k=0}^{\infty} \mathbf{Q}^k = (\mathbf{I} - \mathbf{Q})^{-1}$
4. System reliability is given as $\mathbf{S}_{1,n} \cdot R_n$

The matrix $\mathbf{Q}$ is a sub-stochastic matrix where $q_{i,j}$ is the transition probability among components i and j, weighted by the reliability of component i. This captures the notion that a transition from component i to component j occurs conditional to the successful execution of component i, else component i fails causing the application to fail. Using $\mathbf{S}$ to obtain the expected number of times the application reaches the end state n starting from state 1, the system reliability is thus given by $\mathbf{S}_{1,n} \cdot R_n$.

2.2 Incorporating parametric uncertainties

In this section, we discuss our approach to incorporate parametric uncertainties into the preliminary model presented above. Towards this end, we use Bayesian techniques to construct credible intervals for each parameter, such that the true value of the parameter lies within that credible interval at a specified confidence level. The credible interval will provide the range over which each parameter must be varied in order to systematically explore the impact of its uncertainty on the application-level reliability results.

2.2.1 Posterior distribution

The parameters of the architecture-based model include the intercomponent transition probabilities, and the component reliabilities. To construct a credible interval for each parameter, it is first necessary to determine its posterior distribution. A point estimate for the expectation of Λ_i is given by $R_i = x_i/n_i$, where x_i is the number of successful executions of the component across n_i trials. Each x_i then follows a Binomial distribution:

$$Bin(x_i, R_i) = \binom{n_i}{x_i} R_i^{x_i} (1 - R_i)^{n_i - x_i}.$$

We can define a *posterior* distribution $G(\Lambda_i|x_i)$ of the random variable representing component reliability, incorporating any prior knowledge we have about this distribution with additional data we have obtained since then, as follows. Let $g(\Lambda_i)$ be the *prior* distribution of Λ_i, incorporating any *a priori* knowledge about Λ_i, and $f(x_i|\Lambda_i)$ be the likelihood function that provides the probability of observing x_i given Λ_i. From Bayes theorem [1], it follows that:

$$G(\Lambda_i|X) = \frac{g(\Lambda_i) f(X|\Lambda_i)}{\int_0^1 g(\Lambda_i) f(X|R_i) d\Lambda_i}.$$

Since the expected value of Λ_i is used as the parameter of a Binomial distribution, we know that Λ_i has a prior and a posterior that follows a Beta distribution, given as:

$$Beta(\Lambda_i|a_i, b_i) = \frac{\Gamma(a_i + b_i)}{\Gamma(a_i)\Gamma(b_i)} \Lambda_i^{a_i - 1} (1 - \Lambda_i)^{b_i - 1}.$$

The parameters of the posterior can be expressed in terms of the parameters of the prior. If $g(\Lambda_i) \sim Beta(\Lambda_i|\alpha, \gamma)$, then $G(\Lambda_i|x_i) \sim Beta(\alpha + x_i, \gamma + (n_i - x_i))$ [4].

Next, we determine the posterior distribution for each random variable $\mathcal{P}_{i,j}$ representing the transition probabilities in the architecture. The expectation $p_{i,j}$ of this random variable represents the average probability of transitioning from component i to component j. This average transition probability can be estimated as $p_{i,j} = x_{i,j}/n_i$ where $x_{i,j}$ is the number of times control is transferred from component i to component j and $n_i = \sum_j x_{i,j}$ is the total number of times control is transferred out of component i. The number of times control is transferred to various components from component i thus follows a Multinomial distribution [12]:

$$Multinomial(X_i, k_i) = \binom{k_i}{x_{i,1} x_{i,2} \ldots x_{i,k}} p_{i,1}^{x_{i,1}} p_{i,2}^{x_{i,2}} \ldots p_{i,k}^{x_{i,k}}$$

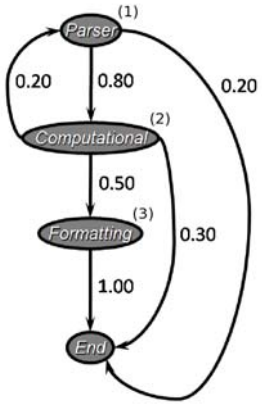


Figure 1. Architecture of the ESA application

Param.	Estimate	True Value	% Difference
R_1	0.932	0.840	9.87%
R_2	0.800	0.830	3.75%
R_3	0.933	0.950	1.82%
$p_{1,2}$	0.818	0.800	2.20%
$p_{1,4}$	0.182	0.200	9.89%
$p_{2,1}$	0.250	0.200	20.00%
$p_{2,3}$	0.417	0.500	19.90%
$p_{2,4}$	0.333	0.300	10.0%
$p_{3,4}$	1.000	1.000	0.000%
App. Reliability	0.7200	0.6756	6.57%

Table 1. Parameter estimates based on limited data

where k_i is the number downstream components that control can be transferred to, $X_i = (x_{i,1}, x_{i,2}, ...x_{i,k_i})$, and $\sum_{j=1}^{k_i} x_{i,j} = n_i$. Let $P_i = (\mathcal{P}_{i,1}, \mathcal{P}_{i,2}, ...\mathcal{P}_{i,k_i})$ be the vector whose elements are the random variables that represent the transition probabilities associated with state i. These random variables have an expectation of $p_{i,1}, p_{1,2}, ..., p_{i,k_i}$. It is important to note that $k_i \leq n$, that is, the number of downstream components that component i can transfer control to, in general, may be less than the total number of components in the application. When the draws from the Multinomial distribution are independent, the posterior and prior distribution for P_i follows a Dirichlet distribution, given by:

$$Dir(P_i|\Psi_i) = \frac{\Gamma(\psi_{i,1} + \psi_{i,2} + ... + \psi_{i,k_i})}{\Gamma(\psi_{i,1})\Gamma(\psi_{i,2})...\Gamma(\psi_{i,k_i})} \prod_{j=1}^{k_i} p_{i,j}^{\psi_{i,j}-1}.$$

Here, $\Psi_i = (\psi_{i,1}, \psi_{i,2}, ..., \psi_{i,k_i}), \psi_{i,j} > 0 \ \forall j$ is the set of parameters representing the number of times each transition was followed, $p_{i,j} \geq 0$, and $\sum_{j=1}^{k_i} p_{i,j} = 1$. Since the marginals of the $Dir(P_i|\Psi_i)$ are $Beta(\psi_{i,j}, \psi - \psi_{i,j})$ distributed, where $\psi = \sum_{j=1}^{k_i} \psi_{i,j}$, it follows that the prior and posterior of each $\mathcal{P}_{i,j}$ are Beta distributed. Letting $\sum_{j=1}^{k_i} x_{i,j} = n_i$, if $g(\mathcal{P}_{i,j}) \sim Beta(\alpha, \gamma)$, then its posterior follows a $Beta(\alpha + x_{i,j}, \gamma + n_i - x_{i,j})$ distribution [4].

2.2.2 Bayesian credible intervals

With the posterior distribution derived for each model parameter, a Bayesian credible interval can be established that provides a confidence level within which the true value of the parameter lies. For model parameter π_i (either a component reliability or a transition probability) that has a posterior distribution $G(\pi_i|X)$, the bounds of the $(1-\alpha)\%$ Bayesian credible interval is the value y_i for which:

$$\int_0^{y_i} G(\pi_i|X)d\pi_i = \epsilon$$

where $\epsilon = \alpha/2$ defines the lower and $\epsilon = 1 - \alpha/2$ defines the upper bound of the interval. These bounds for all the parameters can be computed using the inverse CDF function of the Beta distribution because the posterior of all the model parameters follows this distribution. This inverse CDF is available in software packages such as MATLAB and Excel [11].

3 Illustrations

This section illustrates the Bayesian approach to incorporate parametric uncertainties using an application from the European Space Agency (ESA) to configure an array of antennas. The architecture of the ESA application, along with the true intercomponent transition probabilities, are shown in Figure 1. The application consists of four components: the $Parser$ component, the $Computational$ component, the $Formatting$ component, and the End component, assigned as components 1, 2, 3, and 4, respectively. For the sake of illustration, we set the reliability of the $Parser$ to 0.84, the $Computational$ component to 0.83, and the $Formatting$ component to 0.95, resulting in a true system reliability of 0.6756. Based on these true values, in this section, we first illustrate the impact of uncertain parameters on the system-level reliability results. Subsequently, we illustrate how the Bayesian approach can be used to systematically assess the impact of parametric uncertainties on system reliability and to provide guidance on reducing the uncertainty in system-level reliability.

3.1 Reliability estimation

To demonstrate how limited data leads to parametric uncertainties, we simulated the operation of the ESA application 50 times. During these 50 runs, we recorded the number of times control transfers among a pair of components, the number of times control transfers out of each component, and the number of failures of each component. Based on these observations, we estimate the component reliabilities and intercomponent transition probabilities. These estimates, along with their true values, are recorded in Table 1.

Parameter	IP (Rank), Est.	IP (Rank), Actual
$Parser$	.0626 (2)	.1484 (1)
$Computational$	.1701 (1)	.1201 (2)
$Formatting$	.0200 (3)	.0157 (3)

Table 2. Component ranks with and without uncertainty

These estimates of the model parameters lead to a system reliability estimate of 0.7200, which is off by 6.17% from its true value. Comparing the reliability estimate obtained from limited data with its true value highlights how the uncertainties in the parameters can propagate to system reliability.

3.2 Component ranks

The architecture-based approach allows us to rank the application components according to their importance measures [3] or significance from the point of view of system reliability. In this section, we demonstrate how uncertainties in the parameter estimates can lead to incorrect ranks. For the sake of illustration, we rank each component i based on its Improvement Potential (IP) measure [3], which is defined as the difference between the system reliability estimate when R_i is 1 and the estimate when R_i is assigned its true or estimated value. This expresses the maximum gain in system reliability that can be attained by devoting resources toward perfecting component i. Table 2 shows the IP of each component when using its true and estimated values. While the true IP ranking reveals the $Parser$ as having the highest improvement potential, using estimated parameters causes the $Computational$ component to be ranked higher than the $Parser$. Parametric uncertainties can thus result in a mis-allocation of resources towards prioritizing the $Computational$ component, when in reality improvements to the $Parser$ component will deliver the largest gain in application reliability.

In order to alleviate the impact of making such wrong decisions, it is necessary to perform a sensitivity analysis for each component across a range of parameter values, and use the insights gained from that analysis to evaluate the validity of the component ranks. The more sensitive the system-level reliability is to the reliability of component i, it is more likely that its estimated component rank is incorrect.

3.3 Sensitivity analysis

In order to explore the sensitivity of the reliability estimate to the individual model parameters, we present a study where we set a single model parameter to each bound of its 95% credible interval and then estimate the application reliability. The reliability value evaluated at each bound represents the 95% credible interval for system reliability with respect to the uncertainty in that single parameter. We define the *reliability uncertainty* (RU) metric as the effect of a parameter's uncertainty on application reliability. This metric is given as the size of the credible interval of the application reliability, found by computing the system reliability at the two extreme values of the parameter's credible interval. If the RU for a parameter is very small, we will have a high level of certainty in the reliability estimate, but as RU increases we can only be certain that the system reliability estimate is off by an increasingly higher percentage. The study makes a worse-case assessment by assuming no prior knowledge about any of the parameters, and hence, uses the uniform priors [4] $g(\Lambda_i) \sim Beta(1,1)$ and $g(\mathcal{P}_i) \sim Dir(1,1,...1)$ for all of the parameters.

Param	Credible Interval	RU Value	Rank
R_1	(0.8380 , 0.9724)	0.1212	7
R_2	(0.6609 , 0.8905)	0.1846	4
R_3	(0.6977 , 0.9845)	0.0860	8
$p_{1,2}$	(0.6960 , 0.8977)	0.1587	6
$p_{1,4}$	(0.1023 , 0.3040)	0.2218	3
$p_{2,1}$	(0.1295 , 0.4259)	0.1619	5
$p_{2,3}$	(0.2595 , 0.5923)	0.2237	2
$p_{2,4}$	(0.1920 , 0.5116)	0.2301	1
$p_{3,4}$	(1.0000 , 1.0000)	0.0000	9

Table 3. Credible intervals of the parameters and their impact on application reliability

Table 3 presents the credible interval for each parameter and its resulting RU. The table illustrates how uncertainties in the parameters reduce the confidence in the application reliability. For example, when parameter $p_{2,3}$ is off by approximately by 20%, there is a 95% chance that it causes an uncertainty of up to 24% in the application reliability. Similarly, uncertainty in the parameters $p_{2,4}, p_{2,3}$, and R_2 results in up to 25.57% uncertainty in the application reliability. Furthermore, R_2 has the highest RU value among all components, and corresponds to the component that was incorrectly ranked as having the highest IP. This illustrates how sensitivity analysis can be used to decide on the accuracy of the derived importance ranks of the components.

We also observe that the level of uncertainty in a parameter seems unrelated to the resulting degree of uncertainty in the application-level reliability. For example, parameter $p_{1,4}$ has the smallest non-zero credible interval among all the transition probabilities, but is ranked as having the 3^{rd} most significant impact on the system reliability. This means that the level of uncertainty in a parameter is not the only factor that determines its impact on the level of uncertainty in the application reliability. Another factor may include the frequency with which a transition is taken or a component is executed during a typical run of the application. This frequency indicates the significance of a given transition or a component to the application reliability. Although these significant transitions and components will have smaller credible intervals (because these will be

estimated using a larger number of observations), the uncertainties in them will still have a disproportionate influence on the uncertainty in application-level reliability.

3.4 Reducing parametric uncertainties

The previous sections highlighted the impact of parametric uncertainties on the system-level reliability results. The uncertainties in the parameters arise because of the limited data used in the estimation process. Thus, to reduce these uncertainties, it is necessary to either use a large amount of data or to incorporate prior knowledge while estimating the parameters. In the following illustrations, we show the impact of either approach on reducing the uncertainty.

3.4.1 Increasing data

Increasing the amount of data used in the estimation process can improve the estimates of the parameters. In the limit, however, for these parameters to approach their true values, an infinite amount of data is required. This is not feasible, and hence, a trade-off between the level of uncertainty that can be tolerated and the amount of data used in the estimation process must be reached.

In Figure 2, the level of uncertainty in the application reliability versus the number of simulation trials used to collect data for parameter estimation is plotted. The figure shows that the degree of uncertainty in system reliability decreases at an exponential rate as the number of simulation runs increases. In this illustration, all of the data used to estimate the parameters is collected from the simulation. In practice, however, this data may come from a variety of sources. Thus, the figure more generally illustrates that devoting a small cost towards improving our pre-existing knowledge can lead to a measurable increase in the accuracy of the system reliability estimate. Devoting too much cost towards gathering this knowledge, however, may be wasteful because the level of uncertainty in the system reliability will not appreciably decrease beyond a certain threshold.

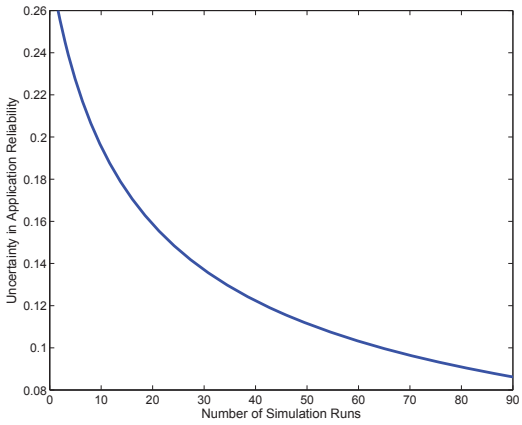


Figure 2. Relationship between reliability uncertainty and available data

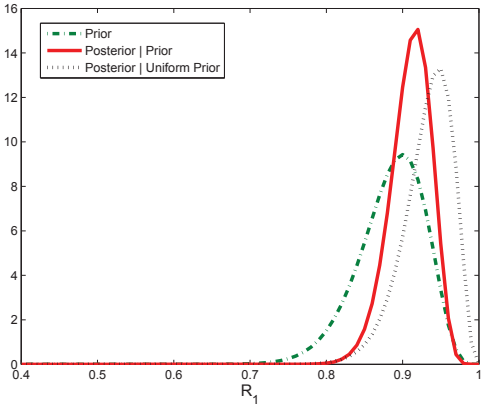


Figure 3. Prior and posterior distributions for the reliability of the *Parser* component

3.4.2 Incorporating prior knowledge

An alternative way to reduce uncertainty is to incorporate prior knowledge in the parameter estimation process. We illustrate this by introducing a non-uniform prior for the reliability of the *Parser* component. This non-uniform prior may be derived based on: (i) how this component operated in a different system; (ii) data provided by the manufacturer; or (iii) our prior experience in its use. We say that the *Parser* component is being reused from a previous system, where it exhibited a reliability of 0.9 across 50 uses.

Figure 3 shows the shape of the resulting prior distribution, the posterior distribution derived with a uniform prior, and the posterior distribution that incorporates this non-uniform prior for the $Parser$ component. Under a uniform prior, the posterior distribution has a wide body whose mean value is below its true reliability. The prior distribution, however, overstates the true reliability of the component and exhibits a similarly wide body as the posterior under the uniform prior.

In incorporating this prior knowledge, we get a new posterior distribution with a mean value of 0.9099. This mean is closer to the true component reliability of 0.84 when compared to the estimate found under a uniform prior (0.932). This new component reliability estimate results in a new system reliability estimate of 0.6998, which is inaccurate by only 3.52%, compared to 6.57% in the case of using a uniform prior for the $Parser$ reliability. Furthermore, the sharper peak of this new posterior reduces the size of the 95% credible interval for the component from 0.1212 to 0.1051, resulting in a reduction in the impact of its uncertainty on the application reliability from 0.1846 to 0.0947.

4 Related research

In this section, we summarize and compare our work to related efforts. Goseva *et al.* [6] model component reliabilities and transitions with Binomial and Multinomial distributions respectively. However, they only estimate the model parameters based on the Beta and Dirichlet distributions. A more recent study [5] applies their Bayesian approach to an empirical case study of the GCC compiler. Uncertainty analysis based on perturbation theory has also been proposed in the context of architecture-based analysis [10, 7].

Unlike previous approaches, this research employs Bayesian techniques to compute credible intervals for all the model parameters and proposes an approach to quantify the impact of uncertainty in the parameters on the uncertainty in the system reliability. This approach thus provides a practical tool for test planners who must identify where to target limited resources to improve the confidence in the reliability of their software application.

5 Conclusions and future research

This paper introduced a Bayesian approach to incorporate parametric uncertainties in the assessment of software application reliability based on its architecture. We illustrated how the approach could be used to systematically assess the impact of uncertainties in the parameters on system-level reliability in a systematic, quantitative manner. The case study suggests that there exist additional factors besides uncertainty in the parameter itself that can affect its impact on system-level reliability. We also explored how the level of uncertainty in the application reliability reduces as a function of increasing data or prior knowledge used in the estimation process.

Our future research seeks to extend the uncertainty quantification procedures to importance assessment and optimization. In addition, understanding how uncertainty across multiple model parameters affects the uncertainty in the application reliability needs to be investigated.

Acknowledgements

This research was supported by a CAREER award from NSF (#CNS-0643971).

References

[1] W. M. Bolstad. *Introduction to Bayesian Statistics*. John Wiley & Sons, Inc., 2nd edition, 2007.

[2] R. Cheung. A User-Oriented Software Reliability Model. *IEEE Trans. Software Eng.*, 6(2):118–125, March 1980.

[3] L. Fiondella and S. Gokhale. Importance Measures for Modular Software with Uncertain Parameters. *Software Testing, Verification, and Reliability*, 20(1):63–85, 2010.

[4] A. Gelman. *Bayesian Data Analysis*. Taylor & Francis, 2003.

[5] K. Goševa-Popstojanova, M. Hamill, and X. Wang. Adequacy, Accuracy, Scalability, and Uncertainty of Architecture-based Software Reliability: Lessons Learned from Large Empirical Case Studies. In *Proc. Intl. Symposium on Software Reliability Engineering*, pages 197–203, Raleigh, NC, November 2006.

[6] K. Goševa-Popstojanova and S. Kamavaram. Assessing Uncertainty in Reliability of Component-Based Software Systems. In *Proc. of Intl. Symposium on Software Reliability Engineering*, pages 307–320, Denver, CO, nov 2003.

[7] K. Goševa-Popstojanova and S. Kamavaram. Software Reliability Estimation under Uncertainty: Generalization of the Method of Moments. In *HASE'04*, pages 209–218, Tampa, FL, mar 2004.

[8] K. Goševa-Popstojanova and K. Trivedi. Architecture Based Approach to Quantitative Assessment of Software Systems. *Performance Evaluation*, 45(2-3):179–204, June 2001.

[9] S. Gokhale and K. Trivedi. Analytical Models for Architecture-Based Software Reliability Prediction: A Unification Framework. *IEEE Trans. Rel.*, 55(4):578–590, December 2006.

[10] S. Kamavaram and K. Goševa-Popstojanova. Sensitivity of Software Usage to Changes in the Operational Profile. In *NASA/IEEE Software Engineering Workshop*, pages 157–164, Greenbelt, MD, dec 2003.

[11] S. Karris. *Numerical Analysis: Using MATLAB and Excel*. Orchard Publications, 3rd edition, 2007.

[12] K. S. Trivedi. *Probability and Statistics with Reliability, Queueing, and Computer Science Applications*. John Wiley & Sons, Inc., 2nd edition, 2002.

[13] S. Wadekar and S. Gokhale. Exploring Cost and Reliability Tradeoffs in Architectual Alternatives Using a Genetic Algorithm. In *Proc. of Intl. Symposium on Software Reliability Engineering (ISSRE 99)*, pages 104–113, Boca Raton, FL, nov 1999.

Architecture-based Reliability Analysis of Concurrent Software Applications using Stochastic Reward Nets

Rehab El Kharboutly
Mathematics and Computer Science
Eastern Connecticut State University
Willimantic, CT 06226
elkharboutlyr@easternct.edu

Swapna S. Gokhale
Dept. of Computer Science and Engineering
University of Connecticut,
Storrs, CT 06269
ssg@engr.uconn.edu

***Abstract*—Architecture-based reliability analysis of software applications has been the focus of several recent research efforts, as these applications continue to grow in size and complexity. Prevalent research in the area of architecture-based analysis predominantly focuses on sequential applications; with a few exceptions that address concurrency. Concurrency, however, is very common in modern software applications that are developed using the object-oriented or the component-based software development paradigms. Reliability analysis considering concurrency within the context of the application architecture is thus necessary for modern software applications. Our preliminary approach to analyze the reliability of a concurrent software application suffers from state-space explosion; due to which it cannot be applied to practical software applications. In this paper, we offer a methodology based on the Stochastic Reward Net (SRN) modeling paradigm to alleviate the model specification challenge associated with the state-space explosion issue. Our method uses SRNs to intuitively and concisely specify the architecture of a concurrent software application. We demonstrate how the expressive power of SRNs can be used to specify execution constraints, further reducing the state space. Finally, we illustrate how SRN-based specification can facilitate sensitivity analysis through an example.**

I. INTRODUCTION

Software applications provide a number of critical functions in many domains including telecommunications, healthcare, and finance. Ensuring that these applications meet or exceed their reliability requirements is therefore necessary for their widespread adoption. As the size and complexity of software applications continues to grow, analytical evaluation of their reliability is becoming an important part of the application design process. Analytical methods incorporate the application architecture in order to identify those application components that are critical from the perspective of its reliability. Identifying such critical components in the early phases can guide the allocation of resources to these components, so that the application reliability can be improved in a cost-effective manner.

Prevalent research in the area of architecture-based software reliability analysis [1-5] has predominantly focused on sequential applications, where control resides in one component at a time. The assumption of sequential execution was valid for applications that were developed using the procedural programming paradigm. Many modern software applications, however, are built using the object-oriented and component-based software development paradigms, and concurrent execution of components is very common in these applications [6]. Reliability analysis of a software application considering concurrency within the context of the application architecture is thus essential for contemporary, concurrent software applications.

In this paper, we present an efficient approach for architecture-based reliability analysis of a software application considering concurrent component execution. We build upon our preliminary Markov model to represent component-level concurrency in the application architecture [7, 8]. A major shortcoming of this preliminary approach is state-space explosion; where the number of states in the model grows exponentially as a function of the number of components in the application. State-space explosion poses two challenges in applying this approach to large, practical applications. First, the process of specifying large models necessary to represent applications with even a moderate number of components is cumbersome and error prone. Second, the solution of these large models can be computationally intractable. These specification and solution challenges intrinsic to the preliminary approach must be mitigated in order to apply this approach to practical applications.

This paper addresses the specification challenge through the use of Stochastic Reward Nets (SRNs) [9]. We choose SRNs to concisely represent the architecture of a concurrent software application for several reasons. First, SRNs offer an intuitive way to represent concurrent behavior. Additionally, they provide a high-level interface for automatically generating the underlying Markov model. Finally, SRNs provide powerful support for modeling execution constraints within the architecture, and such constraints may further reduce the state space of the Markov model, thereby indirectly alleviating the model solution challenge. We illustrate how the SRN-based specification of a concurrent application can facilitate sensitivity analysis and identification of bottlenecks.

This paper is organized as follows: Section 2 presents an overview of our preliminary model. Section 3 describes how large models can be specified using SRNs. Section 4 illustrates the use of SRNs to represent constraints. Section 5 summarizes related research. Section 6 offers concluding remarks and directions for future research.

II. PRELIMINARY MODEL

In this section, we present our preliminary model, and discuss the associated state-space explosion issue. We consider a continuously running application [8] which runs forever unless it fails or is stopped on purpose. We assume that at any given time the application state may be represented by the list of components under execution. The application state thus evolves as components begin and suspend execution, and this evolution can be represented using a state-space model.

We illustrate the architectural representation using an example of application consisting of three components. Let C_i denote component i. Figure 1 shows the complete state-space model of the application architecture. If exactly one application component is executing, then the application could be in states $<C_1>$, $<C_2>$ *or* $<C_3>$. In state $<C_1>$, either C_2 and C_3 can begin execution, or C_1 can terminate. If C_2 begins, then the application transitions to state $<C_1,C_2>$, if C_3 begins then the application transitions to state $<C_1,C_3>$, and if C_1 terminates then the application transitions to state $<SUSP>$, where all the components are in a suspended state. Depending upon which component begins execution from the state $<SUSP>$, the application transitions back to either state $<C_1>$, $<C_2>$ or $<C_3>$. The states and state transitions starting from states $<C_2>$ and $<C_3>$ to $<SUSP>$ can be generated using similar reasoning.

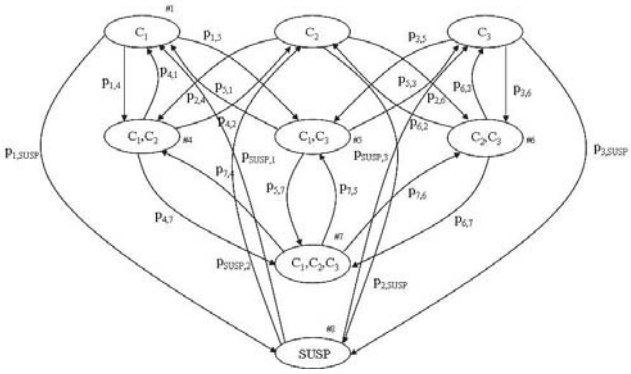


Figure 1 State-space model of the architecture

Next, we describe how expressions for the application reliability can be obtained based on the above representation of the architecture. For this purpose, we let k denote the number of states in which at least one component is executing. We exclude the $<SUSP>$ state in the count, since in this state no component is active, due to which, we assume that the application cannot fail. We let n denote the total number of components. We assume that component C_i fails according to an exponential distribution with failure rate λ_i. We let π_j denote the steady-state probability of state j. π_j is obtained by mapping the state-space representation of the architecture to a Continuous Time Markov Chain (CTMC) model and solving the CTMC model using the numerical methods built into tools such as MATLAB [10].

We assume that a component failure causes the application to fail and is the result of an internal error. As a result, component failure will depend on its total execution time, cumulative over cycles or service requests, which is obtained as follows. For a given execution time t, the average execution time in state j denoted t_j is $t\pi_j$. The execution time of each active component in the state is t_j/c_j, where c_j is the number of active components in state j. Thus, the cumulative execution time spent in a component, as a function of the total execution time is given by:

$$\omega_i(t) = \sum_{j=1}^{k} \pi_j \frac{t}{c_j} I_{i,j} \qquad (1)$$

where $I_{i,j} = 1.0, 1 \le i \le n, 1 \le j \le k$, if component j is active in state i and 0 otherwise.

R_i, the expected reliability of component i, is given by:

$$R_i = e^{-\lambda_i \sum_{j=1}^{k} \pi_j \frac{t}{c_j} I_{i,j}} \qquad (2)$$

Based on Equation (2), the expected application reliability, denoted $R(t)$, is given by:

$$R(t) = \prod_{i=1}^{n} R_i = \prod_{i=1}^{n} e^{-\lambda_i \omega_i(t)} = e^{-\sum_{i=1}^{n} \lambda_i \sum_{j=1}^{k} \pi_j \frac{t}{c_i} I_{i,j}} \qquad (3)$$

Figure 1 shows that an application, where its three components can execute simultaneously, has 8 states in the state-space representation of its architecture. Extending the reasoning, an application with n components will have 2^n states in its representation. Thus, the state-space representation will grow exponentially as a function of n, making it infeasible to both specify and solve the model.

III. SRN SPECIFICATION

The previous section highlights how the representation of the architecture results in state-space explosion. State-space explosion raises model specification and solution challenges, which must be mitigated in order to apply this model to real-life applications. In this paper, we assume that a *tolerance strategy* [8] exists, that is, the large model can be solved using the built-in capabilities in many tools. Matlab for example, can solve the model for an application with 15 components with a state space of 2^{15} x 2^{15}. To represent this model, however, a user has to manually create a 2^{15} x 2^{15} matrix and set the values of 163,840 (2^{15}.15) transitions, which is time consuming and error prone. Therefore, we demonstrate how the model can be specified using a high-level specification mechanism of Stochastic Reward Nets (SRNs). In this section, we first provide an overview of SRNs, and then illustrate how SRNs can intuitively represent the architecture.

a. Overview of SRNs

Stochastic Reward Nets (SRNs) are a descendant of Petri Nets (PNs) in which a reward function could be associated with the markings [6]. PNs have been further extended to

Generalized stochastic Petri nets (GSPNs) by allowing transitions to have zero and exponentially distributed firing times [11]. SRNs are considered to be a superset of GSPNs [6]. SRNs are a richer modeling paradigm due to additional constructs such as guard functions, marking-dependent arc multiplicities and reward rates at the net level. We choose SRNs because of: (i) their expressive power which allows intuitive specification of real-life constrains; and (ii) the ability to solve a SRN model using numerical techniques incorporated in the Stochastic Petri Net Package (SPNP) [9].

A SRN is a directed graph, which contains two types of nodes: places and transitions. A directed arc connecting a place (transition) to a transition (place) is called an *input (output) arc*. Arcs are associated with a positive integer called *multiplicity*. Places can contain *tokens* that move from one place to another through transitions. A transition is enabled when each of the places connected to it by its input arc has at least the number of tokens equal to the multiplicity of those arcs. When an enabled transition fires, a number of tokens equal to the input arc multiplicity is removed from each of the corresponding input places, and a number of tokens equal to the output arc multiplicity is deposited in each of the corresponding output places. A SRN may also include an *inhibitor arc*, which can also have a multiplicity associated with it. An inhibitor arc inhibits the transition it is connected to if the place it is connected to at its other end has a number of tokens equal to at least its multiplicity. The state of a SRN with P places, represented by a vector $(m_1,m_2, \cdots ,m_p)$, is called as a marking of the SRN, where m_i is the number of tokens in place i. The finite set of all possible markings is called the *reachability set*. The markings of a SRN and transition rates among them are equivalent to the corresponding states and state transitions of a continuous time Markov chain (CTMC) [9].

To extend the power of specification, a SRN may also specify enabling (or guard) functions for each transition. SRNs substantially extend the modeling power of Generalized Stochastic Petri Nets (GSPNs) [15], which are an extension of Petri nets [24]. SRNs represent a powerful modeling technique with concise specification and form closer to designer's intuition. This makes it easier to transfer the results obtained from solving the model and interpret them in terms of the entities in the system being modeled.

b. SRN model

We describe the process of specifying the architecture of a concurrent application using SRNs using the three-component example. We represent the working and idle states of each component using places and the sojourn time in each state using transitions. Thus, places C_i_wrk, and C_i_idl, represent the idle and working states of component C_i. Transitions *Tiwak* and *Tislp*, represent the transitions from working to idle and from idle to working states for C_i. Tokens (or a lack thereof) in each place represent the state of the system. In this problem, each component has only one token, which can either be in the place C_i_wrk or C_i_idl indicating that the component could either be in the working or in the idle states but not both. Figure 2 shows the SRN model of the architecture. The notation representing the component parameters is in Table 1.

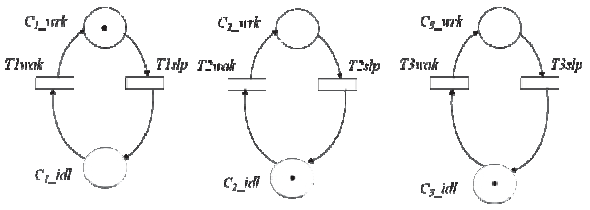


Figure 2 SRN model of three-component concurrent application

TABLE 1 EXECUTION AND FAILURE RATES OF COMPONENTS

Comp #	Transition rates (/min)		Failure rate (λ_i)
	Working to Idle (ρ_i)	Idle to Working (γ_i)	
C_1	0.3	0.5	0.0001
C_2	0.5	0.1	0.0001
C_3	0.2	0.4	0.0001

The reachability set, which is the state space of the underlying CTMC, is automatically generated from the SRN model using SPNP and is shown in Table 2. The second column shows the active components in each marking. The Markov chain corresponding to the reachability set is shown in Figure 3. SPNP also computes the steady-state probabilities as shown in Table 4.

TABLE 2 REACHABILITY SET GENERATED FROM THE SRN MODEL

	Active comp.	C1_ idl	C1_ wrk	C3_ wrk	C2_ wrk	C3_ idl	C2_ idl
M_0	All idle	1	0	0	0	1	1
M_1	C_1	0	1	0	0	1	1
M_2	C_2	1	0	0	1	1	0
M_3	C_3	1	0	1	0	0	1
M_4	C_1,C_2	0	1	0	1	1	0
M_5	C_1,C_3	0	1	1	0	0	1
M_6	C_2,C_3	1	0	1	1	0	0
M_7	C_1,C_2,C_3	0	1	1	1	0	0

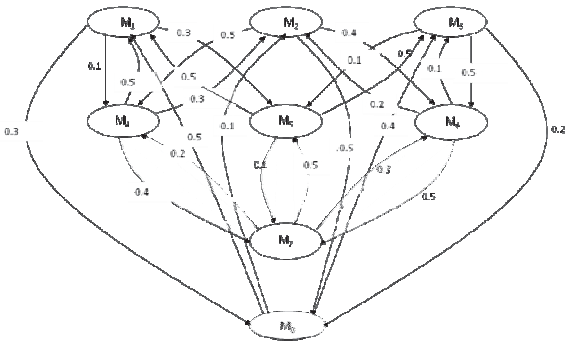


Figure 3 Markov chain generated from the SRN specification

TABLE 3 STEADY STATE PROBABILITIES

State	Steady Probability	State	Steady Probability
M_0	0.1042	M_4	0.03472
M_1	0.1736	M_5	0.347
M_2	0.0208	M_6	0.0417
M_3	0.2083	M_7	0.0594

We assume that the execution time of the application is 100 minutes. Using the failure rates of the components from Table 1, along with the steady-state probabilities from Table 4, the system reliability is computed to be 0.964. This indicates that the execution is successful in 96% of the runs.

c. Sensitivity anlaysis

In addition to reliability estimation, at design time, it is also useful to assess the sensitivity of the application reliability with respect to component parameters. Such sensitivity analysis can identify critical components, and these components can then be allocated additional resources to improve their reliability. Compared to our preliminary approach, the SRN model has fewer parameters; and these are defined at the level of components, rather than at the level of the architectural states. Therefore, the SRN model facilitates sensitivity analysis in a concise and intuitive manner. In this section, we demonstrate how the SRN model could be used for sensitivity analysis.

The reliability of a component is a function of its expected execution time, which in turn is a function of its transition rates from the working to the idle, and from the idle to the working states. Therefore, to assess the relative impacts of the components on the application reliability, we vary their transition rates out of the working states by ±20% in steps of 5% around the initial values, one at a time, while holding all the other parameters constant. We solve the SRN model for each transition rate, and use the resulting steady state probabilities to compute the application reliability.

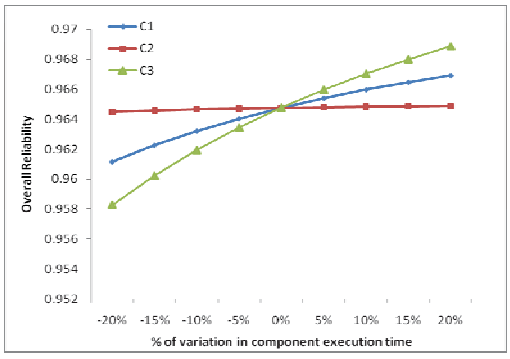


Figure 4 Application reliability as a function of component parameters

Figure 4 shows the application reliability as a function of the expected execution times of the components. The figure indicates that C_3 has the strongest impact on the application reliability, followed by C_1. C_2 has the least impact. This can be explained by considering the initial values of the transition rates listed in Table 1. The table suggests that the transition rate from the working to the idle state is higher in the case of C_1 than in the case of C_3. However, the transition rate from the idle to the working state is lower for C_3 than that of C_1. These transition rates suggest that C_1 and C_3 will have comparable execution times, making it difficult to determine intuitively which one of these will have a higher impact on the application reliability. Our quantitative sensitivity analysis, however, definitively ranks C_3 over C_1, providing solid guidance on devoting resources to reducing the expected execution time of C_3 in order to improve its reliability. Finally, C_2 transitions from the working to the idle states comparatively frequently, and spends a significant portion of the time in the idle state, due to which it is expected to have the least impact on the application reliability. Our analysis results confirm this intuition.

IV. MODELING EXECUTION CONSTRAINTS

SRNs provide powerful constructs such as inhibitor arcs and guard functions to represent execution constraints. Such constraints may also reduce the state space, and hence, alleviate the model solution challenge. We illustrate the modeling and expressive power of SRNs using an application with four components C_1, C_2, C_3, C_4, with the following constraints: i) C_1 can only execute with C_2; and (ii) C_3 cannot execute when C_4 is executing.

Similar to the previous example, the behavior of each component is represented using places and transitions. All constraints on the execution of the components are represented using inhibitor arcs. The inhibitor arcs from place C_2_*idl* to *T1wak* and from place C_1_*wrk* to *T2slp* represent the first constraint. *T1wak* is disabled as long as there is a token in place C_2_*idl* (indicating that C_2 is idle) and C_2 cannot be idle as long as there is a token in place C_1_*wrk* (indicating that C_1 is working). The second constraint is satisfied through the use of two more inhibitor arcs: from place C_4_*wrk* to the *T3wak* transition and from C_3_*wrk* to *T4wak*. The first inhibitor arc causes *T3wak* to be disabled if there is a token in place C_4_*wrk* indicating that C_4 is working. Meanwhile C_4 cannot run if C_3 is working. The resulting reachability graph has fewer states, from a maximum of 16 to 8. The state space, automatically generated using SPNP, is shown graphically in Figure 6.

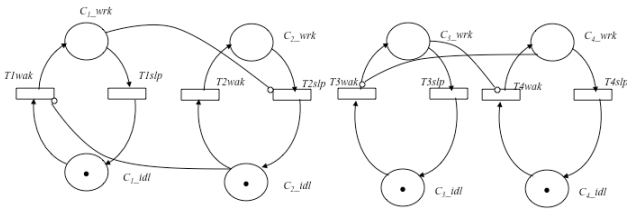


Figure 5 SRN model with execution constraints

V. RELATED RESEARCH

Prevalent research in the area of architecture-based reliability analysis is predominantly focused on sequential applications [1]. A few efforts that consider concurrent applications map the application architecture to a state-space model. Wang et al. [12] presented an approach similar to ours to assess the reliability of an application which follows

the parallel/pipe filter architecture style. Kanoun et al. [13] present a hierarchical approach for a continuously running application. Rodrigues et al. [14] develop a composite solution approach for a terminating application. The analysis methodology described in this paper provides a hierarchical approach for a concurrent, terminating application. Due to its hierarchical nature, the methodology facilitates different types of analyses, which is a significant advantage of this research over the work described by Rodriguez *et al*.[14]. The work closest to ours is by Pettit and Gomaa [15] where a methodology for translating concurrent UML software architectures into an underlying Colored Petri Net (CPN) model is presented. Their work, however, focuses on the performance aspects, unlike ours which focuses on the reliability characteristics.

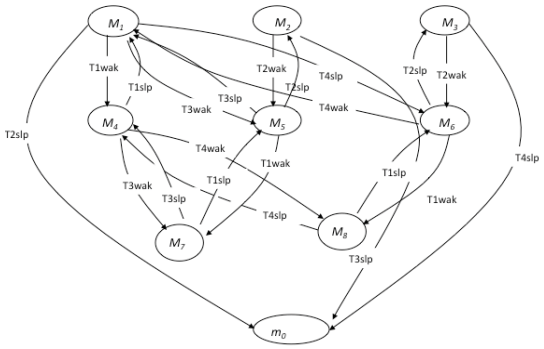


Figure 6 State-space model of architecture with constraints

VI. Conclusions and Future Research

In this paper, we present an efficient approach to analyze the reliability of a software application considering concurrent component execution within the context of its architecture. We demonstrate how Stochastic Reward Nets (SRNs) can be used to effectively and intuitively specify architectural models of large applications, making it possible to apply the approach in practice. We also describe how the expressive power of SRNs could be used to represent constraints on the execution of the application. Through an example, we present how the SRN specification can be used to facilitate sensitivity analysis and identification of bottlenecks. Our future work consists of extending the SRN-based approach to model terminating, concurrent software applications

VII. Acknowlegments

The research at the Univ. of Connecticut was supported in part by a CAREER award from the National Science Foundation (NSF) (#CNS-0643971).

VIII. References

[1] K. Goseva-Popstojanova and K.S. Trivedi,"Architecture–based approach to reliability assessment of software systems," *Performance Evaluation*, vol. **45**(2-3): 578 - 590. 2001.

[2] K. Goseva-Popstojanova, M. Hamill, and R. Perugupalli. "Large empirical case study of architecture-based software reliability." *in IEEE Intl. Symposium on Software Reliability Engineering*, pp. 43– 52, Chicago, IL, Nov 2005.

[3] S. Gokhale,"Software reliability analysis incorporating second order architectural statistics," *Intl. Journal of Reliability, Quality and Safety Engineering*, vol. **12(3)**: 267-290. 2005.

[4] S. Gokhale and K.S. Trivedi,"Analytical models for architecture–based software reliability prediction: A unification framework," *IEEE Trans. on Reliability*, vol. **55**(4): 578-590. 2006.

[5] P. Popic, et al. "Error propagation in the reliability analysis of component based systems." *in Intl. Symposium on Software Reliability Engineering (ISSRE)*, pp.53-62, Chicago, IL, Nov, 2005.

[6] A. Puliafito, M. Telek, and K.S. Trivedi. "The evolution of stochastic Petri nets." *in World Congress on Systems Simulation*, pp.3-15, Singapore, September 1997.

[7] R. El Kharboutly, S. Gokhale, and R. Ammar,"Architecture- based software reliability analysis incorporating concurrency," *The International Journal of Reliability, Quality and Safety Engineering*, vol. **14**(5): 479-499. 2006.

[8] R. El Kharboutly, R. Ammar, and S. Gokhale,"UML-based methodology for reliability analysis of concurrent software applications," *The International Journal of Computers and Their applications*, vol. **14** (4): 250-259. 2007.

[9] C. Hirel, B. Tuffin, and K.S. Trivedi, SPNP: Stochastic Petri Nets. Version 6.0, in *Lecture Notes in Computer Science 1786*. 2000. p. 354-357.

[10] MATLAB version 2.1. 2005, The MathWorks Inc.: Natick, Massachusetts.

[11] G. Ciardo, J. Muppala, and K.S. Trivedi. "SPNP: Stochastic Petri Net Package." *in International Workshop on Petri Nets and Perfomance Models*, pp.142- 151, Kyoto, Japan, December 1989.

[12] W. Wang, Y. Wu, and M.H. Chen. "An architecture-based software reliability model." *in Pacific Rim Dependability Symposium*, Hong Kong, December 1999.

[13] J.-C. Laprie and K. Kanoun, Software reliability and system reliability. M. R. Lyu (editor) Handbook of Software Reliability Engineering. 1997, New York, NY: MCGraw Hill. 27-69.

[14] G. Rodrigues, D. Rosenblum, and S. Uchitel. "Using Scenarios to predict the reliability of concurrent component-Based software systems." *in FASE 2005, M. Cerioli (Eds.)*, pp.111- 127, Springer Verlag, Heidelberg, April 4-8.

[15] R.G. Pettit IV and H. Gomaa. "Improving the reliability of concurrent object-oriented software designs." *in the Ninth IEEE International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS'03)*, pp.262 - 269, Oct 2003.

Ensuring Continuous Data Accuracy in AISEMA Systems

Irina Diana Coman, Alberto Sillitti, Giancarlo Succi
Center for Applied Software Engineering
Free University of Bolzano
Bolzano, Italy
{IrinaDiana.Coman, Alberto.Sillitti, Giancarlo.Succi}@unibz.it

***Abstract* — Automated In-process Software Engineering Measurement and Analysis (AISEMA) systems are powerful tools to monitor and improve the software development process. However, to be useful, it is required that such tools never stop working. Therefore, they need the support of advanced monitoring systems able to detect and locate malfunctions and inform automatically human operators providing all the information required to solve the problem. This paper describe the approach and the tools developed to support a specific AISEMA system developed to support both managers and developers in implementing continuous process improvement initiatives.**

AISEMA; development process; monitoring.

I. INTRODUCTION

The success of software measurement programs is strongly dependant on the automation of the related data collection (Pfleeger, 1993; Daskalantonakis, 1992; Offen and Jeffery, 1997; Hall and Fenton, 1997; Iversen and Mathiassen, 2000). Manual data collection suffers of several limitations including: it is time consuming, tedious, error prone, and often biased or delayed (Johnson and Disney, 1999). Semi-automated data collection is better (tools such as LEAP (Moore, 1999)) but there are still context switching problem (Johnson *et al.*, 2003) with a negative impact on the performance of the developers since it requires to switch continuously between working activities and data collection.

A new generation of tools (such as PROM (Sillitti *et al.*, 2003) and Hackystat (Johnson *et al.*, 2003)) has been developed to overcome such limitations providing a fully automated, non-invasive data collection. Such tools allow data collected from on-going projects to be used for improvement of the same project, therefore they are also called Automated In-process Software Engineering Measurement and Analysis (AISEMA) systems.

AISEMA systems aim at automatically collecting the data, but also at providing tailored analyses for decision support. They reduce the cost of data collection, as they run in the background and let people focus on their work without any additional workload or distractions. They can collect a large variety of data. Based on these data, they propose: support for process management (Remencius *et al.*, 2009; Danovaro *et al.*, 2008), assessment of low-level processes (Coman and Sillitti, 2009), etc.

Ensuring continuous data accuracy is one of the main challenges during the usage of an AISEMA system (Coman *et al.*, 2009). The changes in the environment (such as software updates, software crashes, hardware failures, changes in security policies, etc.) affected sometimes the accuracy of data, mainly by disabling some of the data collection components, hindering data transfer or causing data loss. Not all such events are avoidable. Consequently, small amounts of data might be lost from time to time. However, it is very important to limit as much as possible these missing data and to have detailed information on the cause why they are missing and on their type. Such information helps to assess whether the missing data invalidate or not a specific analysis, thus ensuring reliable results of data analyses.

In most of the cases, existing systems have already error prevention mechanisms located at each of theirs components. This is the case of the PROM system (Sillitti *et al.*, 2003). Such mechanisms ensure a good functioning of the components. However, they cannot prevent, for instance, a silent disable of the components as result of repeated crashes of the host system. Moreover, in some cases, the disabling of the components is perfectly acceptable (for instance, when a developer chooses not to collect some specific data). Thus, to assess whether the disabled status of a component represents a failure of the system or not, additional context information is needed.

In PROM, as the components interact with the server over the network medium, the correct functioning of the system as a whole does not depend only on the correct functioning of each individual component. Because the client components are not aware of their broader context (and are not meant to be aware), there is a need for a separate component that monitors the functioning of the system as a whole. Such component should identify potential problems and use local information from specific components to localize the actual cause of the problem. Additionally, such component should log the occurrence of the problem and, if the solution is known, to proceed with solving the problem. If the solution is unknown, the component should notify the maintainers of the system.

The initial solution used during most of the case-study was to have a human performing such monitoring. However, this is

extremely time consuming and costly. Moreover, ideally, the monitoring should be continuous and thus requires an automated solution. The solution developed was to enhance the existing AISEMA system with characteristics of autonomic computing (Horn, 2001; Kephart and Chess, 2003) such as self-monitoring and self-healing. This paper presents in detail the concrete approach and its implementation. It is organized as follows: Section 2 presents the related work; Section 3 introduces our approach; Section 4 describes the proposed implementation; Section 5 summarizes the results obtained; finally, Section 6 draws the conclusions and presents future work.

II. Related Work

The initial manifesto of autonomic computing (Horn, 2001) proposed it as the single approach able to mitigate the effects of an increasingly acute software complexity crisis. The manifesto pointed out that the complexity of the IT infrastructure grows constantly and threatens to get beyond human ability to manage it. Continuously increasing number of connections, dependencies and interactions between software components make the maintenance and management of the software systems increasingly complex, up to the point of surpassing human ability of managing these tasks. Thus, the only solution resides in building software systems that are able to configure and manage themselves, adapting to changing environments and adjusting their behavior for most efficient use of their resources to solve the required tasks.

Autonomic computing identifies four key aspects that self-managing software should possess: self-configuration, self-optimization, self-healing, and self-protection. Of these four, the self-healing is the main aspect that AISEMA systems should have to ensure continuous data accuracy.

Self-healing is defined as the capability of a software system to automatically detect, diagnose, and repair localized problems resulting from bugs or failures in software and hardware (Kephart and Chess, 2003). Koopman (2003) proposes a taxonomy that identifies four main aspects of a self-healing strategy: fault model, system response, system completeness, and design context.

The fault model (also called a fault hypothesis) identifies the faults that the system should be able to tolerate. The system response represents the strategy of the system to detect and treat faults. The system completeness and design context include the factors that influence the scope of self-healing capabilities.

Each of these aspects can be further described by a set of properties. Table I gives an overview of the characteristics of our approach (PROM) in terms of the aspects and properties defined in the taxonomy of Koopman.

TABLE I. Characteristics of the approach in terms of a taxonomy for self-healing problem spaces

Aspect	Property	PROM
Fault model	Fault duration	Permanent
	Fault manifestation	Corrupted or missing data
	Fault source	Changes in the environment, non-malicious
	Granularity	Component
	Fault profile expectations	Expected and likely faults
System response	Fault detection	Anomaly detection via supervisory checks and nonintrusive testing of results
	Degradation	Degraded performance
	Fault response	Reactive
	Recovery	Warm or cold component reboot, recovery of corrupted settings
	Time constants	Valid data occur much more often than anomalies
	Assurance	"Good enough" data quality
System completeness	Architecture completeness	Closed, complete system
	Designer Knowledge	Complete
	System Self-knowledge	Partial knowledge on component presence
	System evolution	Upgrades and extensions
Design context	Abstraction level	Software component within distributed system
	Component homogeneity	Heterogeneous components and resources
	Behavioral predetermination	Predetermined data type
	User involvement	Fully automatic
	System linearity	N/A
	System scope	Multiple computers and users on Internet or closed network system

Shehory (2006) identifies three main types of problems that self-healing approaches usually address: concurrency problems, functional problems, and performance problems. The approach proposed here addresses functional problems.

There are already many projects exploring various self-healing strategies (Hoover *et al.*, 2001; Raz *et al.*, 2002a; Shelton *et al.*, 2003; Shehory, 2006; Mikic-Rakic *et al.*, 2002). The proposed approaches focus on reconfigurable architecture (Shehory, 2006; Mikic-Rakic *et al.*, 2002), graceful degradation (Shelton *et al.*, 2003), optimized usage of resources (Hoover *et al.*, 2001), or inferring specifications from underspecified components (Raz *et al.*, 2002a). Most of the proposed approaches focus on distributed systems and address software failures in one or more of the components. By contrast, the approach proposed here focuses on failures due to environment factors such as disable of components or loss of connectivity. The proposed approach makes also use of the specificities of the domain, namely the characteristics of the automatically collected data.

Among the existing self-healing approaches, one that is very similar to the approach presented here is that of Raz *et al.* (2002a). They focus also on detecting failures by searching for semantic anomalies in the data. In their case, the data are dynamic data feeds that change outside user control and are usually under-specified. Their approach has two phases: *setup*

and *usage*. During *setup*, the approach aims at inferring invariants from data presented as input. These invariants are used during the *usage* phase to detect anomalies in the data. During the *setup* phase, human intervention is required to find a training set size, establish parameter values, select attributes to take into account, and select from the proposed invariants. During the *usage* phase, human intervention is also desired to eliminate from the results the false positives (normal data falsely declared as anomalous).

Similarly to the approach of Raz *et al.,* the approach proposed here also detects failures by searching for anomalies in the data. However, the domain is very different and it allows to take one step further by taking action to correct faults based on the detected anomalies.

By contrast to the approach of Raz *et al.*, the specifications of the data are very well known as the data come from the components of the system itself. Thus, data invariants can be defined by the administrators of the system, together with the possible causes and steps to take for restoring proper functioning. Moreover, the definition of data invariants can also take advantage of overlapping data coming from different sources. Once the invariants have been defined, the system can perform fully automatically the cycle of fault detection, cause identification, and fault resolution. Human intervention might be required again only in cases when none of the proposed solutions solve the problem.

Another contribution of the approach proposed here is a model for defining data invariants together with potential causes of violation and actions to take for resolution of the violation. The model also allows the combination of basic invariants to easily specify more complex ones while hiding the complexity of the definition. While the current implementation expects predefined data invariants, the approach can be easily extended with automated inference of data invariants.

III. THE APPROACH

The main goal is to ensure continuous high accuracy of automatically collected data. The approach proposed here is to perform regularly a set of checks on the collected data to detect anomalies and then to take action to prevent further anomalous data. Anomalies in the data indicate a problem in the functioning of the system. When anomalous data are detected, the system investigates several possible causes to localize the actual problem. To do so, it performs a set of tests regarding the status of the network and of the components installed on the client machine from which data originated. After identifying the cause(s), the system proceeds to heal itself. Depending on the actual cause, the actions taken can consist in warm or cold reboot of components on the client machine, restoring corrupted files or notifying the user that user action is needed. Moreover, the system logs all checks and tests performed and their results to provide a complete view on the status of the data at any given time.

An anomaly is defined as a violation of one or more data invariants. A data invariant expresses a condition that all accurate data should satisfy. The definition of data invariants is based on domain knowledge. As the collected data are well defined and the system specifications are known, the easiest approach is to use such knowledge for data invariant definition.

Thus, simple data invariants can define the domain of values for each type of data collected (for instance, the time spent in some activity should be positive, and below 24 hours during a single day), the required fields (according to the known data structure), or the frequency with which they are collected (according to the specifications of the system).

The above type of invariants describes mainly the structural properties of data, rather than the semantic ones. The violation of such an invariant can help identify severe problems (such as no network connection or hardware failure) but do not reveal more subtle issues (such as data from one component with legal but incorrect values). Additional data invariants that define relations in the data can help detect also semantic anomalies. Such invariants are usually called adaptive invariants to distinguish them from invariants that impose certain specific values rather than relations.

The automatically collected data come from various components. As such, the data usually overlap to some degree. Data invariants that make use of these overlaps are used to identify anomalies in data coming from one component. Moreover, as different data represent different views on the same artifact, there are some relations in the data that should be respected. Such relations that should hold at any time are usually called *adaptive invariants*. The violation of an adaptive invariant means that the data are not semantically consistent.

A. The Data Invariant Model

The definition of an invariant contains the data property or the data relation that has to be ensured, a description of the problem to be reported if the invariant is violated, and a list of the potential causes of the identified problem. The actual computation of a data invariant can be a direct check of a data property, or it can build on the results obtained on checking several other data invariants. This gives particular strength to the model as it allows the definition of extremely complex invariants while making their definition easy by hiding their computational complexity from the user.

The self-healing ability requires not only that anomalous data are detected, but also that the cause of anomaly is removed. There is not a one-to-one relation between anomalies and causes. The violation of a data invariant can be due to one or more of the causes listed for that invariant, or even to a new cause, not yet identified. Thus, each cause is defined together with a set of tests that should fail only when the cause is present. Such tests usually concern local conditions on the machine which generated anomalous data. However, the only limits to what such tests check depend only on the possibility of actually performing the test.

The various tests associated with a potential cause is useful not only to identify the presence of a cause, but also to distinguish between malign and benign manifestations of the same cause. The benign manifestation of a cause means a false-positive in the sense that the violation of the adaptive invariant is not due to a problem in the functioning of the system, but rather to special circumstances (for instance no upload of data

due to user being on a sick leave). The malign manifestation of a cause represents a real failure of the system, one of its components or the network medium.

Finally, each test has to define also a list of actions to take if the test fails. Such actions should remove the cause of anomalous data and should restore the proper functioning of the system.

Thus, an invariant's definition should contain such information as shown in Table II. Table III shows an example of a simple data invariant and Table IV provides an example of the definition of one of the possible causes of violation of such an invariant.

TABLE II. STRUCTURE OF A DATA INVARIANT

Field	Meaning	Intended user
Name	Short, descriptive name for the data invariant. Helps in reports for administrators.	Human
Description	A more detailed description of the invariant and, if needed, the reasons justifying it.	Human
Computation	The actual function or formula that returns true is the invariant is satisfied and false if it is violated.	System
Problem reported if the invariant is violated	A description of the problem identified by a violation of the invariant.	Human
List of possible cause(s)	A list with the possible causes of the invariant's violation	System

TABLE III. EXAMPLE OF A DATA INVARIANT'S DEFINITION

Name	Data upload
Description	For each user, there should be at least one upload of data each day.
Computation	AnyUploadCheck (method)
Problem	NO uploads during the day.
Possible causes	1. AISEMA system inactive on user's machine. 2. Transfer scheduler NOT running. 3. Corrupted configuration of connection on user's machine. 4. Server component down. 5. Server unreachable from client machine.

TABLE IV. EXAMPLE OF A POSSIBLE CAUSE'S DEFINITION

Cause	Test	Type	Action to take if test fails
AISEMA system inactive on user's machine	Check flag of system activity to be set to active.	Benign	Just annotate that the user has deactivated the system on his/her machine.
	Check that data are collected on user's machine (there are files stored).	Malign	Warm reboot of the client components.

IV. IMPLEMENTATION

The approach is implemented in two components: PROM Data Inspector (server-side component) and PROM Console (client-side component). Both components are written in Java. PROM Data Inspector implements the actual problem detection and takes steps to solve the identified issues. PROM Console provides additional information on the status of the clients when needed and it propagates to the client the actions recommended by PROM Data Inspector to solve the identified problems. Figure 1 shows the relations between the existing architecture of PROM and these two components.

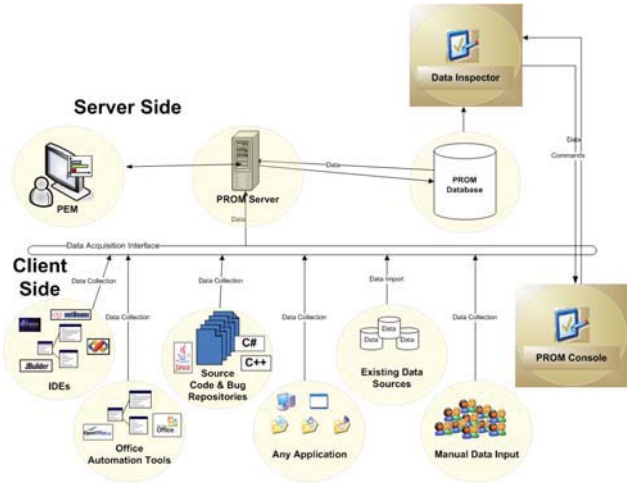


Figure 1. PROM Architecture and the relationship with PROM Data Inspector and PROM Console

The definitions of data invariants are stored in the PROM database. At present, the implementation of the formula of the invariant is in a separate module of the Data Inspector. However, PROM Data Inspector can be easily extended to use formulas implemented in external modules. This makes the actual structure of the data storage transparent to the Data Inspector, ensuring that the self-healing mechanism is independent from the way in which data are stored.

Data invariants can be of two types: basic and summary invariants. The basic invariants have a simple, single data property to check. They are usually meant for testing fixed data invariants rather than adaptive invariants that require more complex checks of data relations. For such checks, the summary invariants are more appropriate. A summary invariant has a computation formula that usually involves other basic invariants. This allows to check for complex data relations. Moreover, if desired, several distinct invariants can be defined to allow a more precise identification of the actual cause of violation.

PROM Data Inspector runs at regular intervals, loads the invariants from the database and performs a check of all defined data invariants. For any invariant that is violated, the component goes through the list of possible causes and runs the associated tests. For every failed test, the corresponding action is taken and its result is stored. If the execution of the action fails, an error notification is sent to the administrators of the system. After successful execution of an action, the system performs again the test that had previously failed. However, the system has to wait until the next round of overall check of data invariants to be sure that it solved the problem that was actually causing anomalous data. Thus, the overall algorithm has the following steps:

1. Check all data invariants and store results.

2. For each data invariant that was violated:
    a. For each possible cause:
        i. Perform all associated tests
            1. If a test fails, run the corresponding action;
            2. If the action completed successfully, perform again the test;
            3. If the action could not be run or the test failed again after action has been taken report to the administrators.
    b. If all tests of all possible causes passed from the first run (i.e., no cause has been identified as responsible for the observed violation), notify the administrators.

The PROM Data Inspector stores all the results of the check of invariants in the database for future reference, together with a timestamp. Optionally, it can also generate a report and send it by email to the administrators. The detailed results of the checks performed can be used for having a clear view on the status of the data at any given time. The Data Inspector ensures the compatibility with the graphical interface PEM (PROM Experience Manager), so that the status and results of all tests can be displayed in a chart allowing for fast identification of days and users with problematic data.

The PROM Console resides on the client machine and communicates with the Data Inspector on the server. It gathers information about the local context, such as the PROM components installed, the state of their configuration files and the data that are collected. It sends such data to the Data Inspector which uses them to perform the tests for cause identification. Upon request from the Data Inspector, PROM Console notifies the user that some specific action needs to be taken, or it implements directly non-invasive actions such as restoring of corrupted configuration files.

V. Results

PROM Data Inspector and PROM Console were in usage during the last 2 months of data collection in the an experiment in a company (Coman *et al.*, 2009). In a first step, Data Inspector represented mainly an automation of the data checks that were previously performed manually. However, it soon became obvious that it can also perform more complex checks that are very tedious to be performed manually. Thus, the summary invariants were introduced, allowing the combination of many basic invariants.

The initial reports of the PROM Data Inspector were just listing all the tests performed together with the results obtained. However, such reports were hard to read without knowing the inner functioning of Data Inspector. The current version addressed this problem by providing at the beginning summaries of the identified problems, together with the possible causes. This new form of reports proved to be understandable also by people that were not directly involved in the definition of data invariants or in the development of the Data Inspector. Figure 2 shows an example of the contents of a report generated for a single user showing an identified problem and listing the possible causes. In the report, each test represents in fact a data invariant. This change of name makes the report more intuitive to people and does not require users to know what a data invariant is.

```
********************Summary********************
1. Problem: NO uploads during the day for users:
    John (60)

   Possible causes:
    PROM inactive on user's machine;
    Transfer Scheduler NOT running;
    PROM server down or unreachable;
    wrong configuration of the Transfer on the user's machine

********************Details********************
FAILED TESTS:

John (60):CheckCleanedEffort:0<=0
John (60):CheckPlugins:Trace=NO;VSPlugin=NO;devenv.exe=NO
John (60):CheckUploads:last upload:NONE;before last upload:NONE
John (60):CheckStoryPoints:source1={} and source2={12452=8}

SUCCEEDED TESTS:
John (60):CheckUncleanedEffort:0<10800
John (60):CheckCleanedCompileTime:compile_time=0 and VS_effort<=0

TESTS NOT RUN:

NONE
```

Figure 2. Example of PROM Data Inspector report showing a detected data anomaly

VI. Conclusions and Future Work

This paper reported on the first step towards transforming AISEMA systems into autonomous systems, by ensuring self-healing capabilities. The proposed approach is to detect data anomalies and to trace them back to the software or hardware failure that caused them. The data anomalies are modeled as violations of data invariants. The data invariants are based on the knowledge of the system and of the data collected. They can be very simple (e.g., time recorded should always be positive) or more complex, combining several basic invariants for testing of relations into the data.

To solve the issues that cause data anomalies, each data invariant contains also a list of possible causes. Each cause has a list of tests that help identify the exact problem and whether it is benign (not a failure but special circumstances) or malign (a failure that has to be solved). For each test there is also a list of actions to be taken when the test fails. This model ensures flexibility (new invariants can be added at any time), simplicity (the complexity of invariants can be hidden from the user by building on underlying levels of invariants), and traceability of issues that affected the collected data (all actions are clearly defined and the results are always stored).

The current implementation uses only user-defined invariants. As future work, it would be valuable to explore the possibilities of automatically extracting invariants from previously collected data. Techniques such as the one used in Daikon (Ernst *et al.*, 2001) or Mean (Raz *et al.*, 2002b) have already been applied to automated detection of data invariants in online data feeds. However, applying such techniques is not

straight forward, as the characteristics of the AISEMA data are quite different from those of online data feeds (i.e., most of the data are not normally distributed, the volume of data is bigger, complex relations between various types of data have to be taken into account).

To preserve the character of self-healing of the proposed approach, an automated detection of data invariants should also be complemented by an automated detection of possible causes and of appropriate measures to be taken for each identified cause. However, this is still an open issue.

REFERENCES

[1] Coman, I., and Sillitti, A. *Automated Segmentation of Development Sessions into Task-related Subsections*, International Journal of Computers and Applications, ACTA Press, 31(3), 2009.

[2] Coman, I., Sillitti, A., and Succi, G. *A Case-study on Using an Automated In-process Software Engineering Measurement and Analysis System in an Industrial Environment*, ICSE 2009, Vancouver, BC, Canada, 16 - 24 May 2009.

[3] Danovaro, E., Remencius, T., Sillitti, A., and Succi, G. *PEM: Experience Management Tool for Software Companies*, 22nd Object-Oriented Programming, Systems, Languages & Applications (OOPSLA 2008), Nashville, TN, USA, 19 - 23 October 2008.

[4] Daskalantonakis, M., *A Practical View of Software Measurement and Implementation Experiences Within Motorola*, IEEE Transactions on Software Engineering, Vol. 18, 1992.

[5] Ernst, M. D., Cockrell, J., Griswold, W. G., and Notkin, D., *Dynamically Discovering Likely Program Invariants to Support Program Evolution,* IEEE TSE, 27(2), 2001.

[6] Hall, T., and Fenton, N. *Implementing Effective Software Metrics Programs,* IEEE Software, March/April 1997.

[7] Hoover, C. L., Hansen, J., Koopman, P., and Tamboli, S., *The Amaranth Framework: policy-based quality of service management for high-assurance computing*, Intl. Journal of Reliability, Quality and Safety Engineering, 8(4), 2001.

[8] Horn, P., *Autonomic Computing: IBM's Perspective on the State of Information Technology*, IBM Corporation, available at http://www.research.ibm.com/autonomic/manifesto/autonomic_computing.pdf, 2001.

[9] Iversen, J., and Mathiassen, L., *Lessons from Implementing a Software Metrics Program*. Hawaii Intl. Conf. on System Sciences, 2000.

[10] Johnson, P. M., and Disney, A., *A Critical Analysis of PSP Data Quality: Results from a Case Study.* Journal of Empirical Software Engineering, 4(4), 1999.

[11] Johnson, P. M., Kou, H., Agustin, J., Chan, C., Moore, C., Miglani, J., Zhen, S., and Doane, W. E. J., *Beyond the Personal Software Process: Metrics Collection and Analysis for the Differently Disciplined*. ICSE, 2003.

[12] Kephart, J. O., and Chess, D. M., *The Vision of Autonomic Computing,* IEEE Computer, 36 (1), 2003.

[13] Koopman, P., *Elements of the Self-Healing System Problem Space,* ICSE WADS, 2003.

[14] Mikic-Rakic, M., Mehta, N., and Medvidovic, N., *Architectural Style Requirements for Self-Healing Systems*, Intl. Workshop on Self-healing Systems, ACM Press, 2002.

[15] Moore, C. A., *Project LEAP: Personal Process Improvement for the Differently Disciplined,* ICSE, 1999.

[16] Offen, R. J., and Jeffery, R., *Establishing Software Measurement Programs*, IEEE Software, March/April 1997.

[17] Pfleeger, S. L., *Lessons Learned in Building a Corporate Metrics Program*, IEEE Software, May/June 1993.

[18] Raz, O., Koopman, P., and Shaw, M., *Enabling Automatic Adaptation in Systems with Under-Specified Elements,* Intl. Workshop on Self-Healing Systems, 2002.

[19] Raz, O., Koopman, P., and Shaw, M., *Semantic Anomaly Detection in Online Data Sources*, ICSE 2002.

[20] Remencius, T., Sillitti, A., and Succi, G. *Using Metrics Visualization and Sharing Tool to Drive Agile-Related Practices*", 10th International Conference on eXtreme Programming and Agile Processes in Software Engineering (XP 2009), Pula, Italy, 25 - 29 May 2009.

[21] Shehory, O. *A Self-healing Approach to Designing and Deploying Complex, Distributed and Concurrent Software Systems*, Intl. Workshop on Programming Multi-Agent Systems, ProMAS, 2006.

[22] Shelton, C., Koopman, P., and Nace, W., *A framework for scalable analysis and design of system-wide graceful degradation in distributed embedded systems,* Intl. Workshop on Object-Oriented Real-Time Dependable Systems, 2003.

[23] Sillitti, A., Janes, A. Succi, G., and Vernazza, T., *Collecting, Integrating and Analyzing Software Metrics and Personal Software Process Data* EUROMICRO, 2003.

Specification and Implementation of SPEM4MDE, a metamodel for MDE software processes

Samba Diaw, Redouane Lbath, Bernard Coulette
University of Toulouse,
Laboratory of IRIT
Toulouse, France
{diaw, lbath, coulette}@irit.fr

Abstract— It is widely accepted that MDE is a novel and important approach for building complex systems. This fact forces many organizations to transform their software development processes into model-driven ones. While model-driven development processes – called MDE processes – have started to appear, a tool-supported Process Modeling Language (PML) for describing, enacting and evolving such processes is still lacking. In this article, we propose a PML that combines SPEM 2.0, UML 2.2 and MOF 2.0 QVT concepts. We extend these concepts for the MDE domain and define the behavior of MDE process models in terms of UML state-machines. The result is a language that can be used to define a MDE process, and thanks to its behavioral semantics, to guide the execution of projects based on such a process. To evaluate and validate our approach, we have developed a prototype under the TOPCASED environment.

Keywords: Model-Driven Engineering (MDE), Model-Driven Architecture (MDA), Model Transformations, Process Modeling Language (PML), Model-Driven Development (MDD), MDE Software Process, TOPCASED environment

I. INTRODUCTION

MDE (Model-Driven Engineering) ([1], [2], [21]) is a recent software engineering discipline that advocates the use of models within the software development rather than code. The term MDE was proposed first by Kent in [19] and is derived from the OMG's Model Driven Architecture (MDA) initiative [3], [15]. MDE aims at improving significantly the development of complex software systems by providing the means to switch from one abstraction level to another or from one modeling space to another [2], [21]. The finality is to describe both the problem and its solution by using models, and by clearly establishing a methodology to show how to switch from a problem to its solution by using model transformations [2]. MDE is then a form of generative engineering in the sense that the whole or part of an application is generated from models. In this new perspective, models take an important place among software development artifacts and must be formal in order to be understood or transformed by tools. In [5], Kleppe and al. define a model-driven software development as "*a process of developing software using different models on different levels of abstraction with (automated) transformations between these models*". Therefore, the model-driven software development process [7], [25], [26], [28] – called MDE software process – may be seen as a transformation chain, each transformation consuming one or several source models and producing one or several target models. The description of a software process is called *process model*. It can be expressed through any specific language or notation which is called Process Modeling Language (PML). A process model can be enacted when a development team follows the process model during the development life cycle. One of the major advantages of software process modeling is to help developers using a unified and consistent terminology in order to communicate around the process. Software process modeling should also make possible understanding, reuse, evolution, management, and standardization of processes [27].

Our general research goal is to propose a tool-supported PML to model and enact MDE software processes. However, as stated in related work (section V), there is no PML meeting this goal so far.

The work presented in this paper focuses on modeling and enactment of MDE software processes. More precisely, our goal is to specify and implement a metamodel called SPEM4MDE [22] which is a generic and flexible language that combines SPEM 2.0 [12], UML 2.2 [13] and MOF 2.0 QVT [14] concepts. A first version of the SPEM4MDE metamodel was described in [22]. Since this first work, we have been enriching our metamodel by taking into account the QVT standard for transformations execution and the UML 2.2 BehaviorStateMachines package for describing MDE processes behavior. In addition, we have developed a prototype called SPEM4MDE-PSEE which is a CASE tool supporting MDE process models edition and enactment.

The paper is organized as follows: section II introduces the motivation and objectives of this work. Section III presents the specification of the SPEM4MDE metamodel. Section IV presents the validation of SPEM4MDE through a prototype and a MDE process example which we modeled and enacted using this prototype. Section V discusses related works, while section VI concludes and introduces perspectives.

II. MOTIVATION AND OBJECTIVES

With the emergence of MDE, many organizations have been starting to transform their traditional software development processes into model-driven processes [28]. The underlying objective is to reduce time of development, and to increase software productivity and quality. As established in

[27] for traditional software development, the specification of MDE processes should guide developers in the elaboration and the transformation of models.

Our main objective is to provide the MDE community with a metamodel to define and enact MDE software processes. As defined in [5], a MDE process is a process of developing software using different models and transformations between these models.

The concepts of SPEM 2.0 are quite generic since they are supposed to be able to describe any kind of software (and even system) process including MDE ones. However, SPEM 2.0 concepts are not able to capture the exact nature of most activities and artifacts of a model-driven development. Indeed, most of model-driven development activities are model transformations, and most of model-driven development artifacts are models.

We claim that reification of MDE concepts allows process designers to explicitly describe specific aspects of MDE development. For instance, to make it possible to check the conformance relationship, it is important to specify models and metamodels as input/output parameters of transformations. So we advocate promoting MDE concepts as first-class citizens in MDE process technology.

Therefore, we have decided *to extend the SPEM 2.0 metamodel for MDE domain*. In addition, SPEM 2.0 does not fulfill executability. To overcome this limitation, we reuse the UML state-machines for describing behavior of MDE processes. We also allow the specification of transformations' behavior with QVT in order to benefit from existing tools that implement this standard.

III. SPECIFICATION OF THE SPEM4MDE METAMODEL

SPEM4MDE is formally specified as a metamodel that reuses subsets of the concepts defined by the OMG's standards SPEM 2.0 [12], UML 2.2 [13], and QVT [14]. A first version of SPEM4MDE was published in [22]. Since this publication, we have reorganized and enriched the metamodel, and developed an operational prototype for MDE process modeling and enactment. More precisely, the metamodel has been enriched by adding a new package for describing model relationships and traceability at enactment time, and integrating the QVT concepts in the behavioral package. In the following, we describe the new version of SPEM4MDE. The prototype is presented in section IV.

As shown by Fig. 1, SPEM4MDE is structured in the form of three packages: *MDE Process Structure, Model Relationship,* and *MDE Process Behavior*.

The *MDE Process Structure* package outlines concepts that describe SPEM activities and transformations. It merges the SPEM 2.0 *Process structure* package in order to describe a transformation as a specialization of SPEM *Activity*. The *SPEM 2.0 Process structure* package describes a process in terms of breakdown elements, which are activities, input and output products, and roles that perform activities.

The *Model Relationship* package describes relationships between models. It imports some concepts from *MDE Process Structure*.

The *MDE Process Behavior* package reuses the *UML 2.2 BehaviorStateMachines* package and merges the *MDE Process Structure* package in order to describe the behavior of MDE process elements by means of UML state-machines. In addition, it merges the *QVTBase* package in order to describe the execution semantics of transformations. The *QVTBase* package contains a set of basic concepts that describe transformations, their rules, and their input and output models.

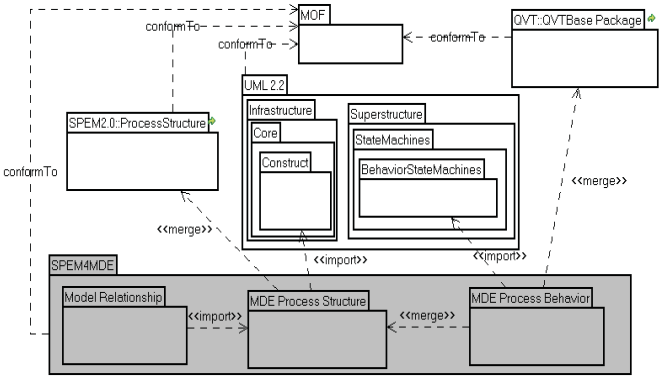


Figure 1. SPEM4MDE packages hierarchy

A. MDE Process Structure Package

Fig. 2 shows the MDE Process Structure package. The objective of this package is to describe the structure of a MDE process that encompasses traditional activities (e.g. edit a model) and model transformations. Activities are described by their input and output products, roles that perform activities, and constraints that specify their precondition, invariant, and postcondition. *InitialActivity* is a pseudo-activity that starts the process while *FinalActivity* is a pseudo-activity that stops the process.

A *Model* is a description of (part of) a system written in a well-defined language (i.e. a model must conform to a metamodel) [5]. The rules WF01, WF02, and WF03 described below, introduce restriction between a model and its compliance model. A model at the *M1* level should conform only to a metamodel (i.e. a model at the *M2* level); a metamodel should conform only to a meta-metamodel (i.e. a model at the *M3* level); and a meta-metamodel should conform only to itself. A model may not have a level (e.g. UML Infrastructure has not a fixed level).

A *TransformationDefinition* is considered as a particular *SPEM Activity*. It is described by its informal rules, its input and output models, and its source and target metamodels. Informal rules describe relationships between constructs of a source metamodel and constructs of a target metamodel. A *TransformationDefinition* is endogenous (i.e. its source metamodel and its target metamodel are identical) or exogenous (i.e. its source metamodel and its target metamodel are different). As a SPEM Activity, a *TransformationDefinition* is performed by roles, specified by *RoleUse* via *ProcessPerformer*. However, since a transformation in

SPEM4MDE is to be executed automatically, a *TransformationDefinition* should have only one *ProcessPerformer* linked to only one *RoleUse*.

The following constraint rules are used to define the static semantics of the MDE Process Structure package. Due to space limitation, the OCL expression is shown for only the first three rules.

- [WF01]: A *Model* at the *M1* level must conform to a Metamodel (i.e. a model at the *M2* level).

  Context Model inv: (self.level=#M1 and self.complianceModel→notEmpty()) implies (self.complianceModel.level= #M2)

- [WF02]: A *Metamodel* must conform to a *Meta-Metamodel (i.e. a model at the M3 level)*

  Context Model inv: (self.level=#M2 and self.complianceModel→notEmpty()) implies (self.complianceModel.level= #M3)

- [WF03]: A *Meta-Metamodel* must conform to itself.

  Context Model inv: (self.level=#M3 and self.complianceModel→notEmpty()) implies (self.complianceModel=self)

- [WF04]: Input and output models of a *TransformationDefinition* should have the same level; source and target metamodels of a *TransformationDefinition* should have the same level; source and target meta-metamodels of a *TransformationDefinition* should have the same level.

- [WF05]: An *Activity* may be decomposed only into *Activities* or *Transformations.*

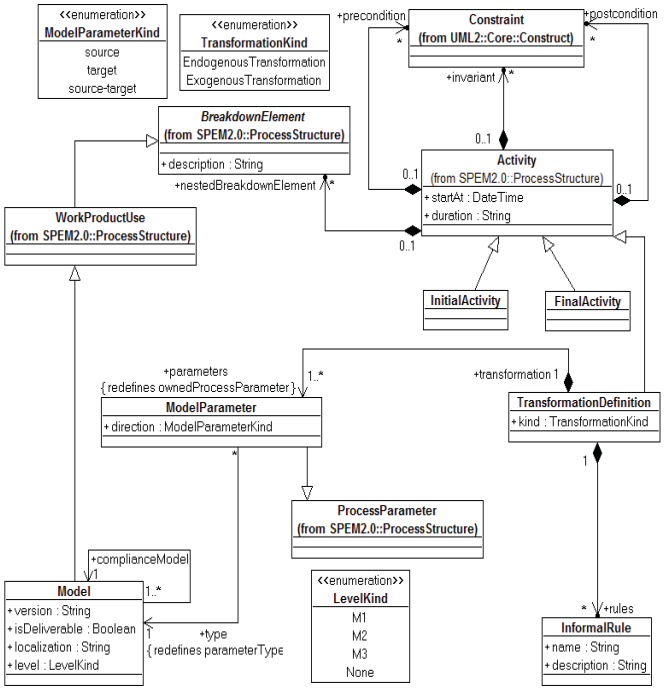


Figure 2. MDE Process Structure Package

B. *Model Relationship Package*

Fig. *3* shows the Model Relationship pac*kage.* The objective of this package is to describe common relationships between models, which are composition, and refinement. A model may be decomposed into other models (e.g. decomposition of UML metamodel into Class Diagram metamodel, Use case metamodel, etc). A model may be the refinement of another model (e.g. PIM to PSM in MDA approach). The concept *Trace* represents a set of trace elements that together describe traceability links between source and target model elements in a *TransformationDefinition*.

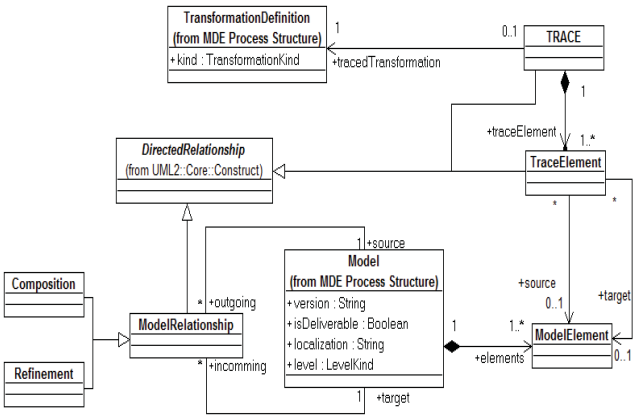


Figure 3. Model Relationship Package

C. *MDE Process Behavior Package*

To allow computer-assisted enactment of MDE processes, it is necessary to be able to model their behavior. The goal of the *MDE Process Behavior* package is to provide concepts for implementing transformations defined in *MDE Process structure* and concepts for defining behavioral models of MDE process elements. Fig. 4 shows the MDE Process behavior package.

The concept *TransformationImpl* describes the implementation of a transformation in a formalism which is either rule-based, program-based, or template-based. A *TransformationImpl* is executed by a required MDE tool. *HumanActor* specifies a person who plays his assigned roles within the process.

Transformation with QVT meaning is a rule-based transformation that extends *TransformationImpl*. To help transformation designers describe transformations with rules, we advocate the use of QVT.

The following constraint rules define the static semantics of the MDE Process Behavior package. The OCL expression is given for only the first rule.

- [WF01]: Input models of *TransformationImpl* must conform to the source metamodels of its associated transformation specification.

Context TransformationImpl inv: (self.parameters→select(i\|i.direction=ParameterKind.in)→ collect(t\|t.model.complianceModel))=

(self.specification.parameters→select(m|m.direction= ModelParameterKind.source) →collect (t| t.type))

- [WF02]: Output models of *TransformationImpl* must conform to the target metamodels of its associated transformation specification.

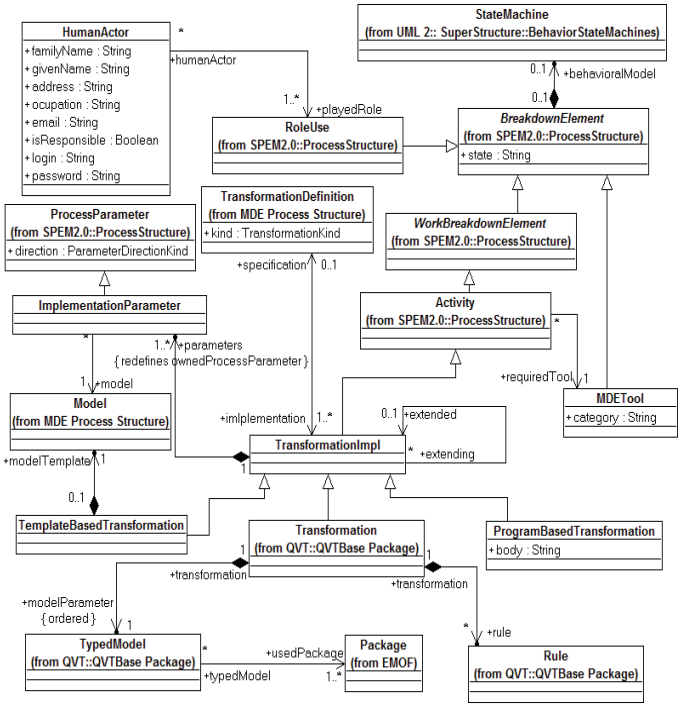


Figure 4. MDE Process Behavior Package

The behavior of a *BreakdownElement* may be described by a UML state-machine. The property *state* in BreakdownElement defines the state of each MDE process element at enactment time.

To illustrate the use of UML state-machines in MDE Process Behavior package, Fig. 5 shows a default behavior of any transformation. This behavior may be reused or adapted when defining a MDE process. We distinguish between three composite states: *Not-Running*, *Running*, and *Finished*. After the process has been instantiated, the transformation is in the state E*xecutable*. The transition from the state *Executable* to the state *Startable* occurs when the function *Startable ()* returns true (i.e. precondition and precedence constraints are fulfilled). The transformation switches from the state *Startable* to the state *Executing* when the required MDE tool is running. A transition to the state *Inconsistent* occurs when the invariant associated to the transformation is no longer fulfilled. From *Executing* state, the transition to the state T*erminated* is triggered when the execution process of the transformation is finished. From *Executing* state, the transition to the state *Aborted* happens when the developer cancels the transformation execution. From the state *Inconsistent*, there are four options: *reconcile* the transformation with the MDE process and then it switches to the state *Executing*, *cancel* the transformation execution and then it switches to the state *Aborted, finish* the transformation execution and then it switches to the state *Invalidated*, or *restart* the required MDE tool and then the transformation switches to the state *Executable*. From the state *Terminated*, if the transformation is validated (i.e. its target models are validated and its postcondition is fulfilled), the transformation switches to the state *Validated*, otherwise it switches to the state *Invalidated*. From the state *Invalidated* the user may restart the required MDE tool and then the transformation switches again to the state *Executable*.

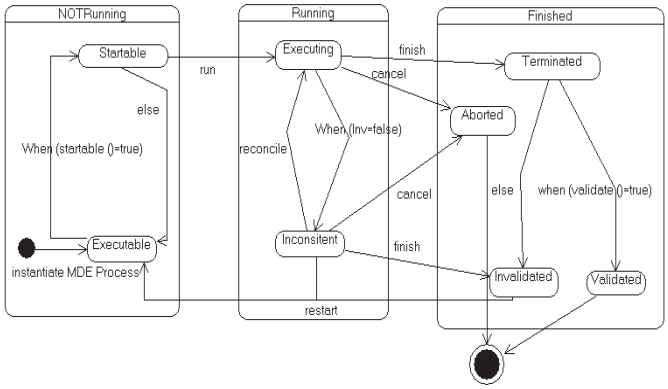


Figure 5. Default Behavior of a Transformation

IV. VALIDATION

A. *The prototype SPEM4MDE-PSEE*

To validate the specification of the SPEM4MDE metamodel, and allow users to define and enact SPEM4MDE process models, we have developed a prototype SPEM4MDE-PSEE under the TOPCASED environment [18]. As shown by Fig. 6, SPEM4MDE-PSEE is divided into two components: *SPEM4MDE Process Editor*, and *SPEM4MDE Process Enactment Engine*.

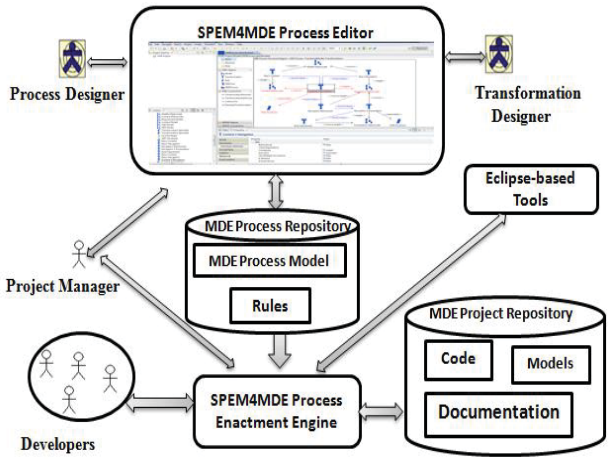


Figure 6. Architecture of SPEM4MDE-PSEE

SPEM4MDE Process Editor allows process designers to describe, and modify MDE process models. Describing a MDE process model includes describing its structure and its behavior. Transformation rules in a MDE process are described by transformation designers. Once the MDE process model is described, the process designer may check it with respect to the

constraints defined in the SPEM4MDE metamodel, or to additional constraints. There are two ways for checking MDE process models: checking on demand (i.e. when the user triggers himself the checking process) or checking during edition (i.e. checking is done automatically by the tool). Outcomes of process editing are stored in a repository called *MDE Process Repository*. This editing component may also be used by a project manager for adapting a MDE process model to a given project.

SPEM4MDE Process Enactment Engine allows developers to enact a project-specific process model by allowing them at each time to execute enactment operators and to know the sate of any MDE process element. It is integrated with other eclipse-based tools (ATL, Smart QVT, etc.) in order to execute transformations. Developers can then keep track of what is the current state of each element of the MDE project, what has been done before and what is left. Outcomes are models, code, documentation, etc. and are stored in a MDE *Project Repository*

B. *Case Study: A process for developing web systems*

In this section we illustrate our approach with a MDE process example: UWE (UML-Based Web Engineering) [8], [9]. We first give an overview of the UWE process. Then, we show the description of an extract of UWE with the *SPEM4MDE Process Editor*. Afterwards, we show the adaptation of another extract of UWE to a very simple MDE development project: a music portal web application. Finally, we show how to use *SPEM4MDE Process Enactment Engine* when enacting the adapted process model.

1) *The UWE Process*

The objective of the UWE process is to give to web developers a systematic and semi-automatic support of web systems development based on models and their transformations. The process covers the whole development life cycle of web systems from the requirements specification to code generation. It is a model-driven development process following the MDA approach. It consists of a set of models and model transformations, specified by metamodels and model transformation languages. The metamodels are the Web Requirements Engineering metamodel (WebRE) [23], the UWE metamodel [9], and the metamodel of the Web Software Architecture approach (WebSA) [8] that contain respectively elements for modeling requirements, structure and behavior, and the architecture of Web systems. The UWE metamodel includes the *Content Metamodel*, the *Navigation Metamodel*, and the *Presentation Metamodel*. The UWE process starts with the definition of a *requirements model* that is a computational independent (CIM) business model. Two sets of platform independent design models (PIM) are derived from these requirements: *functional models* which represent the different concerns of the Web system (content, navigation, business logic, presentation, and adaptation); and *architectural models* which represent the architectural features of the Web system. *Functional models* are afterwards integrated mainly for the purpose of verification into a *big picture model*. A merge of this *big picture model* with the *architectural models* results in an *integrated model* covering functional and architectural aspects. Finally, platform specific models (PSM) are derived from the *integrated model* from which programming code can be generated. The transformation of *requirements model* into *functional models* is a composite *transformation,* each sub-transformation dealing with a separate concern of Web engineering.

2) *Describing UWE's Functional Models Transformation with SPEM4MDE-PSEE*

To illustrate the use of the SPEM4MDE Process Editor, Fig. 7 gives an extract of the UWE process model, related to the "*requirements model to functional models"* transformation. It comes as a set of six sub-transformations: "*Req 2 Content"*, "*Content 2 Navigation"*, "*Req 2 Navigation"*, "*Navigation Refinement"*, "*Navigation 2 Presentation"*, and "*Style Adjustment"*. These transformations are ordered as depicted by the links «*finishToStart*». Source and target metamodels of each transformation are depicted by the links «*source*» and «*target*», respectively. Input and output models of each transformation are depicted by the links «*in*» and «*out*», respectively. However, for readability of Fig. 7, we only give input and output models of the first two transformations. For instance, "*Content 2 Navigation"* has the "*Content Model"* as an input model, "*Content Metamodel*" as a source metamodel, "*Navigation Model"* as an output model, and "*Navigation Metamodel" as a target metamodel. "Navigation Model"* is conformed to "*Navigation Metamodel", while "Content Model"* is conformed to "*Content Metamodel"*. It should not start before "*Req 2 Content"* is finished, and must be finished before "*Req 2 Navigation"* can start. For more details about the UWE process, the reader is referred to [8].

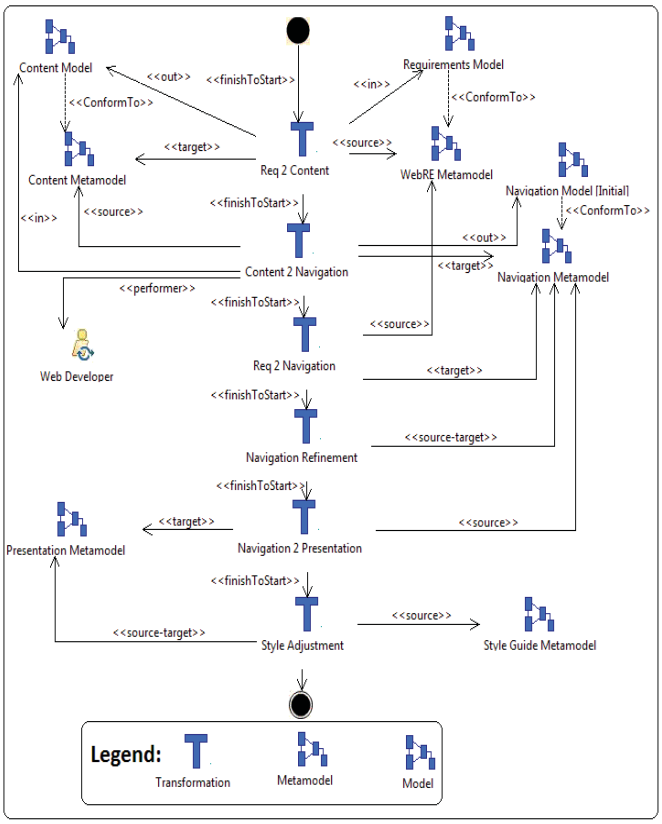


Figure 7. Functional Model Transformations within UWE Process

3) Adapting the UWE Process to a Project Example

Before a process model can be enacted for a given development project, it has to be adapted to the project by the means of *SPEM4MDE Process Editor*. This adaptation consists in specializing it into a project-specific process model, where each transformation is linked to its implementation and assigned resources (human actors, tools, workspaces, and used artifacts), and each activity is associated with its resources.

To illustrate how the UWE process model may be adapted with SPEM4MDE, let us consider a simple project example: a web portal application for vending musical albums, and an extract of the process, limited to the "*Content 2 Navigation"* transformation (shown in Fig. 7). The adaptation of this extract is depicted by Fig. 8. It consists first in specifying the implementation of the transformation ("*Content 2 Navigation Impl"*), and assigning its resources: a human performer (*Bob*) who plays the *Web Developer* role, an input model ("*Music Portal Content Model"*), an output model ("*Music Portal Navigation Model"*), and a tool (Smart QVT) for executing the transformation implementation (rules describing the transformation – not shown in the Fig. – are implemented in QVT). The project-specific UWE process model is linked with the UWE process model through stereotyped associations: «*implements*» for the transformation implementation, «*conformsTo*» for the input/output models, and «*plays*» for the human actor. Moreover, the state of "*Content 2 Navigation Impl*" is set to Executable (see Fig. 5). The adaptation of the other UWE's transformations is done similarly. At this stage, input and output models specify rather references than concrete models, which will be produced once the transformations are performed.

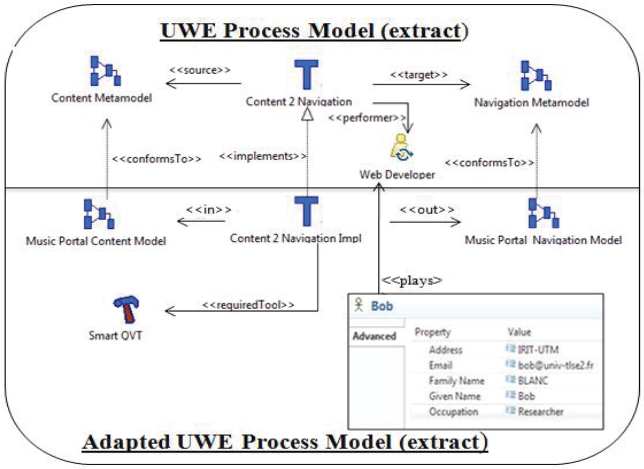


Figure 8. Adaptation of the "Content 2 Navigation" Transformation

4) UWE Process Enactment

Process enactment is based on the behavioral model described by state machines (as discussed in section III (sub-section C)). The *SPEM4MDE Process Enactment Engine* offers to developers entries of menu (i.e. enactment operators) that show for each process element the set of all operations specified by the corresponding state machine. Moreover, it also shows whether these operations are eligible or not. Once an operation is executed by developers, the state of the related process element is updated according to its corresponding state machine, and attached actions (with UML state-machines meaning) are executed, if any. As stated in the precedent section, a process enactment can start only once the process model is adapted to a given project.

To illustrate the enactment of the UWE process, let us consider the extract of the adapted UWE process shown by Fig. 8. Let us suppose that the process is at the stage where the implementation of the "*Req 2 Content*" transformation (shown in Fig. 7) has been performed, producing the "C*ontent Model"* (shown by Fig. 9 below) which is linked to "*Content 2 Navigation Impl"* as an input model.

As shown by Fig. 9, an artist produces albums, each album having one or more songs. A session may be associated to the application's current user. A credit card is used for buying albums, but may have an owner different from the user.

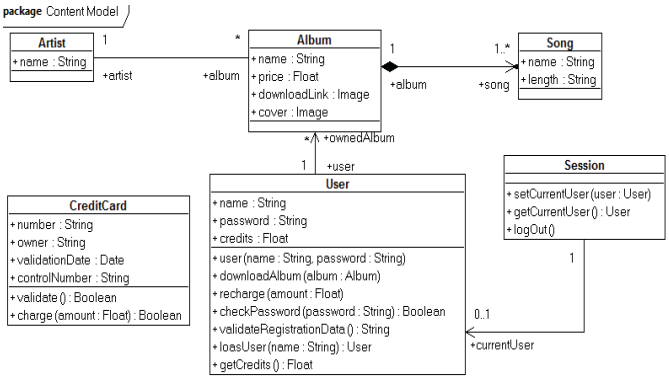


Figure 9. Source Model of "Content 2 Navigation Impl"

Fig. 10 shows the contextual menu displayed by the *SPEM4MDE Process Enactment Engine* for "*Content 2 Navigation Impl*". It indicates all possible operations described by the behavioral model of a transformation (see Fig. 5), and whether each operation is eligible (green mark) or not (red cross).

Initially, "*Content 2 Navigation Impl*" is in the state "*Executable"*. It has switched to the state "*Startable"*, because the "*Req 2 Content*" transformation is already performed (i.e. precedence of "*Content 2 Navigation*" is fulfilled), and the "*Content Model*" is created and validated (i.e. precondition "*Content 2 Navigation Impl*" is fulfilled). Since the current state is "*Startable"*, then the operation "*run*" is eligible. If a non-eligible operation is clicked on, a message displays (not shown in the Fig.) what conditions should be fulfilled in order to make it eligible.

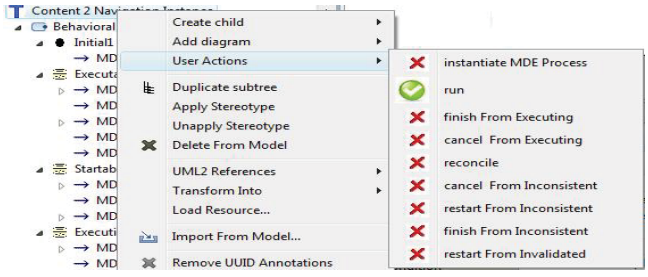


Figure 10. Contextual Menu of "Content 2 Navigation Impl "

When the operation "*run"* is executed, "*Content 2 Navigation Impl*" switches to the state "*Executing"*, its contextual menu is updated consequently (see Fig. 11 below). Moreover, the attached action (i.e. launching the associated tool (*Smart QVT*)) is performed.

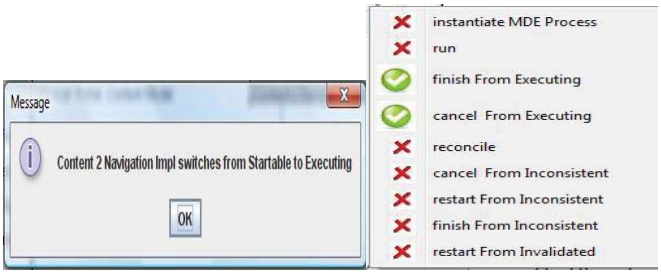


Figure 11. Displayed message (left) and the updated menu (right) after triggering the run action

If the "*finish From Executing*" operation is performed, the current state switches to the state "*Terminated"*, meaning that the output model (i.e. the "*Music Portal Navigation Model*" shown in Fig. 12) is created. At this stage, if this model is validated, "*Content 2 Navigation Impl*" switches to the state "*Validated"*, otherwise it switches to the state "*Invalidated"*. As shown by Fig. 12, the "*Music Portal Navigation Model*" describes the output model of "*Content 2 Navigation Impl*". From the home page, a user may register, recharge his account, or search an album. The result of the search is a list of albums. From this list, the user can view the details of each album and afterwards download it, and may then recharge its account. Before registering, recharging, or downloading an album, the user should log in.

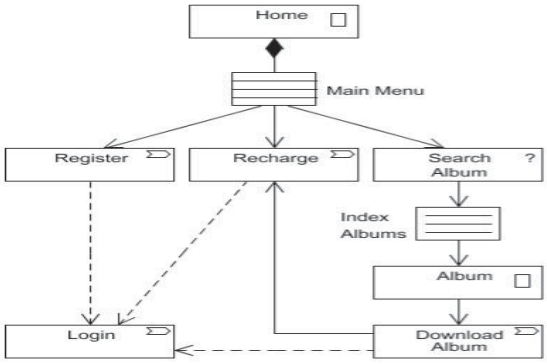


Figure 12. Target Model of "Content 2 Navigation Impl"

V. Related Works

Works related to our domain of interest mainly focus on transformations and MDE processes modeling. In the transformation area, we distinguish between model-to-code and model-to-model transformation approaches. In general, model-to-code can be viewed as a special case of model-to-model transformation; we only need to provide a metamodel for the target programming language. Among model-to-model transformation approaches, we distinguish between rule-based, graph-based, program-based and template-based approaches. Model transformation is the central topic of MDE and is essential to define a MDE process. The first MDE process came with the OMG's MDA initiative [3], [15] which depicted a general purpose process that can be applied to any application domain. Then, starting from the MDA approach, other MDE processes dedicated to middleware service [16], web applications [8], e-learning [17], models composition [20], embedded-systems [10], and a version of the Open Unified Process for MDD [24] have been proposed. However, there is a lack of consistent terminology since there is no unified language to specify MDE processes: each one adopts ad hoc notations and different concepts are used to define the activities and artifacts for software development life cycle.

Many languages and formalisms have been proposed for modeling software processes ([6], [12], and [29]), however only a few of them take into account explicitly MDE concepts [4], [11]. The SPEM 2.0 standard [12] describes a process in terms of activities, input and output products, and roles that perform activities, but fails on describing MDE concepts in a process. Moreover, SPEM does not address the process enactment in its last version. Nevertheless, it clearly suggests two possible ways of enacting SPEM 2.0 process models: mapping the SPEM 2.0 process models into project plans or linking SPEM 2.0 process elements with external behavior formalisms. To overcome the limitations of SPEM regarding enactment, several approaches based on state-machines (eSPEM [29], xSPEM [30], etc.), Petri nets (e.g. Porres' approach [4]), and on workflow (XPDL, BPEL [31], BPEL4PEOPLE, etc.) have been proposed.

The QVT standard [14] is suitable for defining model mappings and executable model transformations, but fails on describing process design aspects.

In [4], an approach that is targeted towards the development of software and systems using MDE methods is presented. The dynamics of this approach is based on Petri Nets. This approach can be integrated with existing approaches for software process modeling, but the metamodel contains only one concept (*transformation Tool*) that is related to MDE.

In [11], an approach to MDA process specification, based on the SPEM 2 standard concepts, is proposed. This approach is supported by a tool called *Transforms* which can be used to instantiate a MDA process for a given domain. Developers can describe steps and associate artifacts to perform MDA modeling and transformations chain. This approach has however some limitations, since it is tightly coupled with MDA concepts. Furthermore, it does not separate the specification of a transformation from its implementation.

VI. Conclusion

So far, MDE concepts are not explicitly supported by existing PML (Process Modeling Language). Reification allows promoting MDE concepts as first-class citizens.

In this paper, we have presented the specification and the implementation of SPEM4MDE, which is a metamodel dedicated to MDE software process modeling and enactment. SPEM4MDE extends SPEM 2.0 by adding concepts and semantics relating to MDE. It also reuses the QVT standard, and UML state-machines for defining the execution behavior of transformations and the behavior of MDE processes. In

addition, SPEM4MDE offers a set of behavioral MDE process models that process designers may reuse or adapt for a particular process. A prototype called SPEM4MDE-PSEE has been implemented under the TOPCASED environment. It helps process designers describe structural and behavioral aspects of SPEM4MDE process models and ensure that such process models are well-formed. SPEM4MDE-PSEE aims also at guiding the execution of projects based on a MDE process. Furthermore, when enacting a MDE process model which is specific to a project, developers can keep track of what is the current state of each element of such a MDE process model, what has been done before and what is left. To demonstrate the capabilities of our prototype, we have used an extract of the UWE process. The structure of this extract and the behavior of transformations that are involved in this extract have been modeled under *SPEM4MDE Process Editor*. *SPEM4MDE Process Enactment Engine* transforms the behavioral model of one transformation in UWE process as entries of menu representing operations that may be triggered at enactment time. Thanks to this menu, developers may know at any time, operations that are eligible or not.

Two important perspectives of this work are under consideration. Firstly, we intend to implement traceability of model transformations. The underlying objective is to replay easily a transformation when change occurs on its input models. Secondly, we envisage extending the SPEM4MDE metamodel for handling model-based collaborative development processes. For instance, the extension of SPEM4MDE for collaborative MDE processes will permit to coordinate developers' activities who work on the same model.

REFERENCES

[1] J. Bézivin, and E. Breton, "Applying the basic principles of model-engineering to the field of process engineering". European Journal for the Informatics Professional. 5, 27-33 (2004

[2] R. France, and B.Rumpe, "Model-driven development of complex software: A Research Roadmap". In: Proc. of the International Conference on Software Engineering (ICSE), pp. 37-54. IEEE Press, Minneapolis, Minnesota, USA (2007)

[3] J. Bézivin, and O. Gerbé, "Towards a precise definition of the OMG/MDA Framework". In: Proceedings of the 16th IEEE international conference on Automated Software Engineering (ASE), pp. 273. IEEE Press, San Diego, (USA) (2001)

[4] O. Porres, and M. C. Valiente, "Process definition and project tracking in model-driven engineering". In: Münch, J., Vierimaa, M. (eds.) PROFES 2006. LNCS, vol. 4034, pp. 127-141. Springer, Amsterdam (2006

[5] A. Kleppe, J. Warmer., and W. Bast, "MDA explained the model-driven architecture: practice and promise", Addison-Wesley (2003)

[6] R. Bendraou, M.P. Gervais, and X. Blanc, "UML4SPM: a UML 2.0-based metamodel for software process modeling". In: Briand, L., Williams, C. (eds.) MoDELS 2005. LNCS, vol. 3713, pp. 17-38. Springer, Montego Bay, Jamaica (2005)

[7] F. Fondement, and R. Silaghi, "Defining model-driven engineering processes". In: 3rd UML Workshop in Software Model Engineering (WiSME), Springer, Lisbonne (2004)

[8] N. Koch, "Transformations techniques in the model-driven development process of UWE". In: 6th International Conference on Web Engineering (ICWE), Volume 155 Article N° 3. ACM, California (2006)

[9] C. Kroiß, and N. Koch, "UWE metamodel and profile: user guide and reference". LMU, Technical Report (2008).

[10] A. Garcia, B. Combemale, X. Crégut, J.N. Guyot, and B. Libert, "TopProcess: A process model-driven approach applied in Topcased for embedded real-time software". In: European Congress on Embedded Real-Time Software (ERTS), Société des Ingénieurs de l'Automobile, Toulouse (2008)

[11] R.S.P.Maciel, B.C Silva, A.P.F. Magalhães, and N.S Rosa, "An approach to model-driven development process specification". In: 11th International Conference on Enterprise Information Systems (ICEIS), pp. 27-32. INSTICC Press, Milan (2009)

[12] SPEM 2.0, http://www.omg.org/spec/SPEM/2.0

[13] UML version 2.2, http://www.omg.org/spec/UML/2.2/

[14] MOF 2.0 QVT 1.0, http://www.omg.org/spec/QVT/1.0/

[15] MDA at, http://www.omg.org/mda/executive_overview.htm

[16] R. S. P.Maciel, B. C. Silva, , and L. A. Mascarenhas, "An edoc-based approach for specific middleware services development". In: 4th Workshop on MBD of Computer Based System, pp.135-143. IEEE Press, Postdam (2006)

[17] H. Wang, and D. Zhang, "MDA-based development of E-Learning system". In: 27th International Computer Software and Applications Conference (COMPSAC), pp. 684-689. IEEE Press, Texas (2003)

[18] TOPCASED, www.topcased.org

[19] S. Kent, "Model-driven engineering". In: Grieskamp, W., Santen, T., Stoddart, B. (eds.) IFM 2002. LNCS, vol. 2335, pp. 286-298. Springer, Turku, Finland (2002)

[20] A. Anwar, S. Ebersold, B. Coulette, M. Nassar, and A. Kriouile, "A QVT-based approach for model composition - application to the VUML profile". In: 10th International Conference on Enterprise Information Systems (ICEIS), pp. 360-367. INSTICC Press, Barcelona (2008)

[21] S. Diaw, R. Lbath, and B. Coulette, « State of the art of sofware development based on model transformations ». Technique et Science Informatiques 29, N° 4-5, 505-536 (2010).

[22] S. Diaw, R. Lbath, V. Thai Le, and B. Coulette, B., "SPEM4MDE: a metamodel for MDE software processes modeling and enactment". In: 3rd Workshop on Model-Driven Tool & Process Integration - Associated to EC-MFA, pp. 109-121. Fraunhofer, Paris (2010).

[23] M. J.Escalona, and N. Koch, "Metamodeling the requirements of web system". In: 2nd International Conference on Web Information System and Technologies (WEBIST), pp. 310-317. INSTICC Press, Setúbal, Potugal (2006)

[24] OPEN UP at: http://www.eclipse.org/epf/openup_component/mdd.php

[25] E. Rios, T. Bozheva, A. Bediaga, and N. Guilloreau, "MDD maturity model: a roadmap for introducing model-driven development". In Rensink, A., Warmer, J. (eds.) ECMDA-FA 2006. LNCS, vol. 4066, pp. 78-89. Springer, Bilbao (2006)

[26] T. Stahl, and M. Volter, "The Model-driven software development Technology, Engineering, Management", Translation copyright by John Wiley & Sons, Ltd (2006)

[27] W. Humphrey, and M. Kelner, "Software modeling: principles of entity process models". SEI - Carnegie Mellon University. Pittsburgh, Pennsylvania, (1989).

[28] X. Larrucea, A. B. García Díez, and J. X. Mansell, "Practical model-driven development process". In: Second European Workshop on Model Driven Architecture (MDA) with a emphasis on Methodologies and Transformations, pp. 99-108. Computing Laboratory, University of Kent, Canterbury, UK (2004)

[29] R. Elner, S. Al-Hilank, A. Bediaga, J. Drexler, M. Jung, D. Kips, and M. Philippssen, "eSPEM – A spem extension for enactable behavior modeling". In: Kuhne, T., Seloc, B., Gervais, M.P., Terrier, F. (eds.) ECMFA 2010. LNCS, vol. 6138, pp. 116-131. Springer, Paris (2010).

[30] R. Bendraou, B. Combemale, X. Crégut and M.-P. Gervais : "Definition of an eXecutable SPEM2.0". In: 14th Asia-Pacific Software Engineering Conference (APSEC), pp. 390-397. IEEE Computer Society, Nagoya, Japan (2007).

[31] Web Services Business Process Execution Language Version 2.0. Working WS-BPEL TC OASIS, April 2007. URL: http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf

Conformance Checking of Software Development Processes Through Process Mining

Artini M. Lemos, Caio C. Sabino,Ricardo M. F. Lima, César A. L. Oliveira
Center for Informatics, Federal University of Pernambuco
Recife, Pernambuco, Brazil
{aml2, ccss2, rmfl, calo}@cin.ufpe.br

Abstract

The design and management of software development processes is essential to reduce costs and improve the quality of software products. The execution of such processes is usually monitored to register important information about the dynamic behavior of the software development process. As a result, a huge amount of information is stored in the database and the software managers are deluged by data. Then, it is important to find ways to automatically analyze the process execution logs to discover the actual process model, characterize trends in it, and flag anomalies. Process mining is a well established technique designed to attack this challenge. This paper demonstrates the application of process mining technique and tools to perform a software development process conformance analysis to detect inconsistencies between a process model and its corresponding execution log. The work conducts a case study using an event log database with more than 2,000 process instances gently provided by a Brazilian software house. The paper aims at closing the gap between process mining and software engineering areas. We hope to motivate and guide software managers to adopt process mining technique as a practical tool for the analysis and improvement of software development processes.

1. Introduction

Software development usually follows a sequence of activities designed to capture the best practices from software development projects. Such sequence of activities defines the software development process, which should be constantly revised and improved to embody strategies for accomplishing software evolution. Even when the process is automated, due to personal preferences, previous experiences, or even time and cost pressures, it is common to see project managers adopting their own software development processes. Moreover, usually some process activities are ignored to save time and meet the project deadline. Therefore, the first step of any software process improvement project should be the conformance analysis to determine if the defined process is being actually followed.

In this context, it is important to discover the real software development process to compare it with the company's formal process. Software development process discovery has been an intensive task, either done through exhaustive empirical studies or in an automated fashion.

Software houses usually automate their development processes through a monitoring system. This system is capable of registering valuable information about process execution, such as timestamps, task identification, stakeholders involved in the execution of an activity etc., in the form of execution logs. Such execution logs (also called event logs or audit trails) are the starting point for the process mining technique, which was developed to semi-automatically extract useful information from them [4].

Processing mining has been successfully applied in many different areas, including software engineering. In [1] authors explore process mining techniques for discovering development processes from publicly available open source software development repositories. In [6] authors define a process mining framework to discover software development processes from event logs generated by software configuration management (SCM) systems. Unfortunately, these works are more oriented to the computational intelligence area. They focus on describing the mining techniques, demonstrating their efficiency in discovering software process models. Less attention is devoted to the application and use of such techniques to evaluate the software development process in a real environment.

This work aims at bridging this gap between process mining and software engineering areas by demonstrating how to explore process mining techniques to perform a conformance analysis and evaluate whether the established software development process is being followed. The conformance analysis main goal is detecting inconsistencies

between a process model and its corresponding execution log. Despite some works have explored process mining techniques for the conformance analysis of general business processes [5], it is not of our knowledge any study that applies process mining to conformance checking of software development processes.

In this work we use the process mining ProM tool [9] to mine the event logs of a real company and discover the process model that is actually employed by them. We are particularly interested in the process *control-flow perspective*. In the control-flow conformance analysis, we want to evaluate if process activities are being executed in the correct order and discover which activities are actually being executed in the process instances. For this objective, we employed a feature of the ProM framework that generates a Markov chain from the process logs.

The paper explores a real database with event logs generated in the past five years from the execution of a software development process. The database was gently provided by a Brazilian software house with annual revenue of more than US$ 500 million. The database includes more than 2,000 cases (process instances).

The paper is organized as follows. Section II describes the process mining technique and presents the ProM tool. Section III discusses the conformance checking study conducted in this work. It begins by presenting the formal software development process defined by the company; then describes the event log characteristics and the mining strategy; in the next step it demonstrates how to use ProM to identify problems in the even logs and how these problems were corrected; finally, it discusses the conformance analysis study herein conducted. Conclusions and future works are discussed in Section IV.

2. Process Mining

Business process mining, or just *process mining*, is a "nascent research field at the intersection of data mining and business process modeling" [3] that targets the extraction of non-trivial and useful information based on execution logs [4]. Since many systems record their transactions in log files, process mining techniques can be used for a large class of business processes. By using process mining, many kinds of information can be collected about the process, such as control-flow, performance, organizational information and decision patterns.

It is important to note that most process mining techniques do not generate a model for only reproducing all log entries, but collects general information about the process real execution, considering that exceptional behavior or some minor errors may occur in the log file that should not be part of the process.

Once the process model is obtained and possibly validated, it can be analyzed in order to better understand how it actually works. In the analysis of the process model, some changes can be proposed and their effects can be estimated and measured. That means that the impact of some changes can be calculated without having to change and verify the real process. Therefore, process mining techniques can be used to gain insight on how to improve the process quality.

Many process mining algorithms have been proposed. The ProM Framework [9] is a tool that features many mining and analysis plug-ins implementing several of these algorithms.

ProM is an extensible open source framework implemented in Java with a user-friendly graphical interface that provides a wide variety of plug-ins that support, implement, and use process mining techniques. There are currently more than 230 plug-ins, distributed on functionalities like filtering, mining, analysis, exporting and conversion.

The input log files for ProM are in a common log format: Mining XML (MXML). ProM can also take as input process model files, such as PNML and many others. When the log file is opened, ProM collects some statistical data about it and present them in the main window. Also, many plug-ins become available to operate on the log.

The mining plug-ins aim at discovering multiple perspectives of the process, such as the control-flow, the organizational structure, the data perspectives and some other information. Details about these perspectives are given in [9]. The analysis plug-ins deal with verification of process models, conformance checking between the log and the model, performance analysis, among other things.

ProM also provides filters for cleaning the log from undesired elements. Filtering "is usually a projection of the log to consider only the data you are interested in" [9]. Also, some mining plug-ins have configurable settings in order to obtain better results depending on the user's purposes.

ProM allows multiple combinations of plug-ins, making it possible to analyze the results of a mining algorithm or merge the contents of multiple perspectives to obtain a more complete model for the process.

With an extensible framework, new algorithms can be easily added as new plug-ins and this makes ProM a very powerful tool. ProM works with several modeling languages, such as Petri Nets [2], EPCs [7], YAWL [8], and supports the conversion between most of them, being a very versatile tool.

3. Discovering a Software Development Process

In this section, we explore the case of a Brazilian software house with annual revenue of more than US$ 500 million. Five years ago, the company defined a software devel-

opment process and deployed a monitoring system to register relevant information about its execution. Since then, a huge volume of event logs has been generated, in a way that became impossible to the software quality team to manually extract useful and trustful information from the database.

In this context, we applied process mining techniques to extract important information on the real software development process from the event logs. The main objective is to discover the actual process being executed by the development teams and compare it against the documented process. The database under consideration includes more than 2,000 instances of the process.

3.1. The Formal Software Development Process

In its documented specification, the software development process begins with the activity *Proposal and Production Budget* (PPB) as it can be seen in Figure 1. The PPB includes two tasks: 1) creating a functional specification document, which aims at detailing the recommended solution; 2) and filling in a worksheet to estimate the software development effort. A manager analyzes the functional specification document in the activity *Functional Specification Document Analysis* (FSDA) before submitting the proposal to the client. In the next step the production planning and budget proposal is sent to the client - activity *Send Production Planning and Budget* (SPPB). Then, the client analyses the production planing and budget proposal and returns an approval/reject response - activity *Client Proposal Analysis* (CPA).

When the customer approves the proposal and production budget (CPA activity), the team leader analyses the technical specification to evaluate its complexity. Only complex specifications are validated - activity *Validate Technical Specification* (VTE). The *Development* (DEV) activity starts after validation. During this activity, programmers implement the software, create the processing scripts and database objects script (triggers, functions, packages, and view). When the development phase is completed, the *Code Verification* (CODV) activity is performed to check whether code complies with the specification and follows the development patterns. After the code verification phase, the program is tested in the *Testing* (TEST) activity. Eventually, with the software validated and tested, the process reaches the *Documentation* activity (DOC) and the process terminates.

This is an iterative process, allowing specification refinements/corrections and negotiation of the production planning and budget proposal. As one can see in Figure 1, activities FSDA, CPA, VTE, CODV, and TEST represent points where the process may move back to previous activities.

3.2. Mining the Software Process Logs

The company's quality team is interested in assessing the model (or models) of the real process and compare it against the process that was supposed to be executed. Such conformance analysis will be the starting point for a process improvement project in the company.

The Mining XML (MXML) is an extensible, XML-based format for storing process event logs which can be analysed by the ProM tool [9]. Since the company's database adopts a private file format, a compiler was developed to translate the event logs into the MXML file format.

ProM allows one to discovery different *perspectives* of the process being analyzed, namely control-flow, performance, and organizational perspectives. In the conformance analysis conducted in this paper, we are interested in the *control-flow perspective*. To discover which tasks are actually performed and their order of execution is of particular importance for the software quality team. The idea is to compare the control-flow model of the formal process against the control-flow model (or models) discovered from the real event logs.

In the ProM tool, information about each event is recorded in the form of an audit trail entry. Our event logs store the following information on each audit trail: 1) **taskID**: name of the task involved in a process event; 2) **event type**: a mark to indicate whether the event is a start or complete point in the process instance; 3) **timestamp**: moment at which the event took place; 4) **originator**: worker who perform the task (originate the event).

Since the audit trail entry identifies the tasks that are executed in a process (*taskID*) and allows for inferring their order of execution (*event type* and *time stamp*), then the control-flow perspective can be discovered.

Existing techniques for process mining perform very well for well-structured and organized event logs. However, the situation is not that simple upon dealing with real event logs. In this case the logs are usually very obscure and have noises, making it difficult to extract useful information. Therefore, in these situations a preprocessing phase to remove noises and organize the event logs is an obligation.

Our event log database was generated in the past five years, in a very active company, with hundreds of process instances executed each year. Thus, the preprocessing phase is essential to produce a well structured event log database. We used the reports produced by ProM tool to carefully analyze the database and a number of problems were found and fixed, as follows.

Incomplete timestamps - a) The monitoring system only registers the moment a given activity initiates, but it does not monitor the activity duration. However, this would be a concern only if we were interested in analyzing the *per-*

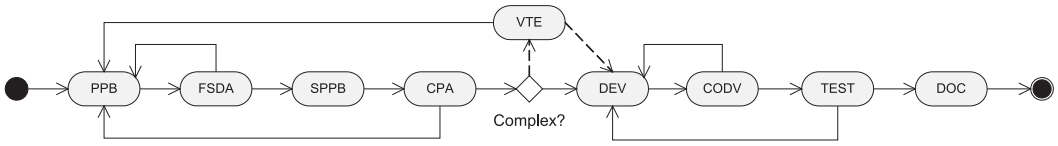


Figure 1. The Formal Software Development Process

formance perspective of the process, which is not the case. b) The original log recorded timestamps using information about days, months, and years, but missing hour, minute, and second details. Thus, activities occurring within a day were assumed to be parallel activities, rather than sequential ones. To solve this problem we redefined timestamps such that all activities in a process instance received different timestamps. We used the formal company's process to determine the order of activities in time. One might argue that the solution is biased by the control-flow of the formal process. However, observing the original database, in more than 90% of the cases, if two activities in the same process instance have different timestamps, they follow the control-flow order defined for the formal process, which is a good result for the conformance analysis being conducted. Therefore, our assumption for the timestamps modification is acceptable. Nevertheless, despite the good conformance results for the control-flow order, in most cases, the set of activities being executed in a process instance are only a subset of the activities defined in the formal process. So, a deeper conformance analysis is still required.

Database corruption - Data stored in the audit trail entry of a small number of process instances appeared to be corrupted. For instance, we found timestamps with invalid date values. Moreover, some process instances had no start or complete events. Fortunately, the number of process instances with this kind of problem represented less than 2% of the whole database. Thus, they were discarded without significative loss for the conformance analysis study.

After the database preprocessing, we proceeded with the conformance analysis. We want to answer two main questions: 1) does the real process respect the ordering of execution defined in the formal process?; 2) which subset of activities in the formal process are actually executed in the real process? As we already observed, the answer for the first question is yes for more than 90% of the cases. But we need to investigate more to answer the second question.

3.3. Conformance Analysis

In this section, we describe the approach employed to analyze the software process based on the information provided by process mining.

ProM features a plugin capable of generating a Markov chain from the discovered model, where each node is a process activity and transitions are labeled with probabilities as in ordinary Markov chains. Through the Markov chain it is possible to discover if sequences of activities defined for the process actually occur, and with which frequency. It is also possible to measure the probability of skipping a given activity that is defined in the flow and the probability of each decision that must be taken during the process execution.

We used Prom's Markov chain model discovery feature in our study. Table 1 presents the Markov chain generated using ProM's *sequence clustering analysis plug-in*.

With the Markov chain in hands, we can start the conformance analysis of the software development process. The purpose is to detect points in the actual process that are not in conformance with the company's expectations and provide useful information for the company to initiate the process improvement project.

Figure 2 presents the formal process enriched with transition probabilities assessed from the Markov chain (Table 1). For clarity and readability, Fig. 2 includes only transitions relevant for the control-flow conformance analysis. Besides the transitions already present in the formal process, we included some transitions (dashed arrows) to indicate undesirable situations observed in the discovered process model.

Now, let us analyze the results presented in Fig. 2 to point out and discuss some conformance problems.

The probability of transitioning from the initial state (in) to the development state (DEV) is of 25.2%. This means that about one in each four projects started by the development activity, skipping the whole planning stage.

The activity send production planing and budget (SPPB) is almost never reached by any other activity in the process model and could be removed from the formal process.

Although *validate technical specification* (VTE) is performed for complex software projects, the execution of such activity in the actual process is fairly rare, with probability of 0.2%. This points out that this activity is executed in very exceptional cases and, therefore, could be removed from the company's standard development process.

A more critical problem can be seen in the end of the process. Software verification, testing, and documentation are important activities of the best practices for software development projects. Nevertheless, such activities are ignored by 43.6% of the software development processes analyzed (see transition from the development phase (DEV) to the

Table 1. Markov chain model for the process

	in	PPB	FSDA	SPPB	CPA	DEV	VTE	CODV	TEST	DOC	out
in	-	0.56	0.018	0.003	0.128	0.252	-	0.018	0.007	0.014	-
PPB	-	-	0.285	0.021	0.276	0.136	-	0.005	0.008	0.003	0.266
FSDA	-	0.068	-	-	0.305	0.19	-	0.032	0.004	0.016	0.385
SPPB	-	-	-	-	-	0.658	-	0.026	-	-	0.316
CPA	-	0.01	0.006	0.002	-	0.767	0.002	0.065	0.015	0.002	0.131
DEV	-	0.016	0.006	-	0.013	-	0.002	0.462	0.061	0.004	0.436
VTE	-	-	-	-	-	0.75	-	0.25	-	-	-
CODV	-	0.009	-	-	0.003	0.035	-	-	0.234	0.013	0.706
TEST	-	0.006	0.006	-	0.029	0.02	-	0.018	-	0.553	0.368
DOC	-	0.032	-	0.001	0.028	0.016	-	0.036	0.015	-	0.872
out	-	-	-	-	-	-	-	-	-	-	-

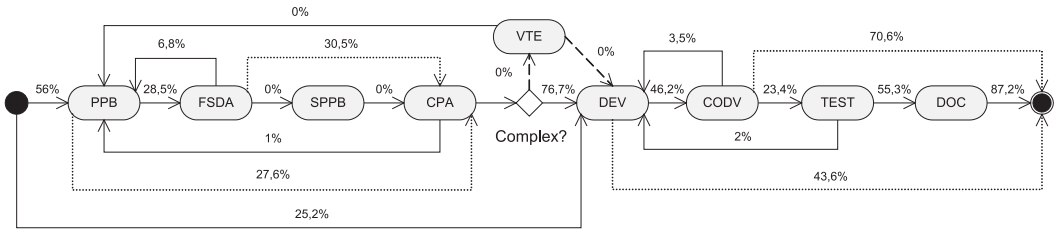


Figure 2. The actual development process with the undesired transitions shown as dashed arrows

Table 2. Activity execution frequency

Activity	Absolute occurrence	Relative occurrence
DEV	1556	27.385%
PPB	1356	23.865%
CODV	853	15.012%
CPA	841	14.801%
FSDA	442	7.779%
TEST	342	6.019%
DOC	250	4.4%
SPPB	38	0.669%
VTE	4	0.07%

terminal state (out)). This shows that the software project terminates in the development stage with a very high probability, without even being verified. Even worst, looking at the transition from the *code verification* activity (CODV) to *testing* (TEST), one can notice that, if the software happen to be verified, it is usually neither tested (TEST) nor documented (DOC).

Eventually, Figure 2 demonstrates that the actual process is not iterative. Thus, it is very unlike to observe a return to previous activity during the process execution. Apparently, when a problem occurs during the process execution, it goes straight to the terminate state.

In ProM, a cluster of process instances with same sequence of activities is called a *process type*. Using ProM's sequence clustering plugin we can discover the process types present in the log and the frequency of occurrence of each one. The goal is to evaluate the conformance of these process types against the formal process.

ProM discovered 190 different process types in our event log. Figure 3 presents the two most frequent. The most frequent process type only executes the development (DES) activity, which is a clear indication of nonconformance against the formal process. On the other hand, the second most frequent process shows that many projects are cancelled by the company after the proposal and production budget (PPB), which is a fairly acceptable behavior.

We then analyzed the list of the ten most frequent process types to select those with at least three activities. Out of the ten most frequent process types, only the 4th, 5th, and 10th had three or more activities. Figure 4 depicts these process types. The *testing* (TEST) and *documentation* (DOC) activities are not executed in the fourth and fifth most frequent process types. A good conformance result is observed in the tenth most frequent process type. Only *validation* (VTE) and *send production planing and budget* (SPPB) activities are missing. Nevertheless, VTE activity is optional in the formal process. Additionally, through the analysis of the Markov chain, we had already identified that the SPPB activity is almost never executed and recommended its elimination from the formal software development process.

Figure 3. The most frequent process types

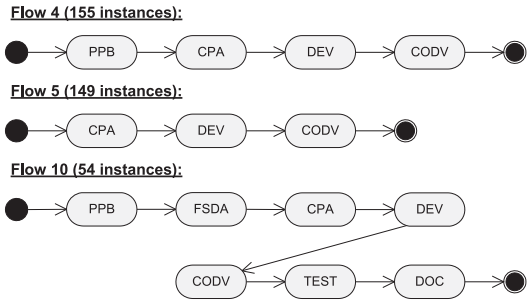


Figure 4. Most similar process types to the formal development among the ten most frequent

4 Conclusions

This paper demonstrated the application of process mining techniques to examine if the actual software development process employed by a company conforms to its formal specification. The work is conducted within the context of a Brazilian software house. A total of 2,000 process instances, recorded in five years of operation, were evaluated.

The conformance checking demonstrated that, although some projects follow the formal process, most of them do not occur as expected. Some serious problems were identified. The initial phase of the formal process, responsible for budged/functionalities proposal/approval, is usually skipped. This may have important impact on the budged planning and increase the costs with software maintainance. Another conformance problem was observed in the final phase of the process. The software implementation is validated against the specification only in 46.2% of the cases. Even worst, if the software happens to be validated, in 70.6% of the cases it is neither tested nor documented.

It is not the purpose of this work to evaluate the impact of the conformance problems in the company's productivity. Indeed, the conformance analysis produced a set of reports indicating inconsistencies detected between a process model and its corresponding execution log. For instance, one of the reports enumerate the most common *process types*[1] in the event logs. In this report, we observed that the tenth most common process type conforms to the formal software development process. However, only the software quality team is able to attest if this is a good scenario and take actions to solve the detected issues.

This work aimed at bridging the gap between process mining and software engineering areas. From a software engineering perspective, it describes how to apply the process mining tool in a practical fashion. We believe that software project managers can benefit from this paper by gaining insights on how to incorporate process mining tools in their professional routines.

Although in the conformance analysis we have explored the *control-flow perspective*, the procedures may be easily adapted to evaluate other software development process perspectives, such as performance, cost, resource allocation, and many others. As a future work, we intend to demonstrate how process mining can be applied to analyze other software development process perspectives and develop a guide to help software engineering people in the use of process mining techniques and tools.

[1] *process type*: a cluster of process instances with the same sequence of activities

References

[1] Chris Jensen and Walt Scacchi. Data mining for software process discovery in open source software development communities, 2004.

[2] Wolfgang Reisig. *Petri nets: an introduction*. Springer-Verlag New York, Inc., New York, NY, USA, 1985.

[3] Aubrey J. Rembert and Clarence Ellis. An initial approach to mining multiple perspectives of a business process. In *The 5th Richard Tapia Celebration of Diversity in Computing Conference*, pages 35–40, New York, NY, USA, 2009. ACM.

[4] A. Rozinat, R. S. Mans, M. Song, and W. M. P. van der Aalst. Discovering simulation models. *Inf. Syst.*, 34(3):305–327, 2009.

[5] A. Rozinat and W. M. P. van der Aalst. Conformance checking of processes based on monitoring real behavior. *Information and Systems*, 33:64–95, March 2008.

[6] Vladimir Rubin, Christian W. Günther, Wil M. P. Van Der Aalst, Ekkart Kindler, Boudewijn F. Van Dongen, and Wilhelm Schäfer. Process mining framework for software processes. In *2007 international conference on Software process*, ICSP'07, pages 169–181, Berlin, Heidelberg, 2007. Springer-Verlag.

[7] Wil M. P. van der Aalst, Arjan J. Mooij, Christian Stahl, and Karsten Wolf. Service interaction: Patterns, formalization, and analysis. In *SFM*, pages 42–88, 2009.

[8] Wil M. P. van der Aalst and Arthur H. M. ter Hofstede. Yawl: yet another workflow language. *Inf. Syst.*, 30(4):245–275, 2005.

[9] B. F. van Dongen, A. K. A. de Medeiros, H. M. W. Verbeek, A. J. M. M. Weijters, and W. M. P. van der Aalst. The proM framework: A new era in process mining tool support. In *Applications and Theory of Petri Nets 2005: 26th International Conference*, volume 3536, pages 444–454. Springer Verlag, June 2005.

SET-MM – A Software Evaluation Technology Maturity Model

Raúl García-Castro
Ontology Engineering Group
Departamento de Lenguajes y Sistemas Informáticos e Ingeniería de Software
Facultad de Informática, Universidad Politécnica de Madrid, Spain
rgarcia@fi.upm.es

***Abstract*—The application of software evaluation technologies in different research fields to verify and validate research is a key factor in the progressive evolution of those fields. Nowadays, however, to have a clear picture of the maturity of the technologies used in evaluations or to know which steps to follow in order to improve the maturity of such technologies is not easy. This paper describes a Software Evaluation Technology Maturity Model that can be used to assess software evaluation technologies in a research field. To illustrate the use of this model, we have employed it for assessing the maturity of software evaluation technologies in some evaluation initiatives within the semantic research field.**

I. Introduction

For research to reach maturity it is necessary to improve the quality of the research processes and their results over time. However, to apply the vague concept of maturity to a research field, which is itself difficult to define, may seem an arduous task.

It is well known that any research field has to periodically assess its status by measuring different aspects of it (e.g., outreach to other fields, quality of publications, technology maturity) and that the results obtained from these assessments are the ones that initiate the actions needed towards improvement and maturity in the field.

Nevertheless, to perform successful assessments for choosing the most appropriate actions, it is necessary to have clear goals and be aware of what is needed to achieve those goals.

Maturity models have been used for decades to guide improvement by providing both a framework of incremental maturity levels to be compared against and different measurable goals that must be satisfied to achieve each maturity level.

The most relevant maturity models are probably the Capability Maturity Model (CMM or SW-CMM) [1] and its successor, the Capability Maturity Model Integration[1] (CMMI). Both maturity models have been defined by the Software Engineering Institute with the goal of improving an organization's software development processes.

Other maturity models have also been defined, most of them inspired by CMM or CMMI, mainly in the software engineering domain but also in domains such as the organization management one (e.g., the Portfolio, Programme and Project Management Maturity Model [2]) or the e-learning one (the e-learning Maturity Model [3]).

One of the measurable aspects of a research field is what technologies support the evaluation of the research outcomes. In fact, the application of software evaluation technologies in any research field to verify and validate research is a key factor for research evolution by means of experimentation-driven research [4].

The goal of this paper is to describe a Software Evaluation Technology Maturity Model (SET-MM) that can be used to assess software evaluation technologies in any research field.

Similarly to other maturity models, the goal of the SET-MM is not to derive a figure for a specific level but to provide some guiding to improve the current activities in the research field.

To illustrate how the SET-MM maturity model functions, we have used it to assess the maturity of software evaluation technologies in some evaluation initiatives within a concrete research field, that of semantic research.

This paper is structured as follows. Section II introduces other maturity models that are related to the one here presented as they cover similar domains. Then, Section III presents the scope of this work and the process followed to define the maturity model, whereas Section IV enumerates the assumptions taken into account when defining this maturity model. Section V describes the five maturity levels of SET-MM and the main notions behind the model. Section VI presents how we have used the maturity model here presented to assess the maturity of software evaluation technologies in the semantic research field. Finally, Section VII draws the conclusions from this work and proposes future lines of research.

II. Related Work

This section presents other maturity models that cover similar domains and that have served us as input for defining the maturity model presented in this paper.

The *Capability Maturity Model for Software* was defined by the Software Engineering Institute in the early 1990s [1]. Since then different maturity models have appeared, each collecting the best practices to be used when comparing an organization's practices and guiding process improvement.

The proliferation of such models has led to their combination into a single improvement framework, namely, the *Capability Maturity Model Integration* that, in its current

[1] http://www.sei.cmu.edu/cmmi/

version, contains three different models: CMMI for Acquisition (CMMI-ACQ) [5], which is used for acquiring products and services; CMMI for Development (CMMI-DEV) [6], for developing products and services; and CMMI for Services (CMMI-SVC) [7], for providing superior services.

The CMMI models focus on the following dimensions that affect organizational improvement: people, procedures and methods, tools and equipment, and processes.

From the number of maturity models that have been defined we present here six, all related to SET-MM and largely inspired by CMM or designed to work in conjunction with it. While the first three models (namely, the Testing Maturity Model, the Measurement Capability Maturity Model and the Software Measurement Process Capability Maturity Model) focus on process maturity, the other three focus on technology maturity (as our model does).

The *Testing Maturity Model* (TMM) [8] was designed to assist software development organizations in evaluating and improving their testing processes.

It covers nine different attributes of a mature testing process: testing policies, test life cycle, test planning process, test group, test process improvement group, test-related metrics, tools and equipment, controlling and tracking mechanism, and product quality control.

The *Measurement Capability Maturity Model* (M-CMM) [9] enables organizations to assess their software and software process measurement capabilities and provides organizations with directions for improving their measurement capability.

It covers the following main areas: measurement focus, measurement design, measure collection, measure analysis, technology support, measurement feedback, measurement training, and measurement management.

The *Software Measurement Process Capability Maturity Model* (SMP-CMM) [10] helps organizations to assess their measurement processes and provides guidelines for improving them.

It supports improvement in five areas: metrics plan, data collection, data gathering, data analysis, and feedback activity.

Daskalantonakis et al. [11] defined a measurement technology maturity model for assessing the software measurement technology of an organization.

Their maturity model covers ten different themes: formalization of the development process, formalization of the measurement process, scope of measurement, implementation support, measurement evolution, measurement support for management control, project improvement, product improvement, process improvement, and predictability.

Wettstein and Kueng [12] defined a maturity model for performance measurement systems, that is, systems that track and manage the performance of an organization (or part of it).

Their maturity model covers six different dimensions: quality of measurement process, scope of measurement, data collection, data storage, use of measures, and communication of results.

Gao et al. [13] define five levels to evaluate the maturity level of software test automation in the test processes of an organization.

In their work, maturity is evaluated through two different perspectives regarding the existence of systematic solutions and tools that support a) the different tasks to be performed during software testing; and b) the measurement of the testing process.

If we classify the main areas covered in these maturity models according to the four improvement dimensions (people, procedures and methods, tools and equipment, and processes) taken into account in CMMI, it can be observed how all these approaches cover the four dimensions and emphasize the areas related to processes while understate those related to people.

While the model presented in this paper is quite similar to the maturity models described above, the main difference is that these maturity models focus on an organization, whereas in our case the focus is on a research field.

This entails two major distinctions between SET-MM and the other models. First, and contrarily to the maturity models presented, we do not cover processes in our model. This is so mainly because at a research field level it is not possible to define, measure, or control any process since the field is composed of highly distributed and heterogeneous members. The second distinction is that data themselves are main topics in SET-MM since the ability to improve largely relies on the capacity of reusing the different evaluation data (e.g., workflows, test data, results). By contrast, in the other maturity models data are secondary issues.

III. Research Methodology

This section presents the scope of the work presented in this paper and the process followed to define the maturity model.

As mentioned above, the maturity model here presented is solely focused on software products and does not cover software processes.

Besides, this maturity model is grounded in the notion of evaluation as defined by the ISO/IEC 14598 standard on software product evaluation [14] and in the following evaluation entities: in any *evaluation* a given set of *tools* are exercised, following a given *evaluation workflow* and using determined *test data*. As an outcome of this process, a set of *evaluation results* is produced.

These entities are depicted in Figure 1. A detailed description of them and of their life cycles can be found in [15].

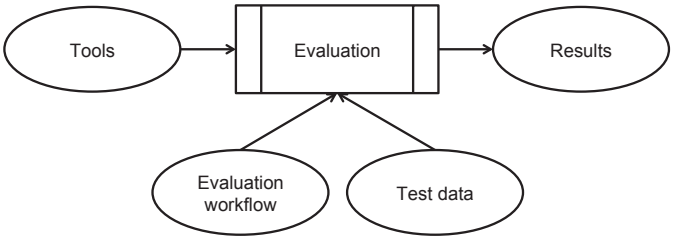


Fig. 1. Main entities in a software evaluation scenario

Furthermore, any evaluation activity is always expected to be the input of some decision-making process. Niessik and

van Vliet [16] exemplify this by proposing a generic process model for measurement-based improvement in organizations; such a model is composed of a measurement cycle followed by an improvement one in which changes in the organization are implemented based on the measurement results.

In our case, the scope is limited to the measurement (i.e., evaluation) cycle; thus, it does not cover improvement, since the analysis and change processes of a research field cannot be controlled nor monitored.

The process followed to define the maturity model contains the same steps than those defined by [11]:

1) First, we identified the set of assumptions upon which the different software evaluation technology maturity levels are defined. Each of these assumptions defines one or more themes (i.e., aspects) that influence maturity.
2) From these themes, five evolutionary stages were defined; these stages have to be followed by any research field in order to reach the highest level of maturity for that particular theme.
3) Then, level *i* of the maturity model corresponds with the *i-th* stage of the themes used to characterize and evaluate the software evaluation technology maturity.

IV. Work Assumptions

This section enumerates the assumptions taken into account while defining the maturity model.

These assumptions (and their corresponding themes) were derived by analysing a) existing maturity models and abstracting their main concepts; and b) evaluations performed in the semantic research field, focusing on the technologies that support these evaluations.

Assumption 1. Software evaluation is facilitated by a well-defined software evaluation workflow and such well-defined workflow will very likely yield quality evaluation results. Furthermore, having both a common framework for defining software evaluations and the means for easily defining evaluation workflows and also automating them is a significant factor that contributes greatly to the improvement of such workflows. Therefore, the following theme is important: *Formalization of the evaluation workflow*.

Assumption 2. Automation of software evaluation tasks enables to perform cost-efficient software evaluations and to diminish manual errors in them. Furthermore, the only way of managing and processing large quantities of software evaluation data is through the use of dedicated evaluation infrastructures. Therefore, the following theme is important: *Software support to the evaluation.*

Assumption 3. The quality of a software evaluation workflow depends on whether such workflow can be applied to different and heterogeneous types of software products. Only by applying the same evaluation workflow across different types of software and under different settings we can validate our hypotheses and conclusions. Therefore, the following theme is important: *Applicability to multiple software types*.

Assumption 4. The automated generation and manipulation of test data for software evaluations helps to focus on how to define test data instead of on how to manage them. Moreover, test data that can be used in different software evaluations are easier to understand, and the results obtained from them are easier to interpret. Therefore, the following theme is important: *Usability of test data*.

Assumption 5. Software evaluation results that are described in some machine-processable format enable the automated integration and exploitation of such results. This integration of results allows the compilation of a significant body of evaluation results, which leads to the exploitation of such results in unexpected ways. Therefore, the following theme is important: *Exploitability of results*.

Assumption 6. The quality of a software evaluation increases when different teams with different viewpoints define and support such evaluation. A software evaluation is more respected when it is supported not by a single team or organization (e.g., the software developers) but by multiple teams (e.g., software providers, users) or by the whole research field. Therefore, the following theme is important: *Representativeness of participants*.

V. Maturity Levels of Software Evaluation Technology

This section describes the five maturity levels of the Software Evaluation Technology Maturity Model. These levels and the goals to be achieved in each of them for a defined theme are presented in Table I.

What follows next is a general description of each level.

A. Level 1. Initial

At this level, a single team defines and carries out the evaluation workflow, which is specific to certain evaluation settings, is informally defined, and is manually performed with no software support. Evaluation is applied to a small number of software products of the same type, using test data not formally defined. The results obtained in the evaluation are also informally described, which makes them impossible to verify.

B. Level 2. Repeatable

At this level, the evaluation workflow is completely defined by one or a few teams and, although it is repeatable, it is still specific of certain evaluation settings. Evaluation software has been developed to partially support the evaluation of a small number of software products of the same type; these software products require to implement evaluation-specific mechanisms so they can be integrated with the evaluation software. The test data used in the evaluation are thoroughly described, and the evaluation results are defined in a machine-processable format, which allows combining the results of the evaluated software products.

C. Level 3. Reusable

At this level, several teams define an evaluation workflow that completely covers one type of software products and that can be reused to evaluate with different test data different

TABLE I
LEVELS AND THEMES OF SOFTWARE EVALUATION TECHNOLOGY MATURITY

Level	Formalization of the evaluation workflow	Software support to the evaluation	Applicability to multiple software types	Usability of test data	Exploitability of results	Representativeness of participants
Initial	Ad-hoc workflow informally defined.	Manual evaluation. No software support.	Small number of software products of the same type.	Informally defined.	Informally defined. Not verifiable.	One team.
Repeatable	Ad-hoc workflow defined.	Ad-hoc evaluation software.	Small number of software products of the same type. Ad-hoc access to software products.	Defined.	Machine-processable. Combined for some software products of the same type.	One or few teams.
Reusable	Technology-specific workflow defined.	Reusable evaluation software: - multiple software products. - multiple test data.	Multiple software products of the same type. Generic access to software products.	Machine-processable.	Machine-processable. Combined for many software products of the same type.	Several teams.
Integrated	Generic workflow defined. Machine-processable and built reusing common parts. Evaluation resources built upon shared principles.	Evaluation infrastructure: - multiple types of software products. - multiple test data.	Multiple software products of different types. Generic access to software products.	Machine-processable. Reused across evaluations.	Machine-processable. Combined for many software products of different types.	Several teams. Stakeholders.
Optimized	Generic workflow defined. Machine-processable and built reusing common parts. Evaluation resources built upon shared principles. Measured and optimized.	Federation of evaluation infrastructures: - autonomous infrastructures. - interchange of evaluation resources. - data access and use policies.	Multiple software products of different types. Generic access to software products. Support any software product requirement.	Machine-processable. Reused across evaluations. Customizable, optimized and curated.	Machine-processable. Combined for many software products of different types. High availability and quality.	Community.

characteristics of such software products. This workflow is supported by evaluation software that can be used to assess any software product of the type covered by the evaluation; the software product must have previously implemented the required mechanisms to be integrated with the evaluation software. Test data and evaluation results are machine-processable; therefore, they can be reused. Furthermore, the results can be combined for all the software products of the same type.

D. Level 4. Integrated

At this level, several teams in collaboration with relevant stakeholders (e.g., users or providers) define a generic evaluation framework that can be used with any type of software product. This generic framework for software evaluation allows building evaluation resources (i.e., evaluation workflow, tools, test data, and results) upon shared principles and reusing common parts. Here, evaluation workflows are defined in a machine-interpretable format so they can be automated. An evaluation infrastructure gives support both to the evaluation of multiple types of software products, taking into account their different characteristics, and to the management of the different evaluation resources. Test data can be reused across different evaluations, and the evaluation results can be combined for software products of different types.

E. Level 5. Optimized

At this level the whole community has adopted a generic framework for software evaluation in which evaluation workflows are measured and optimized. The centralized scenario of the previous levels has now evolved into a federation of autonomous evaluation infrastructures. These evaluation infrastructures must support not only the evaluation workflow but also new requirements, such as the interchange of evaluation resources or the implementation of policies for data access, interchange, and use. This federation of infrastructures permits satisfying any software or hardware requirements of the different software products; customizing, optimizing, and curating test data; and improving the availability and quality of the evaluation results.

One of the notions behind the maturity model, as Figure 2 shows, is that a higher maturity level implies higher integration of evaluation efforts in one field, ranging from isolated evaluations in the lower maturity level to fully-integrated evaluations in the higher level. In this scenario, maturity evolves from a starting point of decentralized efforts into centralized infrastructures and ends with networks of federated infrastructures.

Another notion to consider in this model is that of cost. While the cost of defining new evaluations decreases when the maturity level increases, mainly due to the reuse of existing resources, the cost associated to the evaluation infrastructure (hardware and infrastructure development and maintenance) significantly increases.

VI. ASSESSMENTS IN THE SEMANTIC RESEARCH FIELD

This section presents how we have used SET-MM to assess the maturity of software evaluation technologies in a specific research field.

Other maturity models provide appraisal methods for comparing with the maturity model. However, we do not propose any appraisal method because our scope is a whole research field and, therefore, it would be difficult to obtain objective metrics since any judgment would be subjective.

Therefore, our approach has been, first, to identify some evaluation efforts that stand out because of their impact in the field and, second, to try to assess the maturity of the software evaluation technologies used in them.

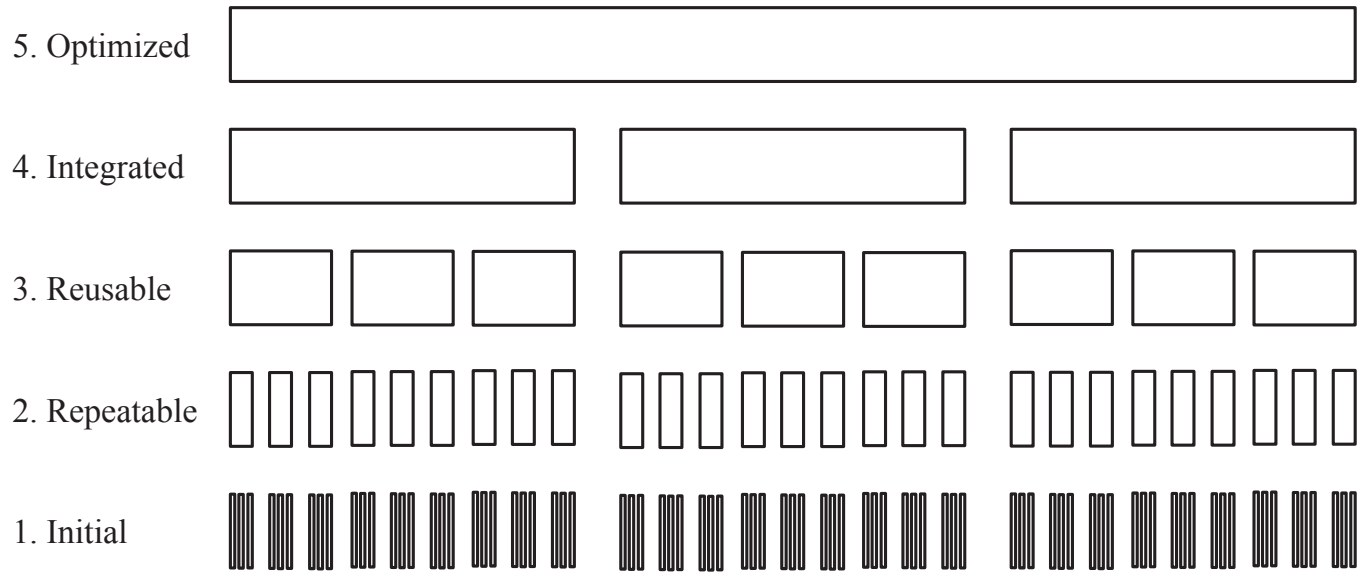


Fig. 2. The SET-MM maturity levels.

A. The Semantic Research Field

The notion of Semantic Web appeared at the beginning of the 21st century [17] and, since then, it has become a research field on its own.

The idea behind the Semantic Web is to have mechanisms to express knowledge and data in the Web so that it can be properly exploited by computers in an automated way.

The way of expressing such knowledge is through ontologies (explicit, structured models of the terminology and conceptual structures used in an application domain), which provide the basis for interpreting and relating data from different sources and that can be used to derive implicit knowledge about data.

One key requirement to make the Semantic Web real is the availability of technologies capable of managing and processing these data at web scale. Because of this, a huge number of research efforts have been devoted to the development and evaluation of semantic technologies.

B. Maturity of Evaluation Efforts

Evaluations in the semantic research field, as in any other field of research, are highly frequent and their main goal is to validate research.

However, if we analyse the technologies used to support those evaluations, we can see that in most of the cases the maturity of such evaluation technologies is at the *Initial* level, which makes almost impossible to reproduce any evaluation.

Nevertheless, there are still some efforts we should highlight because of their maturity in terms of software evaluation technologies.

1) The Lehigh University Benchmark: The Lehigh University Benchmark (LUBM) [18] is the benchmark most used in the semantic field. This benchmark can be used to evaluate the efficiency of ontology storage and reasoning systems and it is composed of a synthetic generator of test data, a set of test queries to be issued to the system, and a test module to automate execution.

LUBM's evaluation technology is at the *Repeatable* level in most of the themes, except in the next two aspects, in which it improves.

The usability of test data is between the *Reusable* and the *Integrated* levels, since the clear definition of test data and their automated generation largely facilitates their reuse across evaluations.

Additionally, the representativeness of participants is at the *Integrated* level because, even if the benchmark was defined by a research group, it is nowadays largely used by researchers and companies.

This high test data maturity can also be observed in the use of LUBM in the field: most of the people that reuse the benchmark only reuse the test data and define their own evaluation settings; furthermore, posterior benchmark improvements have mainly been made on test data [19], [20].

2) The Ontology Alignment Evaluation Initiative: The Ontology Alignment Evaluation Initiative[2] (OAEI) is an international initiative that, since 2004, has been organizing different ontology alignment contests with the goal of establishing a consensus for evaluating ontology alignment methods and their associated tools.

In these contests, ontology alignment systems are compared using a common set of synthetic and real-world tests using a common evaluation framework.

OAEI's evaluation technology is at the *Reusable* level in all themes except one, in which it improves. The representativeness of the participants is at the *Optimized* level; since the OAEI gathers the main stakeholders in the ontology alignment

[2]http://oaei.ontologymatching.org/

topic, its evaluations have become the de facto standard for ontology alignment evaluation.

3) The SEALS Platform: The SEALS Platform [21] is an infrastructure for the evaluation of semantic technologies that offers independent computational and data resources for the evaluation of these technologies.

The SEALS Platform is currently under development in a European project[3] and its goal is to provide the semantic field with an evaluation infrastructure at the *Integrated* level by the end of the project.

To this end, the SEALS Platform provides a common evaluation framework, based on the reusability of evaluation resources, in which different types of semantic technologies can be evaluated. Once the cumulative evaluation results reach a critical mass, the research community will be able to exploit them in novel ways.

VII. Conclusions and Future Work

This paper has presented the SET-MM maturity model that defines the maturity of software evaluation technologies in six different themes.

This maturity model can be used by anyone in a research field willing to assess the maturity of software evaluation technologies, either in the whole field or in specific evaluation initiatives.

SET-MM permits not only to make this assessment but also to guide improvement processes on software evaluation technologies by identifying the different goals to be achieved in each level and for each theme.

This way, the maturity model can be a useful mechanism to increase awareness on the role and on the benefits of software evaluation technologies in research fields.

To illustrate the use of the maturity model, we have assessed the maturity of the software evaluation technologies used in different initiatives within the semantic research field.

Clearly, this assessment is neither exhaustive nor representative of the whole research field. However, it is useful to analyse how software evaluation technologies have been used in different efforts in order to extract those lessons that are worth learning.

CMMI classifies maturity model components into three different categories, namely, required (specific and generic goals), expected (those practices relevant for achieving a goal) and informative (informative material that helps understanding the model). SET-MM currently covers the required components. Future work will deal with the definition of the expected and informative components in order to enrich the model.

Acknowledgment

This work is supported by the SEALS European project (FP7-238975) and by the EspOnt project (CCG10-UPM/TIC-5794) co-funded by the Universidad Politécnica de Madrid and the Comunidad de Madrid. Thanks to Rosario Plaza for reviewing the grammar of this paper.

[3]http://www.seals-project.eu/

References

[1] M. C. Paulk, B. Curtis, M. B. Chrissis, and C. V. Weber, "Capability Maturity Model for Software, Version 1.1," Software Engineering Institute, Tech. Rep. CMU/SEI-93-TR-024, February 1993.

[2] R. Sowden, D. Hinley, and S. Clarke, "Portfolio, Programme and Project Management Maturity Model (P3M3) Version 2.1. Introduction and Guide to P3M3," Office of Government Commerce, United Kingdom, Tech. Rep., 2010.

[3] S. Marshall, "E-Learning Maturity Model Version Two: New Zealand Tertiary Institution E-Learning Capability: Informing and Guiding E-Learning Architectural Change and Development," New Zealand Ministry of Education, Tech. Rep., 26th July 2006.

[4] V. R. Basili, "The role of experimentation in software engineering: past, current, and future," in *Proceedings of the 18th International Conference on Software Engineering (ICSE 1996)*. Berlin, Germany: IEEE Computer Society, 1996, pp. 442–449.

[5] CMMI Product Development Team, "CMMI for Acquisition, Version 1.3," Software Engineering Institute, Tech. Rep. CMU/SEI-2010-TR-032, November 2010.

[6] ——, "CMMI for Development, Version 1.3," Software Engineering Institute, Tech. Rep. CMU/SEI-2010-TR-033, November 2010.

[7] ——, "CMMI for Services, Version 1.3," Software Engineering Institute, Tech. Rep. CMU/SEI-2010-TR-034, November 2010.

[8] I. Burnstein, T. Suwanassart, and R. Carlson, "Developing a testing maturity model for software test process evaluation and improvement," in *Proceedings of the IEEE International Test Conference on Test and Design Validity (ITC 1996)*. Washington, DC, USA: IEEE Computer Society, 1996, pp. 581–589.

[9] F. Niessink and H. van Vliet, "Towards mature measurement programs," in *Proceedings of the 2nd Euromicro Working Conference on Software Maintenance and Reengineering (CSMR 1998)*. Florence, Italy: IEEE Computer Society, March 8–11 1998, pp. 82–88.

[10] M. YongGang and D. JianJie, "Software measurement process capability maturity model," in *Proceedings of the Second International Conference on Computer Modeling and Simulation (ICCMS 2010)*. Sanya, China: IEEE Computer Society, 2010, pp. 400–402.

[11] M. K. Daskalantonakis, R. H. Yacobellis, and V. R. Basili, "A method for assessing software measurement technology," *Quality Engineering*, vol. 3, no. 1, pp. 27–40, 1990-91.

[12] T. Wettstein and P. Kueng, *Management Information Systems 2002 – GIS and Remote Sensing*. Southampton: WIT Press, 2002, ch. A Maturity Model for Performance Measurement Systems, pp. 113–122.

[13] J. Z. Gao, H.-S. J. Tsao, and Y. Wu, *Testing and Quality Assurance for Component-Based Software*. Norwood, MA, USA: Artech House, Inc., 2003.

[14] *ISO/IEC 14598-6: Software product evaluation - Part 6: Documentation of evaluation modules*. ISO/IEC, 2001.

[15] R. García-Castro, M. Esteban-Gutiérrez, M. Kerrigan, and S. Grimm, "An ontology model to support the automatic evaluation of software," in *Proceedings of the 22nd International Conference on Software Engineering and Knowledge Engineering (SEKE 2010)*. Redwood City, CA, USA: Knowledge Systems Institute, July 1–3 2010, pp. 129–134.

[16] F. Niessink and H. van Vliet, *A Pastry Cook's View on Software Measurement*. Wiesbaden, Germany: Deutscher Universitaetsverlag, 1998, pp. 109–126.

[17] T. Berners-Lee, J. Hendler, and O. Lassila, "The Semantic Web," *Scientific American*, vol. 284, no. 5, pp. 34–43, 2001.

[18] Y. Guo, Z. Pan, and J. Heflin, "LUBM: A Benchmark for OWL Knowledge Base Systems," *Journal of Web Semantics*, vol. 3, no. 2, pp. 158–182, 2005.

[19] L. Ma, Y. Yang, Z. Qiu, G. Xie, Y. Pan, and S. Liu, "Towards a complete OWL ontology benchmark," in *Proceedings of the 3rd European Semantic Web Conference (ESWC 2006)*, ser. LNCS, vol. 4011. Budva, Montenegro: Springer, June 11-14 2006, pp. 125–139.

[20] T. Weithöner, T. Liebig, M. Luther, S. Böhm, F. von Henke, and O. Noppens, "Real-world reasoning with OWL," in *Proceedings of the 4th European Semantic Web Conference (ESWC2007)*, ser. LNCS, vol. 4519. Springer, 2007, pp. 296–310.

[21] R. García-Castro, M. Esteban-Gutiérrez, and A. Gómez-Pérez, "Towards an infrastructure for the evaluation of semantic technologies," in *Proceedings of the eChallenges 2010 Conference*, P. Cunningham and M. Cunningham, Eds., Warsaw, Poland, October 27-29 2010.

A Feature-Based Modeling Approach to Configuring Privacy and Temporality in RBAC

Sangsig Kim[1], Yen-Ting Lee[1], Yuanlin Zhu[1], Dae-Kyoo Kim[1], Lunjin Lu[1], and Vijayan Sugumaran[2]
[1]Department of Computer Science and Engineering
[2]Department of Decision and Information Sciences
Oakland University
Rochester, MI 48309
{skim2345,ylee2,yzhu2,kim2,l2lu,sugumara}@oakland.edu

Abstract

Role-Based Access Control (RBAC) has been increasingly popular due to its efficiency, flexibility, and scalability. Traditionally, RBAC is concerned with Separation of Duty (SoD) among roles and role hierarchies. However, there have been demands for extensions of RBAC as environments of RBAC systems have changed. As part of response to the demands, privacy RBAC and temporal RBAC have been proposed. While the two extensions address different aspects, they are often needed together in many systems such as hospital systems. In this paper, we present a feature-based approach that enables systematic enforcement of combined privacy and temporal RBAC in development. The approach models the two extensions as features based on partial inheritance which supports verifiable feature composition. We demonstrate the approach using a hospital example.

1 Introduction

Role-Based Access Control (RBAC) [5] is a popular access control model concerning Separation of Duty (SoD) and role hierarchies. As the environment of RBAC systems has changed dramatically, there is a strong demand for extending the traditional RBAC. For instance, in the environment where data objects contain private information, privacy should be protected. If the environment is time-sensitive, time-related policies should be supported. In response to the demands, privacy RBAC (PRBAC) [12] and temporal RBAC (TRBAC) [1] have been proposed. While each PRBAC and TRBAC addresses a well-defined aspect, there are domains where both PRBAC and TRBAC are desired. For instance, in the hospital domain, a doctor should not be granted access to the records of patients who are not assigned to him/her, and even if the patient is his patient, the doctor should not be able to access the patient's record other than his/her working hours. In the banking domain, access to customer accounts by a teller should be allowed only when there is a transaction request during his/her working hours.

In this paper, we present an approach that enables systematic enforcement of combined PRBAC and TRBAC during development. In our approach, PRBAC and TRBAC are modeled as features [8] based on partial inheritance [9], which enables verifiable feature composition. We use the UML and the Object Constraint Language (OCL) for defining the semantics of PRBAC and TRBAC. We demonstrate the approach using a hospital example and discuss tool support developed for the approach.

The rest of the paper is organized as follows. Section 2 gives an overview of related work. Section 3 presents the designs of privacy and temporal features based on partial inheritance. Section 5 describes how PRBAC and TRBAC can be used for the development of a hospital application through composition and instantiation. Section 5 concludes the paper.

2 Related Work

Several researchers describe access control models using the UML. The work can be categorized into two approaches. One is using the UML as is. Shin and Ahn use UML class diagrams to describe the structure of RBAC and the OCL to define RBAC constraints [15]. Our previous work uses object diagrams to visualize RBAC constraints [10]. Priebe *et al.* view an access control model as a design pattern and use the Gang-of-Four (GoF) pattern template to describe it [13]. The other approach uses UML profiles, an extension mechanism in the UML, for defining access control concepts. Jurjens proposed a UML profile called

UMLsec for modeling and evaluating security aspects for distributed systems based on the multi-level security model [7]. Similarly, Lodderstedt *et al.* proposed a UML profile called SecureUML for defining security concepts based on RBAC [11]. Doan *et al.* extend the UML, not by a profile, but by directly incorporating security aspects of RBAC and MAC into UML model elements [3].

Composition of RBAC features in this work is related to model composition in aspect-oriented modeling (AOD) (e.g., [2, 14, 17, 18]). In AOD, cross-cutting concerns are designed as design aspects that are defined separately from functional aspects (referred to as primary models). Clarke and Walker proposed composition patterns to compose design aspects described in UML templates with a primary model through parameter binding [2]. Straw *et al.* proposed a set of composition directives (e.g., creating, adding) for aspect composition [18]. Similar to Clarke and Walker's work, Reddy *et al.* use sequence diagram templates for specifying behaviors of design aspects and use tags for behavior composition [14]. An aspect may include position fragments (e.g., begin, end) which constrain the location of fragment interactions to be inserted in a sequence diagram. The composition method in their work, however, is not rigorous, and thus it is difficult to verify resulting models. Their position fragments influenced join points in our work. Song *et al.* proposed a composition method for composing a design aspect with an application design [17]. They verify composed behaviors described in OCL by discharging a set of proof obligations. However, their verification is limited to OCL expressions, and the entire composed model cannot be verified.

3 Modeling PRBAC and TRBAC as Features

In this section, we design PRBAC and TRBAC as feature based on partial inheritance [9]. Partial inheritance designs RBAC features in terms of a core feature and several component features. The *Core* feature [9] captures the essential functions of RBAC that every RBAC system must possess, while component features inherit from the *Core* feature and define additional functions. Unlike the traditional inheritance, component features inherit only those properties that are needed to carry out its functions, which establish partial inheritance. This simplifies feature design and composition.

Privacy feature. The *Privacy* feature provides support for privacy-related policies. Fig. 1 shows the feature where the symbol "|" denotes parameters.

In addition to the inherited classes (*User*, *Role*, *Session*, *Operation*, *Object*, *Permission*, and *Reference Monitor*) from the *Core* feature, the *Privacy* feature includes concepts of *Purpose*, *DataCondition*, and *Obligation*. *Purpose*

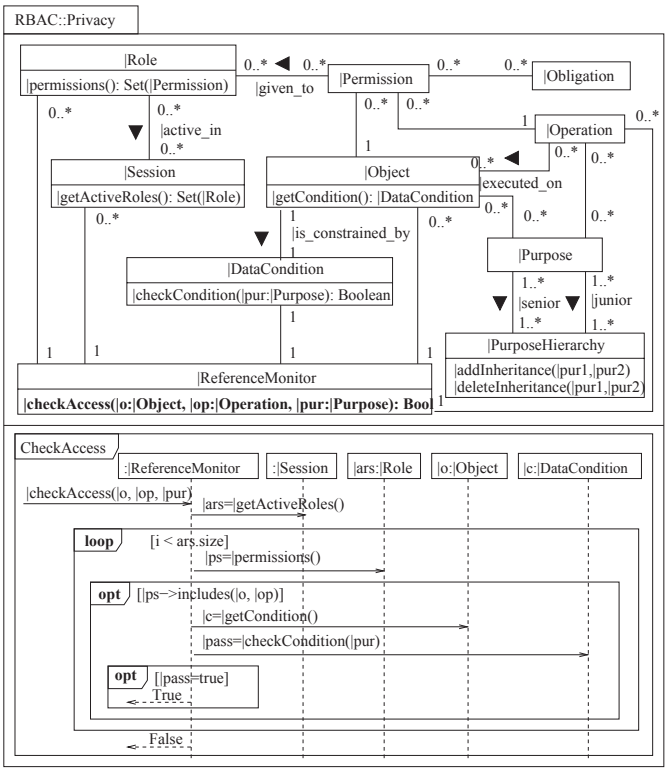


Figure 1. Privacy Feature

is the intention to perform a task that causes access to privacy information. The purpose can be organized according to the hierarchical relationships to simplify the management of purposes since purposes by nature have a hierarchical relationships among them in common business environments. *DataCondition* describes restrictions/qualifications for access to privacy information and the purpose of information use. *Obligation* specifies additional protection measures to personally identifiable information.

These privacy concepts enable privacy policies to be better supported at the modeling level. Suppose that an account receivable clerk requests for access to read customer contact information for sending a bill to customers. This policy can be specified using the template in [4] as follows:

"*Allow [Accounts Receivable Clerk] to perform [Read] on [Customer Contact Information] for [Billing] provided [Billing is the Data Purpose for Customer Contact Information]. Carry out [immediate logging of access and deleting of banking information from customer information for customers with no billing activity for the last 12 months].*" where *Accounts Receivable Clerk* represents role, *Read* represents operation, *Customer Contact Information* represents object, *Billing* represents purpose, *Billing . . . Information* describes data condition,

and *immediate . . . months* describes obligation.

The *checkAccess()* operation in the *ReferenceMonitor* class is redefined, which is denoted in bold, to check data purpose of the object being accessed. The new semantics is defined as follows:

```
1. context ReferenceMonitor::checkAccess (o:Object,
2.                                op:Operation,pur:Purpose):Boolean
3.    pre : true
4.    post: let act:OclMessage = Session^getActiveRoles()
5.    in Act: act.hasReturned() and act.result() = acrs and
6.      let pss:Sequence(OclMessage) =
7.          acrs → collect(r | r^^permission())
8.      in if ((pss → notEmpty()) and
9.          (pss → includes (ps|ps.result()→includes(o,op))))
10.          let cond:OclMessage = o:Object^getCondition()
11.          in Cond: cond.hasReturned() and
12.             cond.result() = c and
13.             result = c:DataCondition^checkCondition(pur)
14.     endif
```

In the semantics, privacy-specific constraints are specified in lines 12-14 checking access purpose against data condition. The sequence diagram *CheckAccess* in Fig. 1 describes checking accessibility based on privacy constraints. In addition to the traditional RBAC checking, the *Privacy* feature checks if the purpose of access matches the data condition. If it does not match, access is denied.

Temporal feature. The *Temporal* feature supports temporal constraints on user-role assignments, role-permission assignments, role hierarchies, and SoD. Fig. 2 shows the feature.

In the feature, the *TemporalConstraint* class is responsible for enforcing temporal constraints on enabling and disabling roles and activating and deactivating roles in session during a specified period or duration. Two types of time constraints are supported by the feature, namely periodicity and duration, which is captured by the hierarchy of temporal constraints in Fig. 2. A periodic constraint is a periodic time interval expressed in terms of (begin time, end time, periodic expression). For example, the policy that the working time of a day-duty doctor starts from 9 am to 5 pm everyday can be specified as *Enable [day-duty doctor] from [9 am] to [5 pm], [Everyday]*. A duration constraint specifies a specific time duration. For example, "*After [1 hours] of [day-duty nurse enabled], enable [nurse on training] for [4 hours]*" describes that a nurse on training is enabled to work 4 hours during day time after 1 hour working of a day-duty nurse. Both constraints can be used in role enabling/disabling, user-role assignments, or role-permission assignments. However, duration constraints can only be

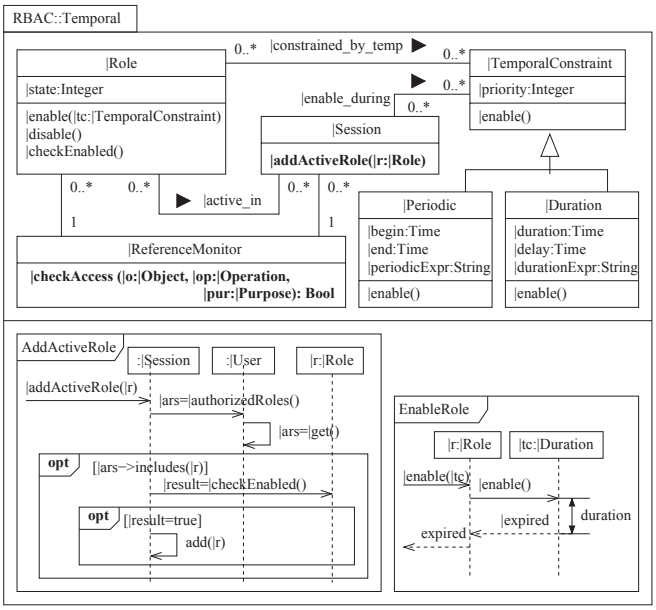


Figure 2. Temporal Feature

specified for role activation/deactivation as a user can activate or deactivate a role at any time [4].

Role hierarchies with temporal constraints can be categorized into *unrestricted* hierarchies, *enabling time restricted hierarchies*, and *activation time restricted hierarchies* based on the effect of temporal constraints. Unrestricted hierarchies are not affected by temporal constraints. In enabling time restricted hierarchies, when a senior role is enabled for a specific time, the permissions of its junior roles are also acquired. Enabling time restricted hierarchies can be either weak or strong. In weak hierarchies, a senior role can acquire permissions of its junior roles without the junior roles being enabled, while in strong hierarchies, the junior roles must be also enabled in order for the senior role to acquire junior roles' permissions. In activation time restricted hierarchies, a junior role can be activated only at the time when its senior roles are activated [6]. Both periodicity and duration place time contraint on SoD in the traditional RBAC.

The *Temporal* feature redefines the *CheckAccess()* operation inherited from the *Core* feature and requires that the requested role be enabled and activated before access is granted. The sequence diagram *AddActiveRole* in Fig. 2 describes role activation in a session. It redefines the *AddActiveRole* sequence diagram in the *Core* feature by adding checking the enable/disable status of the requested role before it can be activated in the session. Note that the *AddActiveRole* scenario does not take into account role hierarchies as role hierarchy is defined as a separate feature (see [9] for the *Hierarchy* feature). The role hierarchy version can be

built by configuring the *Temporal* feature with the *Hierarchy* feature. The *EnableRole* sequence diagram describes role enabling during a specific time.

4 Feature Composition

We use the composition method in our previous work [9] for composing the *Privacy* and *Temporal* features when they are configured together. Syntactically, the composition method combines the properties of configured features into the composed feature based on name matching. Semantically, the behavioral properties that have the same name are composed to be refinement of the behaviors that were composed. The formal definition of refinement for operations and interaction behaviors can be found in [9]. Using the definition, one can reason about the resulting feature for correctness.

Built upon partial inheritance, the composition method also enables step-wise feature composition, allowing immediate evaluation of the impact of composition. The *Core* feature, which is the base of any RBAC configuration, is selected by default forming the first configuration and the n^{th} configuration is built upon the $(n-1)^{th}$ configuration. We view this approach as a special kind of multiple inheritance where the elements having the same name get composed rather than renamed as in the traditional multiple inheritance.

5 Case Example: Hospital Application

In this section, we demonstrate how the resulting feature from composition can be used for designing a hospital application via instantiation. The application is used by day-duty doctors, night-duty doctors, day-duty nurses, night-duty nurses, nurses on training and medical students to perform read/write operations on privacy information such as patients' medical records. Fig. 3 shows the privacy and temporal policies of the hospital system.

Role	Read Access	Write Access	Purpose	Time
Day Resident Phys.	All documents	Surgical Report	Treatment	9 am − 5 pm
Night Resident Phys.	All documents	Surgical Report	Treatment	2 am − 9 pm
Non−resident Phys.	Patient Presc. Order, Progress Notes, Equipment Order	Surgical Report, Patient Presc. Order,	Treatment	9 am − 5 pm
Day Reg. Nurse	Progress Notes, Equipment Order	Surgical Report, Progress Notes, Patient Presc. Order,	Attendance	0 hour after Day Resident Phys.
Night Reg. Nurse	Progress Notes, Equipment Order	Surgical Report, Progress Notes, Patient Presc. Order,	Attendance	0 hour after Night Resident Phys.
Nurse in Training	Progress Notes, Equipment Order	Progress Notes, Patient Presc. Order,	Attendance	1 hour after Day Reg. Nurse
Medical Student	Surgical Report	None	Research	9 am − 5 pm

Figure 3. Permissions

Based on the privacy templates and the temporal templates in [4], we present templates for describing both privacy and temporality as follows:

- **Privacy-periodic events.** *Allow [Role] to perform [Operation] on [Object] for [Purpose] provided [DataCondition]. Carry out [Obligation] from [begin time] to [end time], [Periodic Expression]*
- **Privacy-duration events.** *Allow [Role] to perform [Operation] on [Object] for [Purpose] provided [DataCondition]. Carry out [Obligation] after [delay time] of [event].*

Using the templates, the privacy and temporal policies of the hospital system can be described as follows:

- Allow [Day Resident Physician] to perform [Read] on [All Documents] for [Treatment Purpose] provided [The doctor is the patient's on-duty doctor] carry out [Logging of access and closing the patient's case if the patient is transferred to other hospital] from [9:00 am] to [5:00 pm]
- Allow [Day Registered Nurse] to perform [Write] on [Surgical Report, Progress Notes and Patient Prescription Order] for [Attendance Purpose] provided [The nurse is the patient's on duty nurse] carry out [Logging of Access] after [0 hour] of [Enabling Day Resident Physician]

First Configuration. In order to support the above policies, the system is configured with the *Privacy* feature and the *Temporal* feature together on top of the *Core* feature, which is the first configuration. The configured features are composed using the RBAC composition method in Section 4 to produce configured RBAC. Enabled by partial inheritance, features can be composed in any order. In this demonstration, we compose features in the order of *Core*, *Privacy*, and *Temporal*.

Second Configuration. The *Core* and *Privacy* features are composed, which results in the second configuration, by (1) adding the *Purpose*, *Purpose Hierarchy*, *Obligation*, and *DataCondition* classes and their associated relationships to the core feature, (2) composing the *Role*, *Session*, *ReferenceMonitor*, *Permission*, *Object*, and *Operation* classes which appear in both features, and (3) composing the matching operations (e.g., *checkAccess()*) in both features. Based on the composition method, the semantics of composed operations must refine both core operations and privacy operations that were involved in composition, which can be formally verified using the refinement definition in [9].

For sequence diagrams, the *CheckAccess* sequence diagram in the *Core* feature is composed with the corresponding sequence diagram in the *Privacy* feature. This results in the *O:Object* and *c:DataCondition* lifelines in the *Privacy* feature being added to the *Core* feature. The *O:Object* lifeline is responsible for obtaining data purpose from the requested object and the *c:DataCondition* lifeline checks if

the data purpose matches the purpose of the operation request.

Third Configuration. The second configuration is composed with the *Temporal* feature, which results in the final configuration in this demonstration. They are composed by (1) adding the hierarchy of temporal constraints involving the *TemporalConstraint*, *Periodicity*, and *Duration* classes and their associated relationships to the second configuration, and (2) composing the *Role*, *Session*, and *ReferenceMonitor* classes which appear in both features, and (3) composing the matching operations (e.g., addActiveRole()) in both features. The resulting class diagram is shown in Fig. 4. Ensured by the composition method, the composed operations in the final configuration refines both privacy operations and temporal operations.

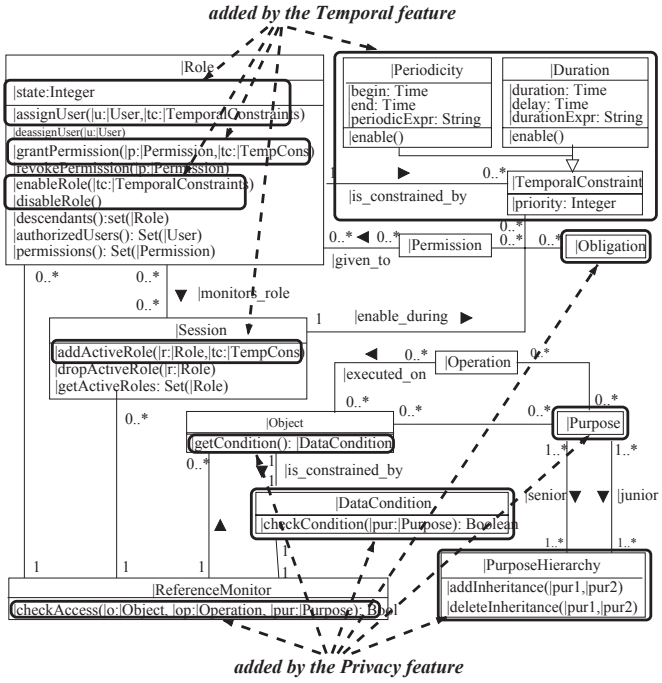


Figure 4. Partial Class Diagram after Third Configuration

For sequence diagrams, the *AddActiveRole* sequence diagram in the second configuration is composed with the corresponding sequence diagram in the *Temporal* feature. This results in adding the *tc:Duration* lifeline to the *AddActiveRole* sequence diagram in the second configuration. Fig. 5 shows the resulting diagram.

The *tc:Duration* lifeline is added to activate the requested role during a given duration. In addition, a role needs to be enabled before it is activated (checkEnabled()).

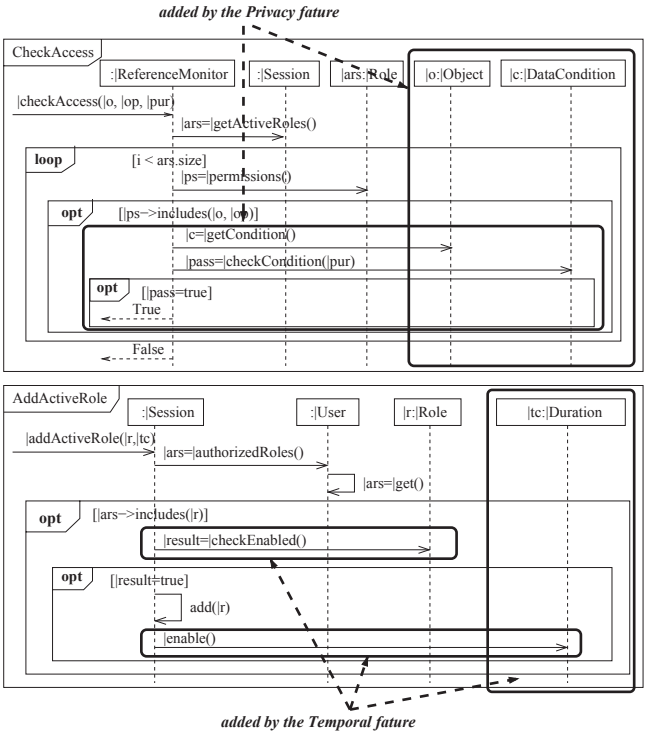


Figure 5. Partial Sequence Diagram after Third Configuration

Fig. 5 shows the composition of *CheckAccess* in *Core* and *Privacy* and *AddActiveRole* in the second configuration and *Temporal*. The composition of the second configuration and the *Temporal* feature results in the final configuration. Fig. 4 shows partial design of the final configuration.

The final configuration can be used for building a design of the hospital system via instantiation. Fig. 6 shows an instantiation of the final configuration in the context of the hospital application. The instantiation is carried out based on a mapping between RBAC elements and application concepts. For instance, the |*Object* and |*Operation* classes in RBAC are mapped, respectively, to the hospital objects such as the *Patient Prescription Order* and *Equipment Order* classes and transaction operations such as *Read* and *Write*. The instantiated model lends itself as an initial design model for the application addressing access control concerns.

6 Conclusion

We have described a feature-based modeling approach for designing PRBAC and TRBAC as features of RBAC to support systematic use of both PRBAC and TRBAC in the development of access control systems. The composition

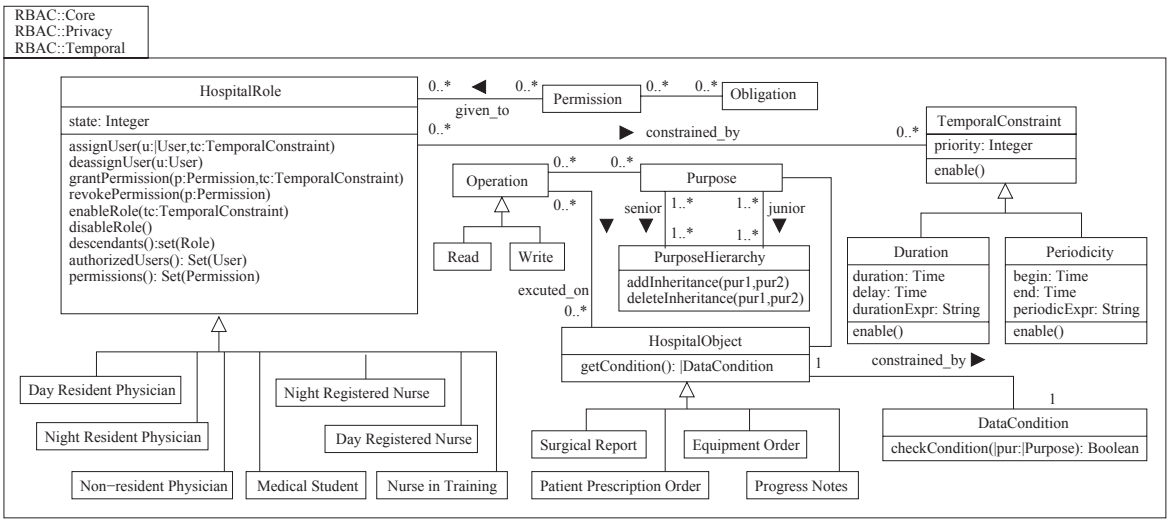


Figure 6. Partial Instantiation in Hospital Domain

in the paper involves only the *Core*, *Privacy*, and *Temporal* features for demonstration, but other traditional features such as the *Hierarchy* and *SoD* features may also be composed with the *Privacy* and *Temporal* features to account for the needs emerged from enivornment changes. Besides the hospital example presented in this paper, we have also carried out a case study using SmallSQL [16], a database management system. Other domains to which the presented approach can be applied include the banking domain and the academic domain where privacy and temporal issues are critical. We have developed a prototype that implements the composition method and feature instantiation based on Rational Software Architect and Eclipse.

References

[1] E. Bertino, P. Bonatti, and E. Ferrari. Trbac: A temporal role-based access control model. *ACM TISS*, (3):191–233, 2001.

[2] S. Clarke and R. Walker. Composition Patterns: An Approach to Designing Reusable Aspects. In *ICSE*, pages 5–14, 2001.

[3] T. Doan, S. Demurjian, C. Phillips, and T. Ting. Research Directions in Data and Applications Security XVIII. In *IFIP TC11/WG 11.3*, pages 25–28, 2004.

[4] D. Ferraiolo, D. Kuhn, and R. Chandramouli. *Role-Based Access Control*. Artech House, 2nd edition, 2003.

[5] D. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli. Proposed NIST Standard for Role-Based Access Control. *ACM TISS*, 4(3):224–274, 2001.

[6] J. Joshi, E. Bertino, U. Latif, and A. Ghafoor. A generalized temporal role-based access control model. *IEEE TKDE*, 17(1):4–23, 2005.

[7] J. Jurjens. UMLsec: Extending UML for Secure Systems Development. In *UML*, pages 412–425, 2002.

[8] K. Kang, S. Cohen, J. Hess, W. Nowak, and S. Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90TR-21, 1990.

[9] D. Kim, L. Lu, and S. Kim. A Verifiable Modeling Approach to Configurable Role-Based Access Control. In *FASE/ETAPS*, pages 188–202, 2010.

[10] D. Kim, I. Ray, R. France, and N. Li. Modeling Role-Based Access Control Using Parameterized UML Models. In *FASE/ETAPS*, pages 180–193, 2004.

[11] T. Lodderstedt, D. A. Basin, and J. Doser. SecureUML: A UML-Based Modeling Language for Model-Driven Security. In *UML*, pages 426–441, 2002.

[12] Q. Ni, E. Bertino, J. Lobo, C. Brodie, C. Karat, J. Karat, and A. Trombetta. Privacy-aware role-based access control. *ACM TISS*, 13(3):1–31, 2010.

[13] T. Priebe, E.B. Fernandez, J.I. Mehlau, and G. Pernul. A Pattern System for Access Control. In *IFIP WG 11.3*, 2004.

[14] R. Reddy, A. Solberg, R. France, and S. Ghosh. Composing Sequence Models using Tags. In *MoDELS Workshop on AOM*, 2006.

[15] M. Shin and G. Ahn. UML-Based Representation of Role-Based Access Control. In *Workshops on Enabling Technologies*, pages 195–200, 2000.

[16] SmallSQL. http://www.smallsql.de/.

[17] E. Song, R. Reddy, R. France, I. Ray, G. Georg, and R. Alexander. Verifiable Composition of Access Control and Application Features. In *SACMAT*, pages 120–129, 2005.

[18] G. Straw, G. Georg, E. Song, S. Ghosh, R. France, and J. Bieman. Model Composition Directives. In *UML*, 2004.

Using Security Patterns to Tailor Software Process

Rosana Wagner
Instituto Federal Farroupilha
Alegrete, Brasil
rosanawagner@gmail.com

Lisandra Manzoni Fontoura
Universidade Federal de Santa Maria
Santa Maria, Brasil
lisandramf@gmail.com

Adriano Brum Fontoura
Instituto Federal Farroupilha
Santo Augusto, Brasil
afontoura@sa.iffarroupilha.edu.br

Abstract — Secure software development processes can reduce the quantity of security errors and the vulnerabilities involved in software projects. A secure development process is composed by activities that propose the insertion of security requirements in all software development phases. These activities can be based on standards and/or security models such as SSE-CMM, ISO/IEC 27001, ISO/IEC 15408. The problem is that the standards and security models describe security requirements which can be followed but do not describe how these requirements must be implemented in software processes. Security patterns describe good security practices which can be incorporated to the software process and satisfy the requirements that are described by the standards and models. This work proposes a methodology for the tailoring of software processes based on security requirements that are defined by the security practices of the Systems Security Engineering Capability Maturity Model (SSE-CMM). The tailoring has as basis a process framework that is elaborated from the Rational Unified Process (RUP) and security patterns proposed on the literature.

Keywords - software processes, information security, processes tailoring, security patterns.

I. INTRODUCTION

The lack of security in software projects is one of the main concerns of the organizations. It is through the vulnerabilities that are present in software projects that the secrecy breaking and the information theft occur. Due to this factor, the organizations are seeking to adopt measurements that are more and more rigorous in terms of protection aligned with standards and security models [1].

The SSE-CMM model (System Security Engineering Capability Maturity Model) [2], currently known as the ISO/IEC 21827 standard, provides a set of good security practices which can be adopted by the organizations in order to increase the software security. The model is recommended for the development of secure software and the elaboration of security management processes.

The security patterns provide solutions that are already consolidated for recurrent problems and serve as reference for the organizations that seek to satisfy security requirements [3]. Thus, patterns can be associated to the SSE-CMM practices, identifying how they may be incorporated into a software process. The incorporation of the patterns into the software process happens at the moment of the process tailoring to answer to the specific needs of a project.

This article proposes an approach to processes tailoring that have as basis a process framework which is built from the Rational Unified Process and from security patterns association rules to the SSE-CMM process areas. The processes that are obtained from the framework aim at the development of secure software.

Tailoring processes consists in altering or adapting the descriptions of a process for a particular purpose. In this article, the tailoring is given by means of selecting and incorporating patterns into organization's software process, which is instantiated from the framework, originating the specific software process to use in a project. A methodology for processes tailoring that is based on the framework is proposed.

The contributions of this work include the methodology proposal for software processes tailoring based on security requirements, the framework for elaborating secure processes, which can be customized by the user, and a tailoring support tool.

II. SOFTWARE PROCESS SECURITY

Elaborating secure processes can be obtained by means of the security patterns incorporation, which describe consolidated solutions for recurrent security problems, that are associated to the practices of the SSE-CMM Model.

The System Security Engineering Capability Maturity Model (SSE-CMM) describes the essential characteristics that must exist in the processes of an organization in order to assure a good systems security by means of 22 process areas (PAs) that are organized in two groups: Security Base Practices and Organizational and Project Base Practices. In this work we considered the Security Base Practices because they are responsible for implementing security in software processes, which is the objective of this work.

Security patterns have been associated to security base practices because they provide solutions for recurrent security problems. The patterns capture the experience of security specialist individuals and provide solutions for security related problems which can be applied by non-specialist individuals. Associating security patterns to the base practices facilitate the processes tailoring, since once the association rules are defined it is not necessary that the process engineer is a security specialist.

The solution that is proposed by a pattern can be implemented with the help of other patterns, which solve parts of the whole problem [4]. The most known pattern catalogues are: Schumacher et al. [3], Rosado [5], Romanosky [7], Kienzle and Elder [8].

III. Elaborating the Security Methodology Tailoring (SMT) Framework

The processes tailoring that is proposed by this work is based on a process framework elaborated from Rational Unified Process. This framework is extended by adding a set of security related activities. The security related activities aims to satisfy the PAs of the SSE-CMM and have been defined from security patterns catalogues [3][5][7][8][9][10]. It is important to highlight that new activities can be added to the proposed framework according to the organization necessities.

The process for elaborating the SMT framework followed the undermentioned stages: elaborating rules to associate process areas to security patterns and incorporating these rules, security patterns and process areas to the framework.

A. Elaborating the Association Rules

Considering that the PAs have a list of objectives that indicate the expected results after its implementation and a list of BPs (Base Practices) that assist in the making of the objectives, the systematics followed to identify the patterns was the undermentioned: based on the objectives of the PA and its BPs, we have analyzed pattern catalogues to search for patterns, one or more, that addressed to these objectives. Examples provided in the PA description were also considered.

The association of each PA to the patterns that satisfy their objectives originates an association rule. As an example of an elaborate rule, we quote the usage of the AssetValuation [4]; ThreatAssessment [4]; VulnerabilityAssessment [4] and RiskDetermination [4] patterns for implementing PA03 – Assess Security Risk.

The purpose of Assess Security Risk is to identify the security risks involved with relying on a system in a defined environment. This process area focuses on ascertaining these risks based on an established understanding of how capabilities and assets are vulnerable to threats. Specifically, this activity involves identifying and the assessing the likelihood of the occurrence of risk exposures. The risk assessment is made in order to support the decisions related to the development, maintenance or system operation whose environment is known.

Table I shows the rules of association to the process areas that are elaborated in this work.

TABLE I – Associating Security Patterns to SSE-CMM Process Areas

Process Areas	*SuggestedPatterns*
PA01 – Administer Security Controls	*Security Provider [7];* *Controlled Process Creator [3];* *Access Control Requirements [3];* *Role Rights Definition [3];* *Role-Based Access Control [3];* *Authorization Pattern [5];* *Multilevel Security Pattern [5];*
PA02 – Assess Impact	*Risk Determination [3];*
PA03 – Assess Security Risks	*Asset Valuation [3];* *Threat Assessment [3* *Vulnerability Assessment [3];* *Risk Determination [3];*
PA04 – Assess Threats	*Threat Assessment [3];*
PA05 – Assess Vulnerabilities	*Vulnerability Assessment [3];*
PA06 – Build Assurance Argument	*Patch Proactively [8];* *Engage Customers [9];* *Check Point [10];* *Red Team the Design [8];*
PA07 – Coordinate Security	*Enterprise Partner Communication [3];* *Share Responsibility for Security [8];* *Gatekeeper [9];* *Buffalo Mountain (organizational) [9];*
PA08 – Monitor Security Posture	*Minefield [8];* *Security Accounting Requirements [3];* *Security Accounting Design [3];* *Audit Requirements [3];* *Audit Design [3];* *Audit Trails & Logging Requirements [3];* *Audit Trails & Logging Design [3];* *Non-Repudiation Requirements [3];* *Non- Repudiation Design [3];*
PA09 – Provide Security Input	*Document the Security Goals [8];* *Document the Server Configuration [8];* *Enterprise Security Approaches [3];* *Enterprise Security Services [3];*
PA10 – Specify Security Needs	*Security needs Identification for Enterprise Assets [3];*
PA11 – Verify and Validate Security	*Task Process Pattern – Technical Review [10];* *Check Point Pattern [5];* *Whitehat, Hack Thyself [7];* *Technical Guide to Information Security Testing and Assessment [11].*

The rules of association pattern to process areas are suggestions elaborated through the literature and may be improved by the organization by analyzing past projects, retrospective sessions, etc.

B. Incorporating the Association Rules to the Framework

After defining the association rules, it is necessary to incorporate them to the framework, as well as their constituent elements (security patterns and process areas).

So that security patterns may be incorporated into software processes, it is necessary that they are described by means of concepts that will be used in the processes modeling. Metamodels are used to describe elements that can be used for the process elaboration. In this work, we have decided to use the PRiMA-M (*Project Risk Management Approach – Metamodel*) metamodel (Fig. 1), which was elaborated in previous works by one of the authors [12]. PRiMA-M represents a set of concepts that are used to elaborate software processes. Process elements, instantiated from the PRiMA-M, can be used in the definition of planned, agile or hybrid processes.

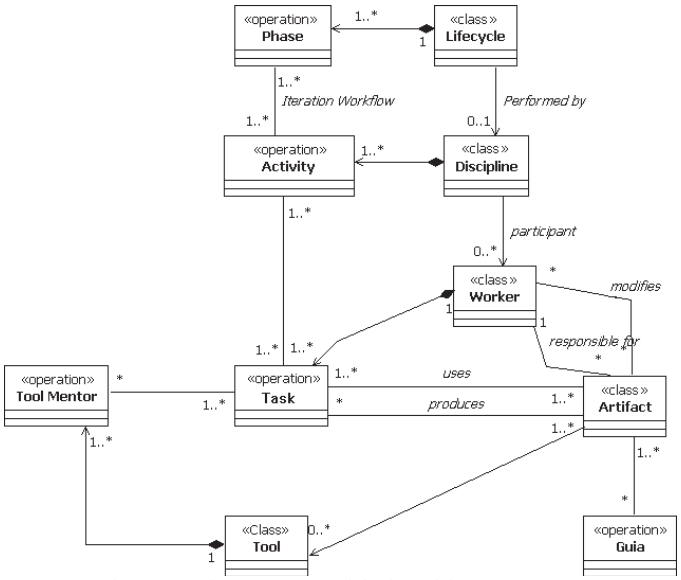


Figure 1. PRiMA – Metamodel adapted from [12][13]

This metamodel was elaborated from the concepts described in the Rational Unified Process. The following is an explanation about the classes of this metamodel. More information can be obtained in [12][13].

Lifecycle of a process is an aggregation of *Phases* which, in turn, are associated to *Activities*. A *Discipline* identifies a set of *Workers* which participate in the *Discipline* and define a set of *Activities* that compose the *Discipline*. An *Activity* specifies a particular collaboration within a *Discipline* and represents *Tasks* grouping, given that a *Worker* is responsible for each *Task*.

Artifacts are products generated during the execution of *Tasks* and may be models, plans, software versions, reports, etc. *Workers* represent the roles executed by individuals in a project. *Tools* are used to assist the making of tasks. *ToolsMentors* describe how to execute a *Task* by using a certain *Tool*.

In order to exemplify how security patterns can be described by means of process elements, instantiated from the PRiMA-M, the AssetValuation pattern, described by using the process elements proposed in the framework, can be visualized in the Fig. 2.

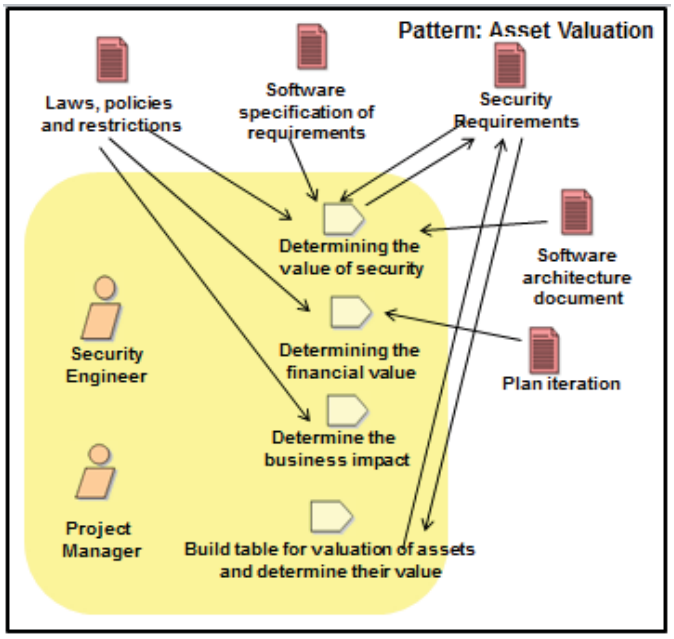


Figure 2. *AssetValuation* pattern represented by using process elements proposed in PRiMA-M

The security engineer executes the tasks with the assistance of the project manager. The main artifact elaborated during the execution of these tasks is the *Security Requirements* document which describes the assets along with the security value, financing value and the impact that the asset may have in the business, as well as the assessment tables that were elaborated for the asset assessment.

Each pattern that is described in an association rule has been defined by using the process elements proposed in the metamodel. The execution sequence of the activities has been defined by means of an activity diagram proposed by the UML. In the RUP, activity diagrams are proposed for each discipline in order to organize the activities that are possible to be executed in the software processes instantiated from it.

Considering that disciplines in the RUP seek to group activity collections related to a concentration area, we opted to group the proposed activities in order to add security to the RUP in one single discipline, named "Software Security".

Another alternative would be to set activities related to security in the existing disciplines in the Unified Process, for example, "Specify Security Needs" could be inserted to the Requirements discipline. Several disciplines would have their activity diagrams altered, which makes difficult for the understanding and implementation of the process. Another advantage of having a separate discipline for dealing with security issues is the facility that organizations will have if they wish to extend their process based on this work.

This discipline has been elaborated according to the orientations for tailoring of the RUP [5], and it must be executed in all phases but with more intensity in the inception and elaboration phase. Considering that, supporting disciplines are concerned with the overall management and structure of a RUP project, Software Security is this type because it concerns security management.

Each activity proposed for patterns implementation has been analyzed in order to define the activity execution sequence. This sequence has been defined from analyzing artifacts that are necessary to the activity execution and artifacts that were generated by activities. The sequenced activities have been organized in an activities diagram for the security discipline (Fig. 3).

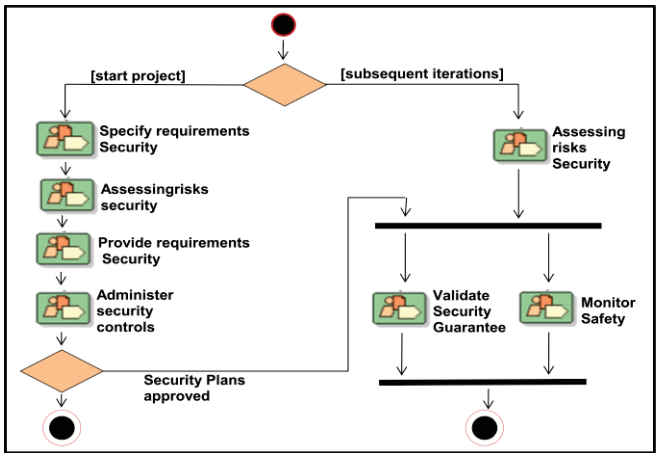


Figure 3. Activities Diagram for the Security Discipline

IV. METHODOLOGY PROPOSED FOR TAILORING

The methodology for software processes tailoring has the objective of allowing the elaboration of a software development process for a specific project, considering security practices that are described in the SSE-CMM and the SMT framework that is described in the section III.

In this work, the processes tailoring considers that there is an organization's standard software process (OSSP), which describes the process elements that must exist in all projects of the organization. The tailoring consists in altering OSSP in order to satisfy the project needs, especially in relation to security, and it results in the project-specific process (PSP). Fig. 4 shows the activities proposed methodology.

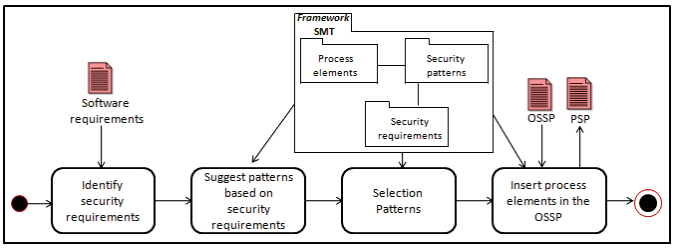


Figure 4. Security Methodology Tailoring (SMT)

Based on the security requirements identified for the project, the security engineer, who is assisted by the project manager, selects a set of process areas SSE-CMM that must be incorporated to the project (software requirements). In this selection, the opinion of specialists and stakeholders must be considered besides the security requirements that were defined for the project.

The patterns that seek to satisfy the PAs are selected based on the association rules defined in the SMT framework. The process engineer selects the patterns that they wish to incorporate to the OSSP from the suggested list. After selecting the security patterns, the process elements associated to their implementation are incorporated to the OSSP originating the PSP.

V. SMT – TOOL

An experimental environment to processes tailoring based on security requirements was developed and it is composed by the tools: *Security-Based Methodology Tailoring* (SMT-TOOL), *Project Risk Management Approach* (PRIMA-TOOL) [12] and *Pattern-Based Methodology Tailoring* (PMT-TOOL) [15].

PMT-Tool is responsible for cataloguing the security patterns. SMT-Tool is responsible for registering the processes areas and associates them to the security patterns. PRIMA-Tool module is responsible for the elaboration of the project software process, from organization's standard process tailoring, inserting in it the elements of the process associated to the selected patterns to the satisfaction of the processes areas proposed by SSE-CMM.

Having concluded the project process tailoring, PRIMA-Tool generates a website with the description of the project-specific software process to be consulted by developers, managers and process engineers. The website facilities easier the adoption of the software process by the developing team. These tools were used for the achievement of the case studies.

VI. CASE STUDIES

Two case studies were carried out to validate the proposed methodology. The first case study was made on XirooPACS system from Animati Computação Aplicada enterprise.

XirooPACS is a system about filing and communication of medical images. Examinations scheduling, patients' registers and the appraisal delivering of doctors and patients ordering by the internet are the main features of this system.

The system analysis was made considering mainly the matter of visualization of exams on web systems by patients and doctors, what makes this to need a highlevel of security.

The second case study was made on Plex system, from Elevata enterprise. PLEX is a credit card system which aims to enable the enterprises to have their own credit cards. This system joins services of cards administration including monitored selling services and evaluated risks.

The access to PLEX system is made on web. As the data bases of this system will involve data of many clients and several enterprises, and also flaws may cause a huge financial loss, this is a system which needs a high level of security involved in the process. The system analysis was made considering mainly the data bases and the integration on clients register and charge.

For the study case realization was used SMT framework proposed by this work. In the database of tools was inserted the elements of the process recommended by Rational Unified

Process, as well as the rules of association presented on Table I. Each pattern associated to a process area was described using the process elements proposed by the metamodel.

The first step in the realization of the case study was the filling of a questionnaire to the identification of the process area necessary for each project. The questionnaire was distributed to the project manager who with his team, answered the questionnaire for the prioritization of the security requirements (process areas) proposed by SSE-CMM model. The questionnaire contains the needs specifications to the understanding of the process areas. It was used a scale of relevance to the following levels: highest, high, medium, low or not relevant. The project manager attributed a level to each process area.

Based on the answers and on the framework configured on the tools the processes were adapted and are analyzed on next section.

A. *Case Studies Analysis*

The developed studies generated project-specific software process to each of the systems. These processes are used as a guide to the project development.

The case studies were made with projects which have very distinct features, although both of them need a high security level and therefore, generate specific-process similar. At the first case study, some process areas were not prioritized for the project security, yet on the second case study all the process areas were considered high or highest to the project, all of these prioritized (Table II).

The verification of whose activities are really high important to the project is not an easy task, once the consideration of adding requirements, which are not really necessary, it can be added extra and unnecessary costs to the project.

Through the two case studies was possible to verify the second project demands more security because the security engineer assigned to all process areas a high or highest importance. More preoccupation related to assess threads and vulnerabilities, build assurance argument, coordinate security and monitor security posture were presented; corresponding to process areas which were not implemented at the first study.

TABLE II. PRIORITIZATION OF THE PROCESS AREAS

PROCESS AREAS	XIROOPACS	PLEX
PA01 – Administer Security Controls	High	Highest
PA02 – Assess Impact	High	Highest
PA03 – Assess Security Risks	High	Highest
PA04 – Assess Threats	Medium	High
PA05 – Assess Vulnerabilities	Medium	Highest
PA06 – Build Assurance Argument	Low	Highest
PA07 – Coordinate Security	Medium	High
PA08 – Monitor Security Posture	Medium	Highest
PA09 – Provide Security Input	High	High
PA10 – Specify Security Needs	Highest	High
PA11 – Verify and Validate Security	High	Highest

The generated processes as the tailoring result were evaluated jointly to the project managers who on their analyses described that the process tailored to security presents clear activities to be implemented.

The rules of pattern association to the process areas are suggestions elaborated through the literature and may be improved by the organization by analyzing past projects, retrospective sessions, etc.

VII. RELATED WORKS

The development of reliable software has been discussed on many works which seek ways of enhancing the assurances of a project security or a software process. Some related works are presented next.

Mellado, Mediana and Piattini consider security requirements since the initial level of development of production lines, through an interactive and incremental process which where can be added additional tasks, according necessities. Through this tasks incorporation these authors search to make easier the conformity with the security requirements and manage the possible varieties which can happen on security requirements. The authors do not exemplify how the security requirements, out of security rules can be unfolded into task. In this sense, this article aims to define the tasks from the patterns association, which, usually, are widely explained or even suggest the tasks to be realized to fulfill the associated requirement.

Paes and Hirata (2007) propose an extension to the RUP with the inclusion of a discipline called "security". The discipline is based on good-practices and on experiences of the authors [5], but does not consider security standards or models. It is not described on this work how the defined process can be tailored to the projects needs.

Hafiz, Adamczyk and Johnson (2007) aim on using patterns to meet the criteria of security software. Although, the authors relate that there is a huge numbers of available security patterns and it is hard to choose which pattern is more recommended to each situation, as well as how to organize them to be used in a project.

This work is different from the others because it purposes a tailoring of software processes using security patterns and also because it elaborates a framework to make easier the elaboration of secures processes. The utilization of practices recommended by SSE-CMM model and of security patterns aim to use practices already consolidated for the development of reliable software.

VIII. CONCLUSIONS

This work proposes a methodology for processes tailoring that considers security requirements, which are proposed by a security model as criterion for tailoring, generating reliable software processes.

Information security has shown itself to be more and more important for the organizations, developers and users, and considering security from the beginning of the software development is desirable.

Elaborating a methodology for software processes tailoring is important to ease the tailoring task. The efficiency of the processes elaborated from the framework will depend of the rules of association pattern to process areas. An initial framework has been elaborated from the Rational Unified Process and activities proposed by the literature; from the security requirements proposed by the SSE-CMM e by the ISO/IEC 27001 Standard; and from security patterns described by Schumacher et al. [4], Rosado [5], Romanosky [7], Kienzle, among others.

The framework proposes a way to organize different elements used to elaborate new process. The organization can define patterns, processes elements and association rules that are adequate to their reality. The framework can be updated and must improve with time and as the team gets more experience. Results of post-mortem analysis of projects can help in this task.

Future work includes the definition of criterion associated to the security rules that seek to facilitate the prioritization of patterns to be applied in determined context and the experimentation of processes tailored by SMT in real projects.

ACKNOWLEDGMENT

We want to express our gratitude to CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior) for the financial support and a special gratitude to the organizations Animati and Elevata and their projects managers Jean Carlo Albiero Berni and Marcio Puntel for all their feedback and the very useful scientific discussions.

REFERENCES

[1] Kroll, J., Fontoura M. L., Wagner, R., (2010) "Usando Padrões para o Desenvolvimento da Gestão da Segurança de Sistemas de Informação baseado na Norma ISO/IEC 21827:2008". In: Simpósio Brasileiro de Sistemas de Informação, Marabá, Pará.

[2] SSE-CMM, 2003, Systems Security Engineering Capability Maturity Model SSE-CMM Model Description Document, Version 3.0, Carnegie Mellon University, Pennsylvania, USA.

[3] Schumacher, M., Fernandez, E. B., Hybertson, D., Buschmann, F. and Sommerlad, P.. 2006. *Security Patterns*, J.Wiley& Sons, England.

[4] Mellado D., Medina, F. E., Piattini M., 2008 *Security Requirements Variability for Software Product Lines* In: IEEE. University of Castilla La-Mancha – Spain.

[5] Rosado, David G. 2006A Study of Security Architectural Patterns. In Proceedings of the First International Conference on Availability, Reliability and Security (ARES'06).

[7] Romanosky, S. (2003) "Operational security patterns", In: EuroPLoP. Available on<http://hillside.net/europlop/europlop2003/papers/WritingGroup/WG4_RomanoskyS.doc> accessed in January 2010.

[8] Kienzle, D. M. and Elder, M. C. (2002) "Security Patterns for Web Application Development", Final Technical Report, Univ. of Virginia.

[9] Coplien, James. Sofware Patterns. Originally published by SIGS Books and Multimedia,1996.

[10] Ambler, S. W. (1998) "An introduction to process patterns", in SIGS Books/Cambridge University Press.

[11] Scarfone, K., Souppaya, M., Cody, A. and Orebaugh, A. (2008) "Technical Guide to Information Security Testing and Assessment: Recommendations of the National Institute of Standards and Technology", National Institute of Standards and Technology (NIST) Special Publication 800-115.

[12] Fontoura, Lisandra Manzoni; Price, Roberto Tom. Systematic Approach to Risk Management in Software Projects through Process Tailoring. In: International Conference on Software Engineering and Knowledge Engineering (SEKE'2008), 2008, Redwood City. Proceedings of the 20th International Conference on Software Engineering and Knowledge Engineering. Skokie: Knowledge Systems Institute Graduate School.

[13] Fontoura, M. L. "*PRiMA: Project Risk Management Approach*". Tese (Doutorado em Ciência da Computação). Universidade Federal do Rio Grande do Sul – UFRGS. Porto Alegre, Brazil. 2006.

[14] Shuya, A. "Welcome to the IBM Rational Unified Process and Certification". In: IBM Rational Software. Available at:<http://www.ibmpressbooks.com/bookstore/product.asp?isbn=0131562924 > Acessed jan. de 2010, 2008.

[15] Hartmann, J.; Fontoura, L. M.; Price, R. T. Tailoring Software Processes with Organizational Patterns Languages using Risk Analysis.In: Simpósio Brasileiro em Engenharia de Software, SBES, 19., 2005, Uberlândia.Rio de Janeiro: PUCRJ, 2005.

[16] Yoder, J. and Barcalow J. (1997) "Architectural Patterns for Enabling Application Security", In: 4th Conference on Pattern Languages of Programs. Edinburgh, United Kingdom.

[17] Paes, C. E. B. e Hirata, C. M. 2007. *RUP* extension for the development of secure systems. In: Portal ACM, Pontifícia Universidade Católica de São Paulo - Instituto Tecnológico de Aeronáutica – São Paulo, Brazil.

Security Analysis of FileZilla Server Using Threat Models

Michael Sanford, Daniel Woodraska, Dianxiang Xu
National Center for the Protection of the Financial Infrastructure
Dakota State University
Madison, SD 57042, USA
{michael.sanford, dcwoodraska, dianxiang.xu}@dsu.edu

***Abstract*— FTP is a widely used protocol for working with remote file systems. Various FTP implementations have had security problems reported as late as 2010. There lacks a systematic analysis of FTP security. In this paper, threat models are built to provide a systematic coverage of potential security attacks against an FTP server. Security tests are then generated from the threat models and applied to FileZilla Server, a popular FTP server implementation. When FileZilla Server is properly deployed, it holds fast against our security attacks. To further evaluate the effectiveness, the security tests are used to exercise a number of security mutants of FileZilla Server where various vulnerabilities are injected deliberately. The security tests have detected all but one of the injected vulnerabilities. This indicates that the threat model-based approach to security analysis of FileZilla Server is effective.**

Keywords- Security testing, FTP, threat modeling, threat tree, mutation testing

I. INTRODUCTION

FTP (File Transfer Protocol) is a widely used method for working with remote file systems. While FTP is an old protocol, various implementations show security problems that were reported as late as 2010 [1-3]. Using FTP across an untrustworthy network (e.g., Internet) can still expose an organization or business to security risks, yet there is no systematic analysis of FTP security to help understand the risks and mitigate them.

This paper presents an approach to FTP security analysis using threat models. We build threat models against most of the FTP services using the STRIDE threat classification system [4]. STRIDE stands for Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, and Elevation of privileges [4]. It provides a systematic coverage of the potential security attacks against an FTP server. We generate security tests from STRIDE threat models and apply them to FileZilla Server, a popular open source FTP implementation. When FileZilla Server is properly deployed, all of the attacks against it fail. When the deployment uses the default settings, however, one of the security tests leads to a successful denial of service attack.

To further evaluate the effectiveness of our approach, we created a number of security mutants of the FileZilla server program by injecting various vulnerabilities into the code. Then we ran the security tests against each of the mutants. A mutant is said to be killed if one of the security tests is a successful attack. The security tests have killed all but two of the mutants. This demonstrates that the threat model-based approach to security analysis of FileZilla Server is effective.

II. THREAT MODELING OF FTP SERVER

Security threats are potential security attacks. Threat modeling usually consists of four steps: modeling system functions, specifying security threats, ranking threats for risk analysis, and mitigating threats [4]. Functional modeling helps identify the system assets (e.g., data) that need to be protected and the entry points that an attacker might exploit. As this paper focuses on the security analysis of an existing system, we are not concerned with threat ranking or mitigation.

Table 1. Partial list of FTP Services vs. STRIDE Threats

Service	S	T	R	I	D	E
Login	4			1	3	2
Change directory	1	3	1	2	1	1
Get current directory						
Change to passive mode		2	2			
Put file to server		4	1		3	1
Get file from server		3	2	1	1	
Create directory		1			1	
Delete a directory		2	1		1	
Delete a file		3	2		2	
Execute commands on server		1		1	1	1
Rename a file		2	1		2	1
Append to a file					1	1
Non-service specific	1		1	1		

The FTP standard has evolved over many years via RFCs (959, 1123, 3659, 4217) published by the Internet Engineering Task Force (IETF). FTP operations include login, change directory, list directory, get current directory, change to passive mode, put file to server, get file from server, create directory, delete directory, delete a file, execute commands on server, rename a file, append a file, etc. These operations are listed in the first column of Table 1. To provide a systematic coverage of the potential attacks against the FTP services, we identify the STRIDE threats with respect to each service. We have also considered the vulnerabilities reported by various

sources, including but not limited to SANS [1], US-CERT [3], Security Focus [2], Vulnerability Scanning [5], Offensive Security's exploit database [6], the now extinct website milw0rm.com (went down during our research) [7], and FileZilla's own bug report system [8].

Attack (threat) trees [9][10] are used to visualize the threat models, An attack tree shows the steps an attacker would go through to compromise a system. The root of an attack tree specifies the ultimate goal of an attack; child nodes represent the sub-goals of their parent node (or the steps to achieve the parent's goal); and leaf nodes describes the primitive attack conditions or actions. An "AND" relation between two or more nodes means that all of the sub-goals or conditions in these nodes must be satisfied in order to achieve the parent's goal. An "OR" relation means that the parent's goal is reached when one of the sub-goals or conditions is satisfied [10].

Figure 1 shows denial of service attacks that prevent a legitimate user from logging in. The first attack attempts to fill up the login table that keeps track of the user connections. The second attack fills up the receive buffer of the server to the point to of crashing. The protocol requires a carriage return to be sent at the end of the command. Otherwise, it just keeps buffering the data coming in until it overflows.

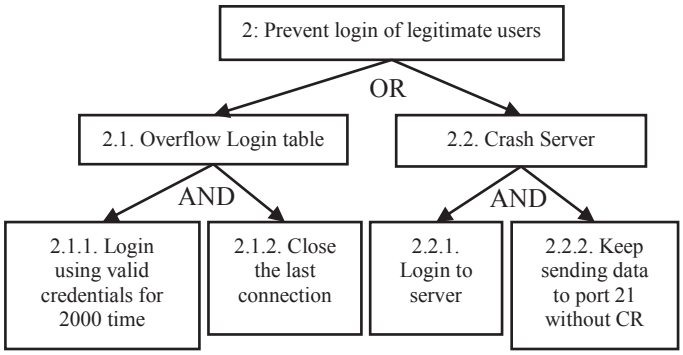


Figure 1. Prevent legitimate login

We have built 14 attack trees for the FTP services. They include 52 unique attack paths. An attack path is the minimalist set of leaf nodes taken to achieve some goal of the tree (root or sub-goal) [10]. Table 1 shows the number of threat trees that apply to each STRIDE category. Note that a threat tree can belong to multiple STRIDE categories. For example, an elevation of privilege attack often leads to information disclosure and tampering.

III. SECURITY TESTING OF FILEZILLA SERVER

After the attack trees have been constructed, we conduct security testing as follows. For each attack tree, we first generate all attack paths (also called test sequences). Then they are converted into security tests. Consider the attack tree in Figure 2. A is achieved if B or C are satisfied. B can be satisfied by 1 and 2 and node C by 3 and 4. Therefore, the attack paths are <1, 2> and <3, 4>. In our approach, we generate the test sequences from an attack tree automatically.

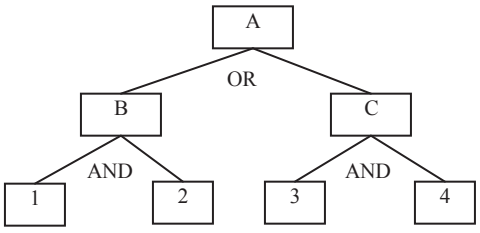


Figure 2. A sample attack tree

Definition 1. Suppose S[i] ($0 \le i \le n$) is a set of sequences. The combinational concatenation of all sequences in S[i], $S[0] \times S[1] \times \ldots \times S[n] = \{\langle s_0, s_1, \ldots, s_n\rangle$: for $\forall s_0 \in S[0], \forall s_1 \in S[1], \ldots \forall s_n \in S[n]\}$.

For example, let S[0] = {p1, p2, p3}, S[1] = {q1, q2}, S[2] = {r1, r2}, where $p_i (1 \le i \le 3)$, $q_j (1 \le j \le 2)$, r_k $(1 \le k \le 2)$ are sequences. Then S[0]×S[1]×S[2] yields the following 12 sequences:

{<p1, q1, r1>, <p1, q1, r2>, <p1, q2, r1>, <p1, q2, r2>
<p2, q1, r1>,<p2, q1, r2>,<p2, q2, r1>,<p2, q2, r2>
<p3, q1, r1>,<p3, q1, r2>,<p3, q2, r1>,<p3, q2, r2>}

Algorithm 1 below describes how to compute the combinational concatenation of sequences, where "∪" in line 7 refers to union of sets.

Algorithm 1: concatSequences
Function: combinational concatenation of sequences
Input: {S[0], S[1], …,S[n]}; Output: S[0] × S[1] × …× S[n]
Declare: S is a set of sequences
S[i][j] is the j-th sequence in *S[i]*

```
1.  begin
2.      if n=0
3.      then return S[0];
4.      S = ∅;
5.      for i=0 to S[0].size-1
6.        for j=0 to S[1].size-1
7.          S ← S ∪ {<S[0][i], S[1][j]>}
8.        endfor
9.      endfor
10.     if n=1
11.       return S;
12.     else //n>1
13.       return concatSequences(S, S[2],…, S[n]);
14. end
```

Algorithm 2 describes how to generate test sequences from (the root of) a threat tree, where "∪" in line 12 refers to union of sets and "×" refers to combinational concatenation of sequences in Definition 1 and Algorithm 1.

Algorithm2: genSequences
Function: Generate attack paths from a threat tree node.
Input: threat tree *node*; Output: attack paths
Declare: *childNodes* is the list of child nodes of a node;
childNodes[i] is the i-th node in *childNodes*
S[i] is a set of sequences

```
1.  begin
```

```
2.    if node is leaf
3.    then return {<node>};
4.    childNodes ← all child nodes of node
5.    if childNodes.size()=1
6.    then  return genSequences[childNodes[0];
7.    for i=0 to childNodes.size-1
8.      S[i]←genSequences(childNodes[i])
9.    endfor
10.   if the relationship between child nodes is "OR"
11.   then
12.     return S[0] ∪ S[1] ∪ …∪ S[childNodes.size-1]
13.   else  // the relationship is "AND"
14.     return S[0] × S[1] × …× S[childNodes.size-1]
15. end
```

Let us apply Algorithm 2 to the attack tree in Figure 2, genSequence(A) = genSequences(B) ∪ genSequences(C) because the relation between B and C is "OR". genSequences(B) = genSequences(1) × genSequences(2) = {<1>} ×{<2>} = {<1,2>} because the relation between 1 and 2 is "AND" , genSequences(1) = {<1>}, and genSequences(2) = {<2>}. Similarly, genSequences(C) = genSequences(3) × genSequences(4) = {<3>} ×{<4>} = {<3,4>}. Thus, genSequence(A) ={<1,2>}∪ {<3,4>} = {<1,2>, <3,4>}.

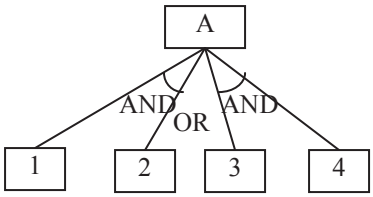


Figure 3. Informal attack tree with combined AND and OR

For simplicity, Algorithm 2 assumes that all the child nodes of a node, if any, have a single logical relation, i.e., either "AND" or "OR", but not a combination of them. This does not lose generality because a mixed relation of "AND" and "OR" can be easily converted into a tree without mixed relation. For example, the nodes in the attack tree in Figure 3 have a combination of "AND" and "OR". It can be converted to the attack tree in Figure 2. They have the same attack paths {<1,2>, <3,4>}.

Once the sequences were generated, they were converted into security tests by determining the system settings (i.e., user accounts and server setting), the actual parameters for the attack actions and conditions in the test sequence (e.g., user name and password for a login session), and oracle values (e.g., criteria about whether or not the attack is successful). If the primitive attack actions or conditions in the security test are programmable, we write code for the security test so that it can be executed automatically. It should be noted that security tests generated from the threat models are different from traditional functional tests in success and failure terminology. If a security test passes, the attack is successful (which would violate security requirements or policies). A failure can be due to a variety of issues, such as failure to connect to server, failure to get to a specific directory, unexpected response from the server, etc., or that the actual attack didn't work.

Table 2. Summary of security tests

Service	S	T	R	I	D	E
Number of threat trees	5	6	3	3	4	2
Number of test cases	8	33	7	24	10	15
Number of automated test cases	7	28	2	17	4	10

From the 14 threat trees we created, we generated 52 security tests. 39 of them can be executed automatically, another 8 can be executed automatically with some tester intervention and 5 are entirely manual tests. Table 2 shows how many tests are related to each classification of STRIDE. Again, one test can belong to multiple STRIDE categories.

IV. EXPERIMENTS

Our experiments consist of two phases. First, we run the security tests generated from the attack trees against FileZilla Server (version 0.9.34). Second, we create security mutants of FileZilla Server by vulnerability injection and run the same security tests against each of the mutants. FileZilla Server version 0.9.34 has 88,596 lines of code (71,940 actual C++ code lines), 107 classes, 1,716 methods, 193 functions, with an average method complexity of 4.74. Method complexity is defined as the number of unique execution paths that can be made through a function or method. In the following, we present the results of testing FileZilla Server and its mutants and discuss the lessons learned from this work.

A. Security Testing of FileZilla Server

When FileZilla Server is properly deployed (i.e., settings are configured properly), all generated security attacks failed to accomplish their goal. This indicates that FileZilla Server is secure against these attacks. When the security tests were first executed against FileZilla Server with the default settings, two of the attacks were successful. One achieved denial of service through creating thousands of login connections with the same id. The attack overflowed the login table of FileZilla Server, which keeps track of the active users and the current activity of that id. A proper setting should limit the number of concurrent connections any given id may have. The default is zero which means unlimited. This default setting doesn't give any hint to the administrator that there is an actual limit that might be exploited. The other successful test gained the version of the server that was running. While the version information is not always harmful, it can give the attacker clues as to what attacks might be possible by looking at the public bug tracker. In the attack trees, the disclosure of version information for production use (not development use) is considered as a security risk although the risk is not necessarily high.

B. Security Mutation Analysis of FileZilla Server

The fact that the properly deployed FileZilla Server has warded off our security tests indicates the good security profile of FileZilla Server. But it does not reflect how effective the

security tests can be. In this paper, we use security mutation testing to evaluate the vulnerability detection capability of the security tests and the thread modeling process.

Mutation testing has been used for evaluating software for the past several decades [11]. The fundamental idea of mutation testing is to run test cases against mutants which are obtained by injecting faults deliberately into the correct or baseline version of a program. These faults would represent the same programming errors a programmer might make. Usually, the percentage of mutants killed by the test cases is an indicator of how effective the tests are [12]. So far, creation of mutants in the existing mutation testing research has focused on syntactic changes, such as replacing && (and) with || (or). However, syntactic changes may not result in meaningful vulnerabilities in security-intensive software. In this paper, we create security mutants according to the causes and consequences of vulnerabilities. The causes of vulnerabilities include design-level defects (e.g., incorrect policy enforcement) and implementation-level programming errors (e.g., buffer overflow and unsafe function calls). Design-level vulnerabilities are a major source of security risks in software [13]. To create design-level mutants we took a high level look at the program by creating a software architecture document of the program. This document, along with a thorough code review gave us a broad view of how pieces fit together throughout the program and gave us the necessary insight to create some design level mutants. A mutant allowing a user to create directories without the necessary permissions would be classified as a design-level vulnerability in the FileZilla program. This is because no matter what the programmer does it is an unfixable problem using the current design. The consequences of vulnerabilities refer to various potential STRIDE attacks. We have created 30 mutants of FileZilla Server, 13 with design-level vulnerabilities and 17 with implementation vulnerabilities. The mutants have also included reported vulnerabilities of FileZilla Server.

Injection of security vulnerabilities into FileZilla Server is conducted independently of threat modeling and test generation. A different person from the team not involved with the threat modeling studied the FileZilla server and created the mutant injections. The security tests generated from our initial set of threat models killed 8 of the first 14 mutants (57.14%). The other mutants were not killed because the threat models did not cover these vulnerabilities. Then, we built more threat models and thus generated more tests to exercise the mutants. It is important to note that no implementation details were discussed between the person that implemented the mutations and the person that implemented the threat models. Only generic information such as "List command causes crash" is given. In the end, we were able to kill 28 of the 30 mutants for a 93.33% success rate. The two remaining mutants were a memory leak which is difficult to capture by threat models and an implementation dependent issue of an FTP command not covered by the threat model.

C. Discussion

Our experiments show that threat model-based testing is an effective approach to the security analysis of FileZilla Server. The security tests generated from the threat models have killed 93.33% of the deliberately-created mutants. One reason for this effectiveness is that the security tests generated from the threat models directly target the unintended behaviors or invalid inputs that are exploited by attacks. They meet the need of security testing to test the "presence of an intelligent adversary bent on breaking the system" [14].

From the experiments, we have learned several lessons. First, compared to traditional modeling that focuses on the intended behaviors of software, threat modeling requires a significantly different way of thinking. It would not be effective unless the threat models are built as if the builder were an intelligent adversary. Examining STRIDE threats against each system function can help build threat models in a systematic manner. This greatly reduces the chance of missing critical threats. Second, the vulnerability detection capability of the threat-based testing depends upon the threat models. A vulnerability may not be revealed if it is not captured by the threat models. Since threat models typically represent known or anticipated attacks, it can be hard to detect unknown vulnerabilities. Third, test automation can reduce the testing workload because the test execution can be easily repeated. For example, the security tests were executed against each of the 30 mutants. It would have been a daunting task if the security tests had not been automated. Although automation is desirable, not all tests can be automated. For example, user interactions may be needed to change server configuration (e.g., permissions), use administrative functions, and verify attack effects. Fourth, security mutation through injection of vulnerabilities is more difficult than traditional mutation. The former requires an in-depth understanding about the system functions and security requirements. The latter usually focuses on syntactic changes. Although the 30 distinct mutants have covered all FTP services and different vulnerabilities leading to various STRIDE attacks, they are far from complete.

V. Related Work

Threat modeling has been a viable practice for secure software development for some time [15]. Threat models can be used to generate security tests for exercising an implementation. In the past, we have proposed security testing approaches based on threat models represented by UML sequence diagrams [16] and threat trees [17]. Our previous work on testing with threat trees focused on web applications. This paper applies threat modeling and testing to FTP services, which has not been found in the literature. This paper has formally described the algorithms for generating test sequences from attack trees and used the security mutation to evaluate the vulnerability detection capability of the threat model-based testing approach.

Blackburn et al. [18] have developed an approach for generating test vectors (i.e., test cases with test input values, expected output values and traceability information) from security properties. Jürjens [19] has developed an approach for testing security-critical systems based on UMLsec models. Test sequences for security properties are generated from UMLsec models to test the implementation for vulnerabilities.

Julliand et al. proposed an approach to generating security tests in addition to functional tests [20]. The above work does not involve threat models in security testing.

Recently, testing of access control policies has gained increasing attention. Martin et al. [21] have been investigating techniques for test generation from access control policy specifications written in XACML (OASIS eXtensible Access Control Markup Language). Masood et al. [22] have investigated a state-based approach to test generation for role based access control (RBAC) policy. Their approach first constructs a finite state model of the RBAC policy and then drives tests from the state model. Le Traon et al. have investigated various issues of model-based testing of access control policies [23]. Mallouli [24] et al. have used formal access control models for generating tests. None of this work has used threat models for security testing.

VI. CONCLUSIONS

We have presented the modeling of various security threats against FTP services. These threat models can be used for security testing of different FTP server implementations. In this paper, the security tests generated from the threat models are executed against Filezilla Server, a popular FTP server implementation. Our experiments demonstrate that this is an effective approach to the security analysis of FileZilla Server. Although the security tests did not find risky vulnerabilities in the properly deployed FileZilla Server, they have detected the vast majority of the injected vulnerabilities in the security mutants. Our study indicates that modeling and testing with attack trees has some limitations. It is difficult to describe data flows among attack conditions and actions and represent repetitive actions. As a result, it requires some effort to transform the test sequences generated from attack trees to meaningful or executable security tests. This is a barrier to automated generation of executable code for those security tests where the attack actions are programmable. We are exploring how to generate executable security tests from formal threat models represented by Petri nets [13].

Currently, our study has created and used 30 security mutants of the FileZilla Server program. To better evaluate the vulnerability detection capability of the threat model-based testing approach, we plan to create mutants in a more systematic way. For example, we can try to inject vulnerabilities according to the various programming bugs that lead to security vulnerabilities.

ACKNOWLEDGMENT

This work was supported in part by NSF under grants CNS 0855106 and CNS 1004843.

REFERENCES

[1] SANS. (2010). *Computer Security Training, Network Research & Resources*. Available: http://www.sans.org/

[2] SecurityFocus. (2010). *Security Focus*. Available: http://www.securityfocus.com/

[3] US-CERT. (2010). *US-CERT: Other Resources*. Available: http://www.us-cert.gov/resources.html

[4] F. Swiderski and W. Snyder, *Threat Modeling*. Redmond, WA, USA: Microsoft Press, 2004.

[5] VulnerabilityScanning.com. (2010). *Vulnerability Assessment available for FTP*. Available: http://www.vulnerabilityscanning.com/FTP-Security.htm

[6] Offensive-Security. (2011). *Exploit Database search results*. Available: http://www.exploit-db.com/search/?action=search&filter_page=1&filter_port=21

[7] JF, Keystroke, ExtreemUK, savec0re, and VeNoMouS. (2010, June 17). *Milw0rm search: port 21*. Available: http://www.milw0rm.com

[8] FileZilla-Project. (2010, June 21). *Open Bug Reports*. Available: http://trac.filezilla-project.org/query?status=accepted&status=assigned&status=new&status=reopened&component=FileZilla+Client&order=time&desc=1&type=Bug+report

[9] B. Schneier, "Attack trees: modeling security threats," *Dr. Dobb's journal,* vol. 24, pp. 21-29, 1999.

[10] A. Jürgenson and J. Willemson, "Serial model for attack tree computations," *Information, Security and Cryptology–ICISC 2009,* pp. 118-128, 2010.

[11] Y. Jia and M. Harman, "An analysis and survey of the development of mutation testing," *Software Engineering, IEEE Transactions on,* p. 1.

[12] J. H. Andrews, L. C. Briand, and Y. Labiche, "Is mutation an appropriate tool for testing experiments?," Proc. of the 27th international conference on Software Engineering, St. Louis, MO, USA, 2005.

[13] D. Xu and K. E. Nygard, "Threat-driven modeling and verification of secure software using aspect-oriented Petri nets," *IEEE Transactions on Software Engineering,* vol. 32, pp. 265-278, 2006.

[14] B. Potter, B. Allen, and G. McGraw, "Software security testing," *Security & Privacy, IEEE,* pp. 32-36, 2004.

[15] D. Xu, "Software Security," in *Wiley Encyclopedia of Computer Science and Engineering*. vol. 5, B. W. Wah, Ed., ed Hoboken, NJ: John Wiley & Sons, Inc, 2009, pp. 2703-2716.

[16] L. Wang, E. Wong, and D. Xu, "A threat model driven approach for security testing," in *3rd International Workshop on Software Engineering for Secure Systems*, Minneapolis, MN, 2007, p. 10.

[17] A. Marback, H. Do, K. He, S. Kondamarri, and D. Xu, "Security test generation using threat trees," presented at the Fourth International Workshop on the Automation of Software Test (AST'09), in conjunction with ICSE'09, Vancouver, Canada, 2009.

[18] M. Blackburn, R. Busser, A. Nauman, and R. Chandramouli, "Model-based approach to security test automation," in *Quality Week 2001*.

[19] J. Jürjens, "Model-based security testing using UMLsec," *Electronic Notes in Theoretical Computer Science (ENTCS),* vol. 220, pp. 93-104, 2008.

[20] J. Julliand, P. Masson, and R. Tissot, "Generating security tests in addition to functional tests," in *Third International Workshop on Automation of Software Test (AST'08)*, 2008, pp. 41-44.

[21] E. Martin and T. Xie, "A fault model and mutation testing of access control policies," in *16th international Conference on World Wide Web (WWW'07)*, 2007, pp. 667-676.

[22] A. Masood, R. Bhatti, A. Ghafoor, and A. Mathur, "Scalable and effective test generation for role-based access control systems," *IEEE Transactions on Software Engineering,* pp. 654-668, 2009.

[23] T. Mouelhi, Y. Le Traon, and B. Baudry, "Transforming and selecting functional test cases for security policy testing," in *Second International Conf. on Software Testing Verification and Validation (ICST'09)*, Denver, CO, 2009, pp. 171-180.

[24] W. Mallouli, J. M. Orset, A. Cavalli, N. Cuppens, and F. Cuppens, "A formal approach for testing security rules," in *12th ACM symposium on Access Control Models and Technologies*, Nice-Sophia Antipolis, France, 2007, pp. 127-132.

Misuse Patterns for Cloud Computing

Keiko Hashizume, Eduardo B. Fernandez
Dept. of Computer and Electrical Engineering and Computer Science
Florida Atlantic University
Boca Raton, FL, USA
ahashizu@fau.edu, ed@cse.fau.edu

Nobukazu Yoshioka
GRACE Center
National Institute of Informatics
Tokyo, Japan
nobukazu@nii.ac.jp

***Abstract*—Cloud Computing is a new computing architecture that allows providers to deliver services on demand by means of virtualization. This work presents misuse patterns that describe how some typical attacks are performed and indicates appropriate countermeasures. These misuse patterns provide insight on how to build secure clouds and are also useful to evaluate the security of existing cloud systems. Specifically, we present a pattern for Malicious Virtual Machine Creation and describe two other patterns we have written.**

Keywords--; cloud computing, misuse patterns, security patterns, virtualization threats

I. INTRODUCTION

The Internet has been developing very rapidly during the last decade. The cost of storage is increasing as well as the cost of the power consumed by the hardware [9]. Thus, organizations need new solutions for these problems. Cloud computing is a new paradigm that improves the utilization of resources and decreases the power consumption of hardware. Cloud computing allows users to have access to resources, software, and information using any device that has access to the Internet.

Virtualization is a key feature for cloud computing [5], it allows many virtual machines to run on a single physical machine. Virtual machines are created and monitored by a Virtual Machine Monitor (Hypervisor), a software layer that mediates between the hardware and the operating systems running in each virtual machine. Virtualization allows users to create, copy, share, migrate, and roll back virtual machines, which provide significant benefits for the users [4]. However, it also brings new security problems.

To design a secure system, we first need to understand possible threats to the system. In [2], we proposed an approach to identify threats by considering the activities in each use case. That approach finds threats as goals of the attacker that must be realized by the lower levels of the system. We need to understand how the specific components of the architecture are compromised or used by an attacker in order to fulfill her objectives. To do this, we apply misuse patterns [3], to describe threats in cloud computing. A misuse pattern describes how a misuse is performed from the point of view of the attacker. It defines the environment where the attack is performed, countermeasures to stop it, and it provides forensic information in order to trace the attack once it happens. Misuse patterns are useful for developers because once they determine that a possible attack can happen in the environment, a corresponding misuse pattern will indicate what security mechanisms are needed as countermeasure. Also, misuse patterns can be very useful for forensic examiners to determine how a misuse is performed, and where they can find useful evidence information after the attack is done. Note that the misuse pattern describes a complete misuse, e.g. stealing information from a database, not just specific steps used to perform the misuse, such as SQL injection or buffer overflow. Misuses do not provide steps to be avoided when building the system, so they are not antipatterns, although they have common aspects.

We present here one example of a misuse pattern to illustrate our approach. While the specific attacks we describe are known, our contribution is describing the attacks in a systematic way, indicating how they can be neutralized. In particular, sharing virtual machine images brings new threats. Virtual machine images are prepackaged software templates that are used to instantiate virtual machines [8]. Cloud providers offer a repository service where providers and users can store their images. Users can either create their own image, or they can use any image stored in the repository. An attacker who creates a valid account can create an image containing malicious code such as a Trojan horse. If another customer uses this image, the virtual machine that he creates will be infected with the hidden malware. This malware can then perform a variety of misuses.

Section 2 presents a template used to describe misuse patterns. In Section 3, we present a misuse pattern for cloud computing, Malicious Virtual Machine Creation. In Section 4, we present some discussion on how misuse patterns can be used, and in Section 5 we offer some conclusions and possible future work.

II. TEMPLATE FOR MISUSE PATTERNS

A. *Name*

The name of the pattern should correspond to the generic name given to the specific type of misuse in standard attack repositories.

B. *Intent or thumbnail description*

A short description of the intended purpose of the pattern (what problem it solves for an attacker).

C. *Context*

It describes the generic environment including the conditions under which the attack may occur. This may include

minimal defenses present in the system as well as typical vulnerabilities of the system.

D. Problem

From an attacker's perspective, the problem is how to find a way to attack the system. The forces indicate what factors may be required in order to accomplish the attack and in what way; for example, which vulnerabilities can be exploited.

E. Solution

This section describes the solution of the attacker's problem, i.e., how the attack can reach its objectives and the expected results of the attack. UML class diagrams show the system under attack. Sequence or collaboration diagrams show the exchange of messages needed to accomplish the attack.

F. Affected system components (Where to look for evidence)

The pattern should represent all components that are important to prevent the attack and are essential to the forensic examination.

G. Known uses

Specific incidents where this attack occurred are preferred but for new vulnerabilities, where an attack has not yet occurred, specific contexts where the potential attack may occur are enough.

H. Consequences

Discusses the benefits and drawbacks of a misuse pattern from the attacker's viewpoint. The enumeration includes good and bad aspects and should match the forces.

I. Countermeasures and Forensics

It describes the security measures necessary in order to stop, mitigate, or trace this type of attack. This implies an enumeration of which security patterns are effective against this attack. From a forensic viewpoint, it describes what information can be obtained at each stage tracing back the attack and what can be deduced from this data in order to identify this specific attack.

J. Related Patterns

Discusses other misuse patterns with different objectives but performed in a similar way or with similar objectives but performed in a different way.

III. MISUSE PATTERNS FOR CLOUD COMPUTING.

We present one of our patterns, although not complete due to space restrictions.

Malicious Virtual Machine Creation

1) Intent

A Virtual Machine Image is a type of virtual appliance that is used to instantiate a Virtual Machine (VM). Virtual Machine Images contain initial file system state and software for the machine. An attacker may create a virtual machine image that contains malicious code so it can infect other users when they create their virtual machines. The attacker may read also confidential data from images that are publicly stored in the provider's repository.

2) Context

Some IaaS (Infrastructure as a Service) providers offer a VM image repository where users can retrieve images in order to initialize their VM. These VM Images can be created and published by the provider or by a client.

3) Problem

To perform some types of misuse it is necessary to be able to create and publish VM images.

The attack can be performed by taking advantage of the following vulnerabilities:

- Any person who has a valid account can create and register a VM image.
- There should be a common place where the users can share VM images.
- VM images contain prepackaged software components for an application. Thus, an attacker can create a VM image with malicious code.
- VM images contain installed and fully configured applications. The configuration may require sensitive operations such as creating username and password [8].

4) Solution

When a user publishes a VM image as public, any other user of the cloud is able to use it to instantiate his VM. This VM image can contain malicious code. The Virtual Machine Monitor (VMM) will run this image in order to instantiate the user's VM. Now, the attacker can have control of the virtual machine and perform malicious activities such as infect other computers. Infected virtual machines may appear briefly, infect other virtual machines, and disappear before they can be detected [4].

Structure

Figure 1 shows a class diagram for the repository for VM images in cloud computing. A Virtual Machine (VM) is an isolated software container that has a CPU, RAM, hard disk, and network controllers. It requires the installation of operating system and applications. A Virtual Machine Image (VMImage) is a pre-installed operating system and application stack. A User retrieves a VMImage from the Provider's Repository. These images can be created and published by other users or by the provider. The Provider is composed of a Hypervisor and Hardware (server, storage and network). The Virtual Machine Monitor (VMM) creates VMs by instantiating a VMImage, and it assigns their instances to the users who requested them. When the instance is launched, it is assigned to a physical server and given other hardware resources. The VM passes system calls to the VMM which executes those calls in the Hardware.

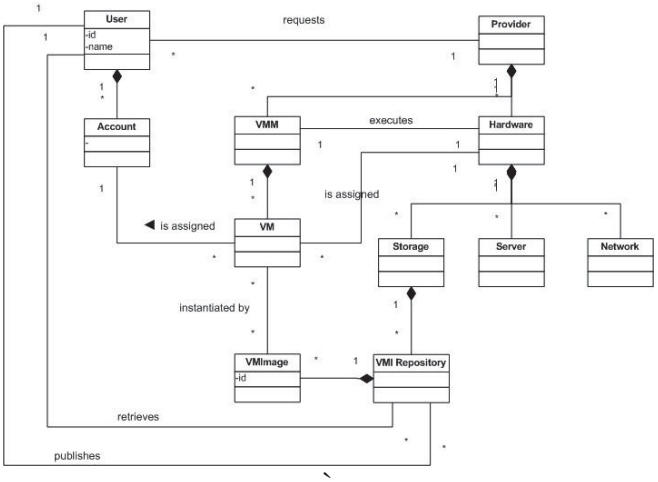


Figure 1. Class diagram for VM Image Misuse Pattern

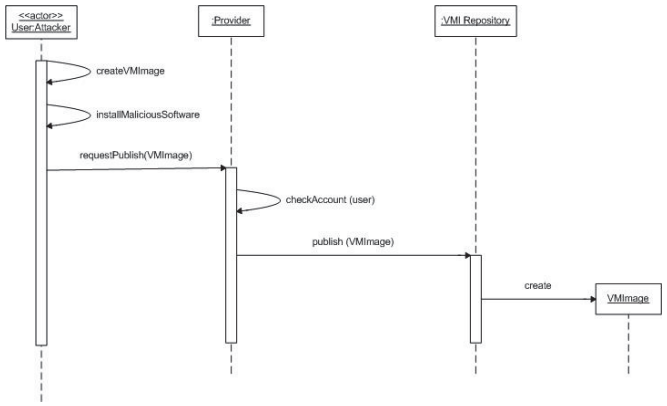


Figure 2. Sequence Diagram for the use case Publish a Malicious VM Image

Dynamics

UC1: Publish a Malicious Virtual Machine Image (Fig. 2)

Summary: The Attacker publishes a Virtual Machine Image that contains malicious code.

Actor: Attacker

Precondition: The Attacker must have an account with the Provider

Description:

a) The Attacker creates a VM Image.

b) The Attacker installs malicious software within the VM image

c) The Attacker requests to the Provider to upload the VM Image.

d) The Provider checks if the attacker (legal user) has an account.

e) The Provider uploads the VM Image into the repository.

f) The Provider sends an acknowledgement to the attacker.

Postcondition: A VM Image is created and placed into the Provider's repository, so any other user can use it and get infected.

UC2: Launch a VM using an infected VM Image (Figure 3)

Summary: A user launches a VM using an infected VM Image.

Actor: User

Precondition: The user must have an account, and the attacker has published her VM Image into the Provider's repository. Also, the user should choose the VM Image created by the attacker.

Description:

a) The User request to the Provider to retrieve a VM Image.

b) The Provider checks if the user has a valid account.

c) If the user is valid, the Provider sends the list of VM Images.

d) The User chooses the VM Image, and he requests to launch a VM to the Provider.

e) The Provider forwards the request to the VMM that will create the VM.

f) Once the VM is launched, it executes the malicious code.

Postcondition: The user's VM is infected and it may infect other VMs.

5) Consequences

Some of the benefits of the misuse pattern are the following:

- An attacker can open an account using a valid credit card and register malicious VM images into the provider's repository.
- VM Images contain all the software dependencies needed to run the Trojan horse, so the attacker does not need to worry whether the victims' software stacks satisfies the Trojan horse's dependencies [8].
- An attacker can get control of the infected virtual machines and obtain some confidential information.
- Repositories that contain malicious VM images can be a way to distribute malware.
- Infected virtual machines may infect other virtual machines. For example, an attacker may create a malicious VM image in order to infect other machines creating a collection of infected machines, Botnet [6].

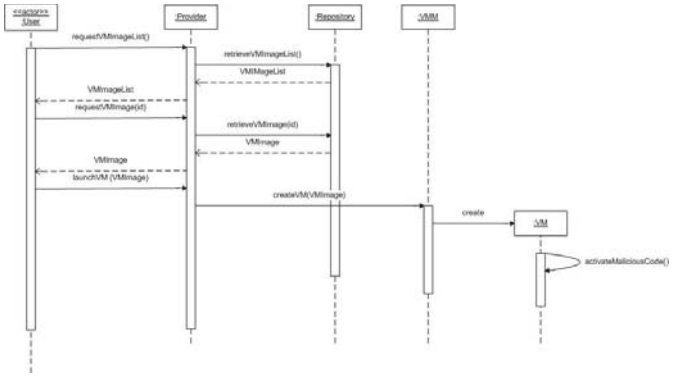


Figure 3. Sequence Diagram for the use case Launch a VM using a malicious VM Image

Possible sources of failure include:

- There is a possibility that the users choose to create their own VM images instead of using a public VM Image.
- Since VM images are dormant artifacts that reside in the repository, they are not harmful if they are not executed.
- Users can only retrieve images that are from certified owners.

6) Countermeasures

Malicious VM Images can be stopped by the following countermeasures:

- [8] proposed an image management system that control access to images, tracks the origin of images, and provides image filters and scanners that detect and repair security violations.
- Verify the background of the user when opening an account; however, this is very hard to do.
- Users can only retrieve images from certified owners.
- Use an Intrusion Detection System (IDS) to identify malicious activities such as in [7].

7) Forensics

Where can we find evidence of this attack?

- Providers can keep logs of the users that publish/retrieve VM Images.
- We can audit a suspicious VM Image.

IV. DISCUSSION

Designers need to understand first possible threats before designing secure systems. However, identifying threats is not enough; we need to understand how a whole misuse is performed. Misuse patterns appear to be a good tool to understand how misuses are performed. It is possible to build a relatively complete catalog of misuse patterns for cloud computing. Having such a catalog we can analyze a specific cloud architecture and evaluate its degree of resistance to these misuses. The architecture (existing or under construction) must have a way to prevent or at least mitigate all the misuses that apply to it. When potential cloud customers buy cloud services they negotiate a Service Level Agreement (SLA) with the provider. This SLA could indicate what misuses the provider is explicitly able to control. Many providers do not want to show their security architectures; showing their list of misuse patterns would give them a way to prove a degree of resistance to misuses without having to show their security details.

V. CONCLUSIONS AND FUTURE WORK

We can describe cloud computing threats as misuse patterns, which describe in a systematic way how cloud misuses are performed. We illustrated our ideas with a specific pattern. We are continuing developing misuse patterns for cloud environments in order to create a relatively complete catalog for it that can be used by designers of secure cloud environments. Finally, we intend to incorporate these patterns into a secure systems design methodology.

ACKNOWLEDGMENT

This work was supported in part by the NSF (grants OISE-0730065). This work was also supported by the National Institute of Informatics of Japan. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect those of the NSF or the NII.

REFERENCES

[1] Amazon, “Amazon Elastic Compute Cloud – Developer Guide”, http://docs.amazonwebservices.com/AWSEC2/latest/DeveloperGuide/

[2] F. Braz, E.B.Fernandez, and M. VanHilst, "Eliciting security requirements through misuse activities" Procs. of the 2nd Int. Workshop on Secure Systems Methodologies using Patterns (SPattern'07). 2008..

[3] E.B. Fernandez, N. Yoshioka, and H. Washizaki, "Modeling misuse patterns", 4th Int. Workshop on Dependability Aspects of Data Warehousing and Mining Applications (DAWAM 2009), in conjunction with the 4th Int.Conf. on Availability, Reliability, and Security (ARES 2009). March 16-19, 2009, Fukuoka, Japan

[4] T. Garfinkel and M. Rosenblum, “When Virtual is Harder than Real: Security Challenges in Virtual Machine Based Computing Environments”, Proceedings of the 10th conference on Hot Topics in Operating Systems. June 12-15, 2005. Santa Fe, NM.

[5] U. Gurav and R. Shaik, “Virtualization – A key Feature of Cloud Computing”, International Conference and Workshop on Emerging Trends in Technology (ICWET 2010)

[6] Intel Corporation, “The Dark Cloud: Understanding and Defending against Botnets and Stealthy Malware”, Intel Technology Journal, Volume 13, Issue 02, 2009

[7] K. Vieira, A. Schulter, C. Westphall, and C. Westphall, “Intrusion Detection Techniques in Grid and Cloud Computing Environment”, IT Professional, July-August 2010, 38-43

[8] J. Wei, X. Zhang, et al, “Managing Security of Virtual Machine Images in a Cloud Environment”, 2009 ACM Cloud Computing Security Workshop (CCSW) at CCS. November 13, 2009. Chicago, Illinois, USA.

[9] S. Zhang, X. Chen, et al, “Cloud Computing Research and Development Trend”, 2010 Second International Conference on Future Network. 2010

A Meta-Process to Support Trade-Off Analysis in Software Product Line Architecture

Edson A. Oliveira Junior and Itana M. S. Gimenes
Informatics Department - State University of Maringá
Maringá-PR, Brazil
Email: edson@edsonjr.pro.br, itana@din.uem.br

José C. Maldonado
Computing Systems Department - University of São Paulo
São Carlos-SP, Brazil
Email: jcmaldon@icmc.usp.br

***Abstract*—The software product line approach has been applied as a successful software reuse technique for specific domains. Such an approach takes advantage of domain and application engineering concepts. One of its most important artifacts is the product line architecture because it explicitly represents similarities and variabilities of a product line, as well as the products that can be generated. Product line architecture evaluation can serve as a basis for analyzing the managerial and economical values of a product line for software managers and architects. Such an evaluation might be carried out in terms of a trade-off analysis of the architecture quality attributes. Thus, this paper presents a Trade-off Analysis Meta-Process for Product Line Architecture, the TAMPro-PLA. TAMPro-PLA must be instantiated to define the essential artifacts for product line architecture trade-off analysis, such as business drivers, quality attributes, and respective scenarios. Therefore, TAMPro-PLA can be used to carry out product line architecture evaluations. It differs from current literature as it allows the definition and application of prodcut line architecture quality attribute metrics to provide support for performing quantitative and qualitative analysis. A proof of concept example based on the SEI's Arcade Game Maker (AGM) product line is presented.**

***Keywords*-meta-process, metrics, product line architecture, quality attributes, software product line, trade-off analysis.**

I. Introduction

A Software Product Line (PL) represents a set of systems sharing common features that satisfy the needs of a particular market or mission segment [8], [14]. This set of systems is also called a product family. The family's members are specific products developed in a systematic way from the PL core assets. The core asset has a set of common features as well as a set of variable parts, which represent later design decisions [14]. The composition and the configuration of such assets yield specific products.

The PL Architecture (PLA) is one of the most important asset of a PL because it abstractly represents the architecture of all potential PL products from a specific domain. The PLA addresses common design decisions, called similarities, as well as its postponed design decisions, called variabilities [16].

The results obtained from a trade-off analysis of PLA quality attributes are important from both academic and industrial view-points due to their potential to increase the productivity and the quality of products, decrease the time to market, improve the PL production capability [8], and to be used as a parameter for evaluating the PL in general [6]. PLA quality attributes take into account variabilities, which increase the effort to evaluate the quality of the overall PL [6]. Thus, we proposed a **T**rade-off **A**nalysis **M**eta-**Pro**cess for Software **P**roduct **L**ine **A**rchitectures, the TAMPro-PLA.

TAMPro-PLA aimed at analyzing PLA quality attributes in order to prioritize them based on quantitative and qualitative analyzes. Such analyzes are carried out by instantiating TAMPro-PLA, performing its activities and defining its elements for a given PLA, such as business drivers, scenarios, and metrics. The instantiation of TAMPro-PLA is illustrated, as a proof of concept, by performing its activities. Such activities generate the artifacts that allow PLA trade-off analysis taking into account the Arcade Game Maker (AGM) [15] PLA complexity and extensibility quality attributes.

This paper is organized as follows: Section II presents important concepts of quality attribute trade-off analysis for single product architectures as well as PLA principles; Section III describes the TAMPro-PLA meta-process, its activities, and essential artifacts for a PLA trade-off analysis; Section IV illustrates the application of TAMPro-PLA to the AGM PL; Section V discusses related work; and Section VI provides final remarks and directions for future work.

II. Quality Attribute Trade-Off Analysis and Product Line Architecture

Trade-off is the term used when one wishes to balance one situation or quality against another, in order to produce an acceptable result. With regard to software quality attributes, trade-off analysis means analyzing between multiple conflicting quality attributes to satisfy user requirements and come up with a better overall system [2].

One of the most consolidated method applied in industry to perform software architecture evaluation by means of trade-off analysis is ATAM (Architecture Tradeoff Analysis Method) [4]. ATAM's trade-off analysis allows discovering architectural problems during early software development phases. The cost to fix such problems in early stages is way less than in later stages.

Trade-off analysis usually takes into consideration the following artifacts of a software architecture [4]: (i) business drivers, which are statements about the architecture's overall goals; (ii) quality attributes defined for an architecture; and (iii) scenarios specified to exercise such quality attributes by taking into account the business drivers.

As the number of scenarios tends to be large in an architecture evaluation [4], stakeholders usually rank and select the most important scenarios. Thus, a widely known technique, applied in ATAM for instance, is the stakeholders voting. Each stakeholder involved in the evaluation process must assign a priority for each scenario based on predefined factors, such as the importance of a scenario. Then, scenarios are ranked and the most important are taken into consideration for performing quality attributes trade-off analysis.

Trade-off analysis can be performed to a PLA [7], [13] to prioritize its quality attributes for PLA evolution and deriving PL products. PLA differs from a single product architecture as it must represent the single architecture of all products that can be generated from a PL for a specific domain. The PLA gives stakeholders a means to develop multi-product architectures, while keeping their explicit models and visualizations in single product architectures. It results in product reduced cost and high quality. In addition, the PLA encompasses similarities, as well as postponed design decisions, the variabilities [16].

III. The TAMPro-PLA Meta-Process

TAMPro-PLA aims at defining artifacts for PLA quality attribute trade-off analysis. This involves the definition of business drivers and scenarios, selection of the PLA quality attributes to be analyzed; definition of managerial and technical questions to be answered with respect to the selected quality attributes; and application of quality attribute metrics to support data collection and analysis.

TAMPro-PLA takes as input the PL quality attributes, PL models, including the feature model. TAMPro-PLA does not depend on specific model notations or approaches. It can take into consideration, for instance, UML models. Regardless their granulatiry, PL models represent the main source of information to support the definition of the TAMPro-PLA's artifacts.

The artifacts that can be defined throughout TAMPro-PLA instantiation activities are:

- **Business Drivers (BD):** represent the main goals of PLA, based on the PLA quality attributes;
- **Defined Scenarios (DS):** a set of scenarios is defined for each PLA quality attribute;
- **Ranked Scenarios (RS):** scenarios are ranked based on PLA factors such as cost/risk of a scenario;
- **Managerial and Technical Questions (MTQ):** they should be answered in order to analyze a PLA and support the quality attribute metrics definition;
- **Quality Attribute Metrics (QAM):** they are defined to support the prioritization of PLA quality attributes in a trade-off analysis.

Figure 1 presents a UML activity diagram, which represents the TAMPro-PLA's activities (rounded rectangles) and their inputs and outputs (squared rectangles).

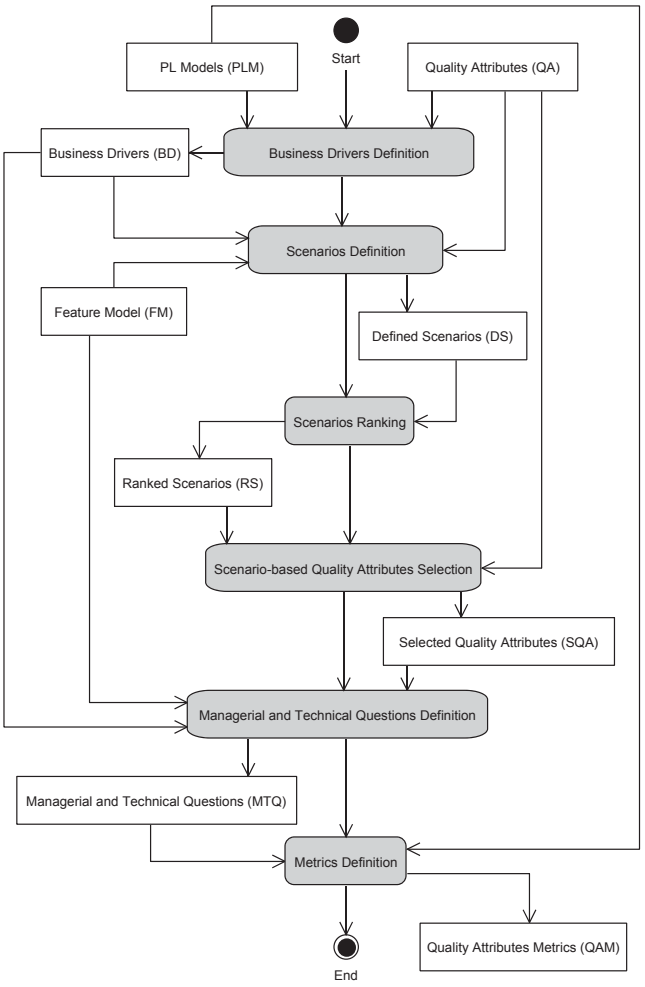


Figure 1. The TAMPro-PLA Meta-Process Activities and Artifacts.

The following items present a brief description of each TAMPro-PLA's activity:

The **Business Drivers Definition** takes as input the PL Models (PLM) and the PLA Quality Attributes (QA), and defines the Business Drivers (BD) that a PLA should reach to develop products. The defined business drivers support the definition of scenarios and managerial and technical questions, as well as they comprise the modeled PL variabilities.

The **Scenarios Definition** takes as input the Business Drivers, Feature Model (FM), and PLA Quality Attributes. It generates the Defined Scenarios (DS) for each PLA quality attribute to support its selection activity.

The **Scenarios Ranking** takes as input the Defined Sce-

narios (DS) in order to rank them based on the following PLA factors: (i) the overall importance of a scenario for the PLA, (ii) scenario generality (mandatory, alternative or optional), (iii) scenario cost/risk associated, and (iv) amount of variability associated to a scenario. It generates as output Ranked Scenarios (RS).

The **Scenario-based Quality Attributes Selection** takes as input the Ranked Scenarios (RS) and selects which quality attributes will be evaluated for a certain PLA as they tend to be in a large number. Its output is a set of Selected Quality Attributes (SQA), which is a subset of the Quality Attributes (QA) set.

The **Managerial and Technical Questions Definition** takes as input the Business Drivers (BD), Feature Model (FM), and Selected Quality Attributes (SQA). It defines the Managerial and Technical Questions (MTQ) that might be answered by defining metrics to support trade-off analysis. Such questions are defined with regard to the PLA business drivers, and they take into account the roles involved in the PL process as in [3].

The following items show some examples of roles and questions. More examples can be found in [16]:

– **PL Manager:** carries out activities such as planning, monitoring, and controlling the PL. Example of PL manager questions are: (i) `what is the effective investment for adopting the PL approach for a company?` and (ii) `which PL configuration is most feasible for a certain domain?`.

– **PL Architect:** in charge of the management of the PLA evolution, as well as for providing quantitative and qualitative design decision support. Examples of PL architect questions are: (i) `what is the required amount of effort to develop a product from a PL, based on the PL artifacts and their variabilities?` and (ii) `what is the impact of adding/modifying/removing the features of/from a PL on its PLA quality attributes?`.

The **Metrics Definition:** takes as input the PL Models (PLM), Selected Quality Attributes (SQA) and the Managerial and Technical Questions (MTQ). It defines Quality Attributes Metrics (QAM) to answer such questions and support data collection and quantitative analysis in PLA evaluations.

Although TAMPro-PLA contains the cited activities, performing of all of them are not mandatory. Stakeholders might already have defined, for instance, quality attribute scenarios and/or metrics. Therefore, TAMPro-PLA might be partially instantiated, as predefined artifacts might be incorporated.

IV. TAMPro-PLA Instantiation Example

As a proof of concept, this section illustrates the instantiation of TAMPro-PLA to perform a trade-off analysis to the Arcade Game Maker (AGM) [15] PLA quality attributes.

AGM is a pedagogical PL created by the Software Engineering Institute (SEI) to support learning and experimenting based on PL concepts. It has a complete set of documents, as well as a set of tested classes and source code for three different games: Pong, Bowling, and Brickles. Although AGM is not a commercial PL, it has been used to illustrate the concepts of several different PL approaches, as well as to support performing PL and architecture evaluation case studies [5], [9].

For this example, it is taking into account the AGM models, specially the feature model, as they are provided by the SEI [15]. The **feature model**, presented in Figure 2, is concerned with four top-level features for AGM products, which are: `services`, `rules`, `configuration`, and `action`.

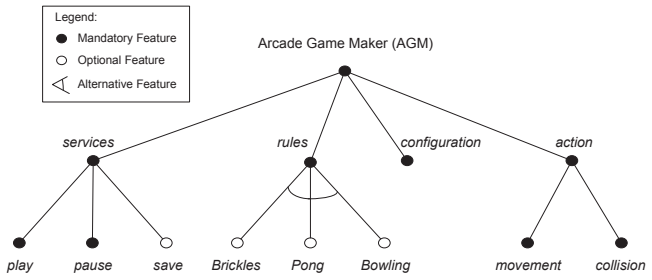


Figure 2. The Arcade Game Maker Feature Model.

In order to illustrate the TAMPro-PLA instantiation, complexity and extensibility quality attributes are considered. Thus, the activities of Figure 1 are realized. TAMPro-PLA takes as inputs the AGM PL models, its feature model and the PLA quality attributes. Therefore, next items present the TAMPro-PLA instantiation by realizing each of its activities.

Business Driver Definition: for the AGM example, we defined two business drivers based on [13] and [7] suggestions, as well as the analysis of the AGM PL models and the complexity and extensibility quality attributes:

- **BD.1 - keep game complexity degree lower than 0.7 (70%), compared to the overall PLA complexity, for at least 50% of produced products**: uphold low maintainability and low cost rates by focusing on complexity. Complexity degrees can provide an indicator of how dificult is to maintain the products derived from a PLA.
- BD.2 - **keep game extensibility degree higher than 0.75 (75%), compared to the overall PLA extensibility, for at least 50% of produced products**: maintain high reuse rate by focusing on extensibility. Extensibility factors can provide an indicator of how reusable is a product in terms of its components.

Scenarios Definition: taking into account the feature model, defined business drivers, and PLA quality attributes, Tables I and II present AGM defined scenarios. Features support the scenarios definition by linking a business driver to one or more scenarios and quality attributes.

Table I
AGM DEFINED SCENARIOS FOR COMPLEXITY.

AGM - Quality Attribute Utility Tree		
Quality Attribute	*Complexity*	
Related Feature(s)	*services, rules, actions*	
Related Business Driver(s)	***BD.1:*** *keep game complexity degree lower than 0.7 (70%), compared to the overall PLA complexity, for at least 50% of produced products*	
Scenario(s)	Sc.1	Variation points and/or variants are added, modified, or removed maintaining the BD.1 true.
	Sc.2	50% of variabilities are removed maintaining the BD.1 true.
	Sc.3	One-game environments have complexity values at most 0.65 (65%) compared to the overall AGM PLA complexity.

Table II
AGM DEFINED SCENARIOS FOR EXTENSIBILITY.

AGM - Quality Attribute Utility Tree		
Quality Attribute	*Extensibility*	
Related Feature(s)	*services, rules, actions*	
Related Business Driver(s)	***BD.2:*** *keep game extensibility degree higher than 0.75 (75%), compared to the overall PLA extensibility, for at least 50% of produced products*	
Scenario(s)	Sc.4	Variation points and/or variants are added, modified, or removed maintaining the BD.2 true.
	Sc.5	50% of variabilities are removed maintaining the BD.2 true.
	Sc.6	Two-game environments have extensibility values at least 0.8 (80%) compared to the overall AGM PLA extensibility.

Scenarios Ranking: stakeholders might take into consideration the following attribute concerns to support ranking each scenario as High (H), Medium (M), or Low (L):

- its **overall importance** for the PLA and its business drivers;
- the **generality** of the scenario with respect to the PLA. It is ranked as mandatory (High), alternative (Medium), and optional (Low) as in [13];
- its **cost/risk**, i.e., the effort involved in providing proper responses to the scenarios, as well as its perceived risk;
- the **number of variability**, encompassed by a scenario.

Table III presents the complexity and extensibility quality attribute scenarios ranking for our AGM example.

Scenario-based Quality Attribute Selection: in a trade-off analysis the number of PLA quality attributes can be large. Stakeholders might wish to select a subset of them to analyze. A widely known adopted strategy is the voting system as in the ATAM method [1], [4]. However, stakeholders can define their own strategy. For the AGM example both complexity and extensibility quality attributes are analyzed. However, as a matter of illustration, we present the rationale for it based on Table III: scenarios Sc.1, Sc.4, and Sc.5 are mandatory and have high number of variability and overall importance to the PLA with medium cost/risk; scenario Sc.6 is optional and it has the same ranking for amount of variability and overall importance as Sc.1, Sc.4, and Sc.5, as well as a low cost/risk; scenario Sc.2 is alternative and it has a high amount of variability and cost/risk, with medium importance to the overall PLA; and scenario Sc.3 is alternative and it has a low cost/risk, and high importance to the AGM PLA.

Table III
AGM COMPLEXITY AND EXTENSIBILITY SCENARIOS RANKING.

Business Drivers			***BD.1***			***BD.2***		
Quality Attributes			***Complexity***			***Extensibility***		
Scenarios			***Sc.1***	***Sc.2***	***Sc.3***	***Sc.4***	***Sc.5***	***Sc.6***
Scenarios Ranking	**Overall Importance**	**H**	X		X	X	X	X
		M		X				
		L						
	Generality	**H**	X			X		
		M		X			X	
		L			X			X
	Cost/Risk	**H**		X			X	
		M	X			X		
		L			X			X
	Number of Variability	**H**	X	X		X	X	X
		M			X			
		L						

Thus, scenarios Sc.1, Sc.4, and Sc.5 are the most important for the AGM example. Sc.1 is related to the business driver BD.1, whereas Sc.4 and Sc.5 are related to BD.2. Therefore, both complexity and extensibility quality attributes are related to important scenarios and are selected for the AGM example.

Managerial and Technical Questions Definition: Table IV presents questions defined by analyzing the AGM feature model, business drivers, selected quality attributes, and scenarios.

Table IV
AGM MANAGERIAL AND TECHNICAL QUESTIONS FOR THE PLA BUSINESS DRIVERS.

Business Driver / Feature / Selected QA (Goals)	**Questions**	
Low Complexity (business driver)	Q.01	What is the complexity of a class/interface in a class model?
	Q.02	What is the complexity of a variation point class/interface in a class model?
	Q.03	What is the complexity of a variability class/interface in a class model?
	Q.04	What is the complexity of a variable component in a model?
	Q.05	What is the complexity of a PLA based on its class model?
High Extensibility (business driver)	Q.06	What is the extensibility of a class/interface in a class model?
	Q.07	What is the extensibility of a variation point class/interface in a class model?
	Q.08	What is the extensibility of a variability class/interface in a class model?
	Q.09	What is the extensibility of a variable component in a component model?
	Q.10	What is the extensibility of a PLA based on its class model?

The **Metrics Definition** activity is not completely presented in this paper, as its realization involves theoretical and empirical validation of metrics. Thus, it was previously defined and validated [11] metrics for complexity and extensibility based on a basic metrics suite [10] for UML models [12]. Therefore, Figure 3 presents a GQM (Goal-Question-Metric) model relating the defined business

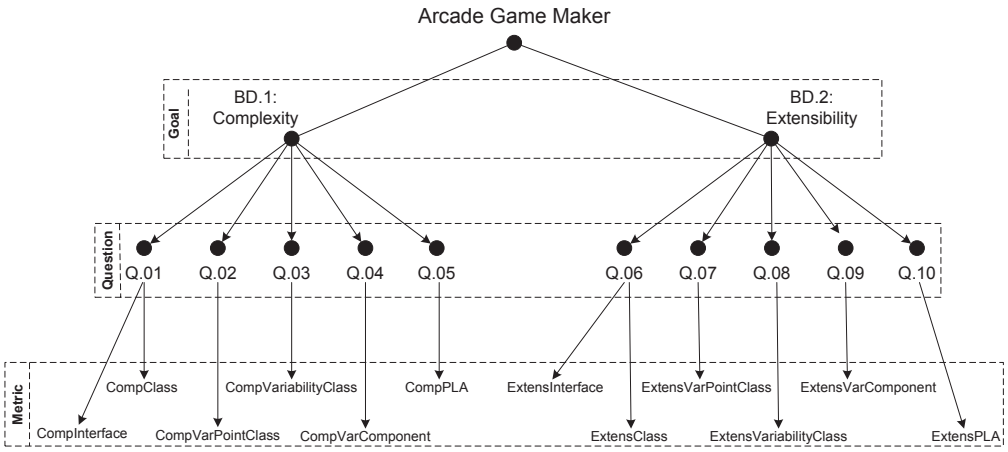


Figure 3. AGM Example Goal-Question-Metric Model.

drivers, questions and metrics.

Once TAMPro-PLA is instantiated and its artifacts are defined, stakeholders are able to perform a PLA trade-off analysis. As such analysis is usually based on the stakeholders analysis experience and knowledge of the domain, and they do not have a systematic realization of activities, some guidelines are suggested. They allow performing a PLA quality attributes analysis based on the instantiated TAMPro-PLA and its defined artifacts. Such guidelines are as follows:

1) generate a set of PLA configurations (PL products), sufficient to apply statistical normality tests and parametric or non-parametric methods for data interpretation;
2) collect quality attributes metrics from the generated configurations;
3) analyze the collected metrics descriptive statistics and present its important data, such as mean, median, and standard deviation;
4) identify how many scenarios satisfy the analyzed quality attributes and verify if either these scenarios are appropriated to the quality attributes or they must be re-stated; and
5) verify which quality attributes satisfy the PLA business drivers.

Therefore, based on such guidelines, stakeholders might decide which quality attribute(s) must be prioritized for PLA development and evolution. Stakeholders can also: provide potential PL products and PLA analyzes; share all data, keeping it for future analyzes; and write a final trade-off analysis report.

V. Related Work

Current literature presents two main works related to ours. Both are extensions of the ATAM method for PLA evaluation and emcompass quality attribute trade-off analysis.

The EATAM (Extended ATAM) method is proposed by Kim et al. [7]. Its main goal is PLA evaluation based on four methods: (i) identification of quality attributes, (ii) identification of architectural views, (iii) definition of PLUC (Product Line Use Case) tags for quality attribute variation points, and (iv) perform ATAM activities to validate the PL products single architecture separatelly. Basically, they use the first three methods to identify and represent variability and the fourth method to analyze the PL products architecture. The first method is strictly based on four main quality attributes: modifiability, portability, scalability, and extensibility. Thus, ATAM PLA trade-off analysis concerns general PLA quality attributes for qualitative analyzes. EATAM does not take into account nor provides metrics to support PLA quantitative analysis. EATAM provides, by means of the PLUC tag, directions of how to identify and relate scenarios to PLA variabilities which is used to perform trade-off analysis. TAMPro-PLA takes some of these directions to create scenarios. However, TAMPro-PLA does not use specific tags to represent variability as it does not depend on the type of the PL models.

The HoPLSAA (Holistic Product Line Software Architecture Assessment) approach, proposed by Olumofin [13], is aimed at evaluating PLA by means of qualitative analysis of variation point scenarios. It emcompasses two stages: (i) PLA analysis and (ii) single product architecture analysis. Its outputs are similar to the ATAM's outputs. Thus, it allows performing trade-off analysis based on PLA quality attributes during its first stage. HoPLSAA does not explicitly provide quantitative analysis due to its ATAM qualitative nature. In addition, HoPLSAA does not take into account metrics for performing trade-off analysis nor quality attribute metrics to support quantitative analysis. However, HoPLSAA allows the statement of scnearios based on PLA variabilities, providing TAMPro-PLA interesting directions fot the scenarios definition activity.

VI. Final Remarks and Future Work

This paper presented the TAMPro-PLA, a **T**rade-off **A**nalysis **M**eta-**Pro**cess for **PLA** to support performing PLA quality attribute trade-off analysis. TAMPro-PLA provides important artifacts for PLA trade-off analysis, such as: business drivers, scenarios, and metrics. Stakeholders must realize the TAMPro-PLA activities to instantiate TAMPro-PLA and produce its artifacts. As it does not depend on the type of PL models, it can be applied in general to perform PLA analyzes.

TAMPro-PLA differs from existing trade-off approaches as it both provides activities, that are realized systematically to produce essential trade-off analysis artifacts for a PLA, and allows the definition and application of quality attribute metrics, corroborating its quantitative and qualitative analysis results.

An example of how to instantiate TAMPro-PLA for quality attribute trade-off analysis of the Arcade Game Maker (AGM) PLA was demonstrated. In addition, guidelines to perform PLA trade-off analysis were suggested. At the end of the trade-off analysis process, stakeholders are able to: (i) decide which quality attribute(s) must be prioritized for PL development and evolution; analyze the potential PL products to predict their generation behavior; confirm the PLA accuracy of its modeled variabilities; share all data, keeping it for future analyzes and write a final trade-off analysis report; and use performed trade-off analyzes as parameter to evaluate overall PLs.

Based on current results, some directions for future work and contribuition might be: (i) the application of TAMPro-PLA to a commercial PL; (ii) the establishment of TAMPro-PLA's feasibility and effectiviness by carrying out experiments taking into account well-qualified industry professionals; (iii) the definition of relevant PL domains to apply TAMPro-PLA in several case studies; (iv) the development of an automated tool to support the TAMPro-PLA instantiation, as well as trade-off analysis; (v) the investigation of how TAMPro-PLA can provide Model-driven Architecture (MDA) support for definition and transformation of PL models; and (vi) how to incorporate TAMPro-PLA into PL evaluation approaches.

References

[1] M. R. Barbacci, "SEI Architecture Analysis Techniques and When to Use Them," Software Engineering Institute (SEI), Pittsburgh, USA, Tech. Rep. CMU/SEI-2002-TN-005, 2002. [Online]. Available: http://www.sei.cmu.edu/pub/documents/02.reports/pdf/02tn005.pdf

[2] M. Barbacci, M. H. Klein, T. A. Longstaff, and C. B. Weinstock, "Quality Attributes," Carnegie Mellon University (CMU) - Software Engineering Institute (SEI), USA, Tech. Rep. CMU/SEI-95-TR-021, 1995.

[3] G. Chastek and R. Ferguson, "Toward Measures for Software Architectures," Software Engineering Institute (SEI), Pittsburgh, USA, SEI Technical Note CMU/SEI-2006-TN-013, 2006. [Online]. Available: http://www.sei.cmu.edu/pub/documents/06.reports/pdf/06tn013.pdf

[4] P. Clements, R. Kazman, and M. Klein, *Evaluating Software Architectures: Methods and Case Studies*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.

[5] P. C. Clements, L. G. Jones, L. M. Northrop, and J. D. McGregor, "Project Management in a Software Product Line Organization," *IEEE Software*, vol. 22, no. 5, pp. 54–62, 2005.

[6] L. Etxeberria and G. Sagardui, "Variability Driven Quality Evaluation in Software Product Lines," in *Proceedings of the Software Product Line Conference*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 243–252.

[7] T. Kim, I. Y. Ko, S. W. Kang, and D. H. Lee, "Extending ATAM to Assess Product Line Architecture," in *Proceedings of the IEEE International Conference on Computer and Information Technology*. USA: ACM Press, 2008, pp. 790–797.

[8] F. J. v. d. Linden, K. Schmid, and E. Rommes, *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2007.

[9] B. Neto, L. Fernandes, C. Werner, and J. M. de Souza, "Developing Digital Games through Software Reuse," *Journal of Information Processing Systems*, vol. 6, no. 2, pp. 219–234, 2010.

[10] E. A. Oliveira Junior, I. M. S. Gimenes, and J. C. Maldonado, "A Metric Suite to Support Software Product Line Architecture Evaluation," in *Proceedings of the Conferencia Latinoamericana de Informática*, Santa Fé, Argentina, 2008, pp. 489–498.

[11] ——, "Empirical Validation of Complexity and Extensibility Metrics for Software Product Line Architectures," in *2010 Fourth Brazilian Symposium on Software Components, Architectures, and Reuse*. Salvador-BA, Brasil: IEEE Computer Society, 2010, pp. 31–40.

[12] ——, "Systematic Management of Variability in UML-based Software Poduct Lines," *Journal of Universal Computer Science (J.UCS)*, vol. 16, no. 17, pp. 2374–2393, 2010.

[13] F. Olumofin, *A Holistic Method for Assessing Software Product Line Architectures*. Saarbrücken, Germany, Germany: VDM Verlag, 2007.

[14] K. Pohl, G. Böckle, and F. J. v. d. Linden, *Software Product Line Engineering: Foundations, Principles, and Techniques*. Secaucus, NJ, USA: Springer-Verlag, 2005.

[15] SEI, "Arcade Game Maker Pedagogical Product Line," 2010. [Online]. Available: http://www.sei.cmu.edu/productlines/ppl

[16] R. N. Taylor, N. Medvidovic, and E. M. Dashofy, *Software Architecture: Foundations, Theory, and Practice*. USA: John Wiley & Sons, 2009.

Design of a UML profile for feature diagrams and its tooling implementation

Thibaut Possompès*‡, Christophe Dony‡, Marianne Huchard‡, Chouki Tibermacine‡
*IBM France – PSSC Montpellier
Montpellier, France
thibaut.possompes@fr.ibm.com
‡LIRMM, CNRS and Montpellier 2 University
Montpellier, France
{possompes, dony, huchard, tibermacin}@lirmm.fr

***Abstract*—This paper proposes an instrumented solution to integrate feature diagrams with UML models to be used as part of a general approach for designing software product lines and for product generation. The contribution is implemented in IBM®Rational Software Architect (RSA). It is intended to be used in the context of large, complex and multi-domain projects, and at allowing model transformations to derive products. Our RSA implementation makes it possible to link feature diagrams with UML model artifacts. It allows traceability between feature models and other different kinds of models (requirements, class diagrams, sequence or activity diagrams, etc.). It is used in a project dedicated to create smart building optimization systems.**

I. Introduction

IT projects are closely related to business domains. However, there exist few techniques to gather, link and manage the knowledge related to each domain. Nowadays projects are particularly confronted to problems where instrumentation of specific business domains is required to enhance their efficiency. Examples of these projects are smart buildings, grids, water management, health care or food systems. The research presented in this paper is done in the context of the RIDER project[1], which brings together several actors who works on improving energy efficiency of buildings. This context requires to be able to link several domains, such as building system components, IT infrastructure, air flow modeling, building thermal management, database models, *etc.* Each domain can be modeled by a different stakeholder, or sometimes being based upon a standard. Software product line approach is perfectly appropriate to manage the variations that can be found in building instrumentation systems. The solutions presented in this paper are applied to the RIDER project and benefit of its feedback. Our approach is used as a part of a general approach for software product lines and for product generation.

There exist several commercial tools such as [3], [9], [21] that allow linking features to model artifacts extracted from various modeling tools; eclipse plug-ins [20], [1], [13], [10], [27], [15], [12]; and standalone feature modeling tools [22], [24], [26]. These tools do not implement the latest feature modeling concepts and are not well integrated in UML modeling tools. There also exist several UML profiles [5], [30] and meta-models [29] implementations but they address neither the latest UML specification nor all feature model concepts that are useful in our context.

This paper is organized as follows. Section II presents a new synthetic and expressive feature diagram model based on the state of the art. Section III presents our feature meta-model which synthesizes this state of the art. Section IV explains how the corresponding UML profile has been created and implemented using RSA, and shows an excerpt of the project feature model. Section V will sum up what has been presented and suggests some perspectives for further study.

II. Synthesis of Existing Feature Diagram Models

Various feature diagram semantics and implementations have been proposed since the initial one given in *FODA* [16]. We use the following criteria to analyze them and to define our model:

1 – Feature definition.

2 – Feature relationships. We identify four criteria to classify existing proposals regarding relations among features.

- *2.1 – Hierarchical relationships.*
- *2.2 – Feature choice constraint relationships.* necessary to guide the user through feature selection (feature dependency, or mutual exclusion).
- *2.3 – Mandatory and optional feature identification.*
- *2.4 – Sub-feature selection semantics.*

3 – Feature logical groups. Allowing to group arbitrary features by business domains, or abstraction layers.

4 – Product and implementation information. To determine what kind of information can help implementing a product from a set of selected features.

[1]The RIDER project ("Research for IT as a Driver of EneRgy efficiency") is led by a consortium of several companies and research laboratories, including IBM and the LIRMM laboratory, interested in improving building energy efficiency by instrumenting it.

Table I
HIERARCHY RELATIONSHIP

	Decomposition	Specialization	
		Enrichment	Realization
FODA [16]	Consists of		
FORM [18]	Composed of	Generalization Specialization	Implemented by
Fey et al. [11]	Refine		Provided by
Zhang et al. [29]	Decompose Detail	Specialize	
Czarnecki et al. [8]	Relation		

A. Feature Definitions

The initial definition of features was introduced by Kang et al. in *FODA* [16] which is extended by the *FORM* [17] method. Features are defined as being essential characteristics of applications, described with domain vocabulary.

FORM has introduced four different perspectives to enrich the semantics of feature diagrams: capability, operating environment, domain technology and implementation technique. Indeed, features are also able to model knowledge of the various domain experts involved in the project. Fey et al. [11] add the *pseudo feature* concept in order to allow for specialization with non-exclusive alternatives and to avoid feature redundancy thanks to implicit inheritance. Zhang et al. define features [28], [29] as being essentially a cohesive set of individual requirements representing the user-visible capability of a software system. Czarnecki et al. [8], [6] consider that features are system properties relevant for some stakeholder, and that any kind of functional or non-functional characteristic of a described system can be represented by a feature. This extends the *intention* definition presented by Zhang et al. [28].

The definition proposed by Czarnecki et al. allows more freedom of expressiveness in feature diagrams. Its combination with *FODA* perspectives allows us to apply feature diagrams to the numerous domains involved in a project. The *property* concept introduced by Fey et al. and the *attribute* concept presented by Czarnecki et al. are semantically very close. It could be assessed that a property expresses the same information that an attribute, *i.e.*, a measurable characteristic. We will use the *property* concept to describe specific feature typed values (*e.g.*, a room volume, *etc.*) and how one property can influence another one.

We extend this concept to allow the customer to choose the property value. Therefore, we will add the necessary semantics to restrict the possible choices to consistent ones.

B. Feature Relationships

1) Hierarchical Relationships: Features and sub-features can be bound by either decomposition or specialization. Table I illustrates the different variations on these concepts. The FODA *consists-of* relationship [4], [16], [25] is designed to represent decomposition. One parent feature can have several sub-features grouped either by an *AND* or *XOR* decomposition semantics. *FORM* [17], [18] introduces the relations *Composed-of*, to describe the constituents of a feature; *Generalization / Specialization*, to specialize or generalize a feature; and *Implemented-by*, to define how a high-level feature can be implemented by a lower-level one. Fey et al. [11] uses two kinds of hierarchical links between features: *refine*, to detail a given feature at a lower abstraction level; and *provided-by*, to link a pseudo-feature with the features which it realizes. *Pseudo-features* express an abstract functionality, quality or characteristic. Fey et al. make it possible to build directed acyclic graphs, which is impossible with *FODA*, by allowing one feature to refine several ones. Zhang et al. [29] identify three different kinds of hierarchy relationships: *decomposition*, to refine a feature into its constituents; *detailization*, to identify feature attributes; and *specialization*, to add further details into a feature. Czarnecki et al. [8], [6], [7] decided not to consider relationships between features in favor of entity-relationship or class diagrams.

We have chosen to keep the hierarchy relationships categorized in Table I. with the following concepts: *decomposition*, which consists in *detailing* the sub-features that compose the parent feature; *specialization*, which encompasses the concepts of *enrichment* for sub-features that add functions to the parent feature, and *Realization* (or *implementation*) that describes how a feature can be implemented.

2) Feature Choice Constraint Relations: FODA [16] uses the concept of *composition rules* to describe how features relate to one another: one feature can *require* another one, or two features can be *mutually exclusive*. Riebisch et al. [23] introduce the *hint* relationship which recommend features to the user.

We have kept the constraints *require* and *conflict* to ensure a coherent product generation. Furthermore the *hint* relation is also convenient to make recommendation to the user during the feature selection process.

3) Mandatory and Optional Features Identification: In the *FODA* [16] and *FORM* [18] specifications and tools, all features are mandatory by default; optionality is represented as a feature property as in [23], [14], [29]. Czarnecki et al. [6] use cardinalities to express optionality. For instance, a *(1,1)* cardinality describes a mandatory feature and *(0,1)* describes an optional one. Setting a cardinality greater than 1 specify how often the sub-features of a parent can be duplicated.

We keep for our model the cardinality based relationship introduced by Czarnecki et al. [6] that brings an interesting enhancement to the initial definition, useful to describe more complex products from a product line.

4) Sub-feature Selection Semantics: FODA [16] and *FORM* [18] methods propose two ways to manage sub-feature selection. A simple link between the parent and its sub-features means that there is no choice constraint; equivalent to an *or* binary choice. A semi circle drawn

across the links means that there is an alternative choice; equivalent to the *xor* binary choice. Likewise, Fey et al. [11] express with a simple link that the choice is an *or*, but the alternative choice is expressed with a complete graph of mutually exclusive constraints among all sub-features. Riebisch et al. [23] and Czarnecki et al. [8] use cardinalities to define how many sub-features can be chosen.

The most convenient semantics relies on cardinality based selection semantics. It allows a maximum flexibility to describe how many sub-features can be selected. Furthermore, we chose to keep the *"or"*, *"and"*, and *"xor"* groups to ease the feature models semantics understanding for non-IT specialists end-users.

C. Feature Logical Groups

The *FORM* [18] method classifies features into four categories called layers, from a functional point of view. Riebisch et al. [23] use logical groups to represents aspects valuable to the customer and explain that *abstract features* could be used to encapsulate features related to a given concept. Fey et al. [11] present *feature-sets* to group features from an arbitrary point-of-view. Zhang et al. [29] present the *binding-time* concept to represent the phases of the software life-cycle in which each feature must be chosen. Czarnecki et al. [6], [8] use abstract features to reference other feature diagrams to reuse a set of features. They also introduce different types of features that can be considered as feature groups.

The *binding-time* concept can be extended according to the software development process chosen by the final user to allow him to assign features or groups of features to a development phase. It can be modeled by the *feature set* concept presented by Fey et al. The different kinds of features presented by Riebisch et al. [23] could be easily modeled by sub-layers. Hence, we choose to keep the layer concept to organize features in logical groups and sub-groups accordingly to the type of information they represent. We propose to enhance Fey et al. *feature-set* concept by adapting Zhang et al. [29] constraints meta-model to describe constraints inside a group of features, and constraints between two groups of features. The feature-sets could also be used along with the Kano method [19] to help the user choosing a set of product features that yield high customer satisfaction. The customer preference categories can be modeled as feature-sets encompassing the corresponding features. We also keep the feature-sets idea to reference sub-parts of a feature diagram. Hence, a feature-set could be a leaf of the diagram that encompasses another feature hierarchy. Staged configuration can be modelled by creating one feature-set for each configuration stage, and associating it with a group of stakeholders that have the same business concern. Czarnecki et al. [6], [8] have presented four types of features that have been integrated in our meta-model proposal: *concrete features* can be stored in the implementation layer; *aspectual features* can be stored in a sub layer of implementation layer; *abstract features*, *e.g.* performance requirements, can be represented by feature properties; and *grouping features* can be modeled as a feature set.

D. Product and Implementation Information

Riebisch et al. [23] argue that the feature hierarchy must be organized to make easier the choice of features by using composition relations and require associations. Zhang et al. [29] present a feature attribute for representing the feature *binding-state*. It must be used in a *binding-time* context, *i.e.* when features are implemented in the software product at a given software life-cycle phase. Mathematical relationships have been presented to describe the relative impact of one feature to another.

III. A Synthesis Meta Model for Feature Diagrams

This section describes a meta model synthesizing the choices we presented and motivated in the previous section. This work extends the work of Asikainen et al. [2] in order to apply our work in the context of a rich industrial project.

As depicted in Figure 1, a product line contains features, and a feature belongs to one product line. A product belongs to one product line and can be composed of as many features as needed. Features associated to a product must be analyzed in order to check whether all constraints, like mutual exclusion between features or require relations, are satisfied.

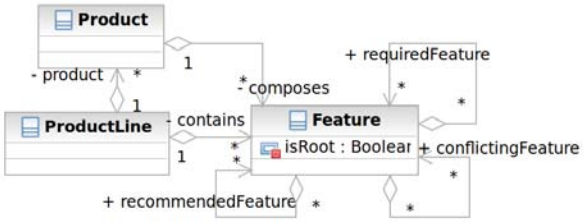


Figure 1. Product lines relation to features and products

Mutual exclusion and require relations link features regardless of their position in the hierarchy, they are modeled by the *conflictingFeature* and *requiredFeature* relationships. Furthermore, we add the *recommendedFeature* role used to advise a user to choose another feature pertinent in the product. These roles are depicted in Figure 1 in reflective association on the *Feature* class.

Feature properties (Figure 2) are used to describe either a feature parameter related to its inner requirements (*e.g.* the bandwidth capacity of a network) or a characteristic chosen by the user during the product definition (*e.g.* the frequency of automatic backups of a word processing software).

Features can have a variability type (*VariabilityKind*) which is further described below:

- *fixed*, A property value is fixed throughout all products of the product line.
- *variable*, A property value can change, within a product, depending on other features properties, *e.g.* the text buffer size of a text field in the user interface can vary accordingly to the type of information we want to store (*i.e.*, name or address).
- *family-variable*, A property can vary from product to product accordingly to the selected features, *e.g.* the phone book capacity depends on the presence of internal memory property and the sim card capacity.
- *user-defined*, A property value can be freely chosen in a given product, *e.g.*, the frequency of automated backups in a word processing software.

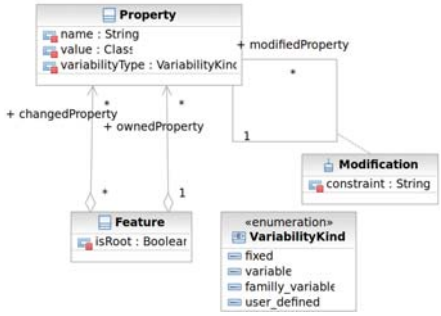


Figure 2. Feature properties

The hierarchy relationships and sub-feature groups are depicted in Figure 3. The relations between a feature and its sub-features are grouped by the *RelationshipGroup* class that contains the cardinalities necessary to restrict the number of sub-features to choose. Cardinalities can be chosen freely, but some groups have fixed cardinalities: *OrGroup*, *(0,*)*; *AndGroup*, *(*,*)*; and *XorGroup*, *(0,1)*. The *DirectedBinaryRelationship* class represents the kind of association that links a parent feature and a sub-feature. It is specialized either by *Enrich*, *Implement*, or *Detail* classes.

Figure 4 shows how layers and feature sets can be associated with the project stakeholders. A stakeholder represents any kind of people (*e.g.* domain experts, IT architects, *etc.*) allowed to choose features. Feature sets and layers can be attached to a concern specific to the project, *e.g.* network architecture, or business requirements. A *Layer* represents a specific view onto the software application, a feature can be in only one layer at a time. A *FeatureSet* inherits from the *Feature* class, and allows grouping features from an arbitrary point of view, *e.g.*, a business domain, or representing the features that must be implemented to fulfill a norm.

Figure 5 depicts how constraints can be applied to feature sets: *mutex*, when only one feature can be selected in the feature set; *None*, when there is no constraint among features;

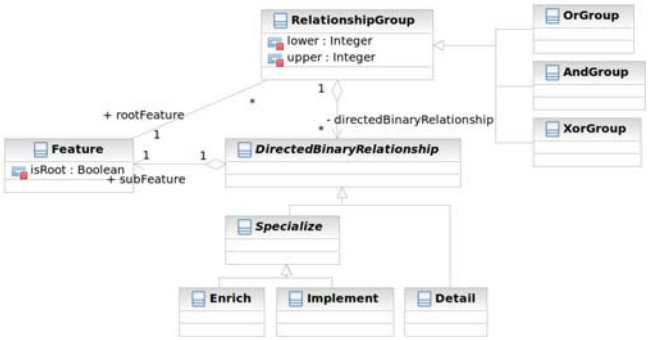


Figure 3. Groups and hierarchy relationships

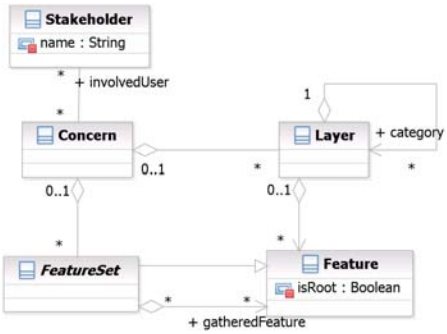


Figure 4. Stakeholders concerns

All, when all features or none of them can be selected. The *ConstraintRelation* class describes the relation between two feature sets: one feature set can require another one, two feature sets can mutually require each other, or be mutually exclusive. A *BindingPredicate* is used to represent how a constraint must be applied on each constrained feature-set. The choice of the specialized class must be made according to the kind of feature-set.

IV. UML Profile Implementation

A. Description

Our profile integrates the previously described feature meta-model into the UML meta-model hierarchy with an appropriate semantics. We describe here which UML meta-classes we have chosen to extend, which information has been added with the stereotypes and how we have restricted the semantics of extended meta-classes thanks to OCL constraints. However, the profile could be implemented in different ways by choosing to extend different meta-classes. We chose the meta-classes that had the closest semantics to our concepts, added the required information with the stereotypes and restricted the initial semantics of meta-classes to what is necessary for our profile with the Object Constraint Language (OCL) constraints.

Contrary to [5] we have based our feature diagram profile on the *Components* UML meta-class. A component, as a

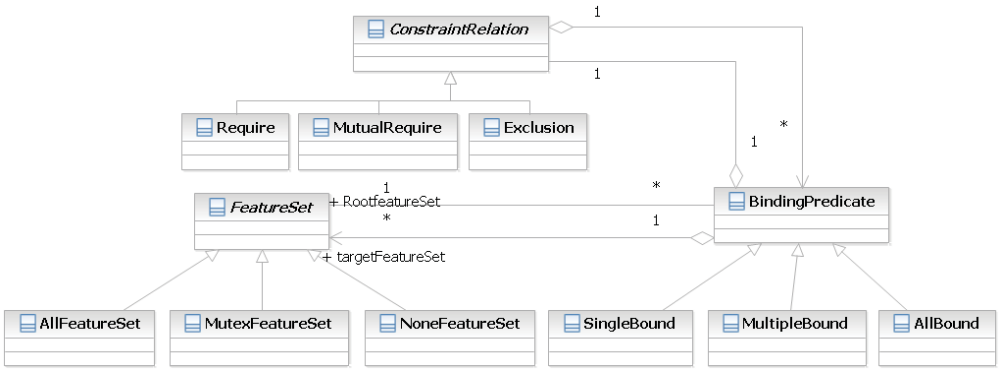


Figure 5. Constraints on feature sets

feature, can be seen as a high level view of a software element and, as such, is the concept the closest to what we want to express. Components have ports, which can be linked together. Ports are thus reasonably well suited to support the relationships between features and other elements defined in section III. It is the UML 2 concept the closest to what we want to express because it is a high level view of the structure of a software element. This first choice influences the other meta-classes selection.

Table II
STEREOTYPES EXTENSIONS

Stereotype	Extended Meta-class
Feature	Component
Stakeholder	Actor
Concern	Class
ModelRelationship	Dependency
Layer	Package
ProductLine	Component
Product	Component
Property	Port
RelationshipGroup	Port
Modification	Usage
DirectedBinaryRelationship	Association
BindingPredicate	Port
ConstraintRelation	Association

The *port* meta-class allows us to represent the interaction of a feature with other elements. It can be linked to other ports or components. The modification of a property value can be modeled by textual or OCL constraints placed upon the relationship between two properties.

We choose to create a Rational Software Architect plug-in to leverage its UML modeler, modeling editors, views and tools. All managed models are instances of EMF models. Hence using Rational Software Architect allows simplifying tasks like creating a specific plug-in for integrating feature modeling capabilities into standard EMF-based UML models and diagrams. The plug-in is currently used in the RIDER project.

Table II gives an abstract of our specialization choices, but due to space constraints we did not explicit all our design choices.

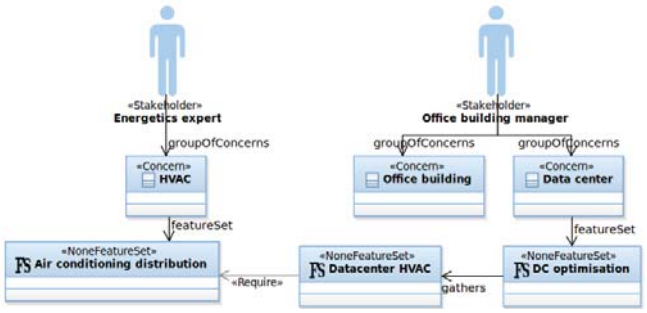


Figure 6. Stakeholders and feature sets

B. Example

Figure 6 shows an excerpt of a RIDER feature diagram expressed, using our profile, by two stakeholders having different concerns on a RIDER smart building project. One focusing on HVAC (Heating Ventilation Air Conditioning) domain and the second on data centers and office building optimization problems. Each feature set encompasses several features that are not shown in this example because of space limitation. The example shows in particular *require* high level constraints applying on groups of features (feature sets). The example also shows how *properties* can be used to add further information on the product context (such as the building volume).

The feature selection process is achieved by showing feature diagrams to the user accordingly to his concerns thanks to model transformations.

V. CONCLUSION AND PERSPECTIVES

In this paper we have presented a new feature diagram model and its profile implementation in UML2. This feature diagram model is first based on a synthesis of existing ones, and it has then been improved to fit the requirements of the

RIDER smart buidings project in which it is applied. Thanks to Rational Software Architect, we have generated a tool making it possible to produce feature diagrams conforms to our model. This synthesis is achieved by classifying the existing concepts into categories. This approach allows a full integration of feature diagrams into UML models and facilitates model transformations. In comparison with [2] we add several concepts such as *layers*, *stakeholder concerns*, *feature-sets*, and *group constraints*.

For the time being, we still need to guide users to organize features into layers and sub-layers in order to best integrate them into the software development life-cycle. Hence, the next steps will be to create a framework able to use the full potential of our feature meta-model, and to develop automated model transformation functionalities to automatically generate UML models.

REFERENCES

[1] M. Antkiewicz and K. Czarnecki. FeaturePlugin: feature modeling plug-in for eclipse. In *eclipse '04: Proceedings of the 2004 OOPSLA workshop on eclipse technology eXchange*, pages 67–72, New York, NY, USA, 2004. ACM.

[2] T. Asikainen, T. Mannisto, and T. Soininen. A unified conceptual foundation for feature modelling. In *Proceedings of SPLC '06*, pages 31–40. IEEE Computer Society, 2006.

[3] BigLever Gears. http://www.biglever.com/overview/software_product_lines.html.

[4] Y. Bontemps, P. Heymans, P. Y. Schobbens, and J. C. Trigaux. Semantics of FODA feature diagrams. In *Proceedings SPLC 2004 Workshop on Software Variability Management for Product Derivation*, pages 48–58, 2004.

[5] M. Clauss. *Untersuchung der Modellierung von Variabilität in UML*. Technische Universität Dresden, Diplomarbeit, 2001.

[6] K. Czarnecki, S. Helsen, and U. Eisenecker. Formalizing cardinality-based feature models and their staged configuration. *University of Waterloo*, 2004.

[7] K. Czarnecki, S. Helsen, and U. Eisenecker. Staged configuration using feature models. *Lecture notes in computer science*, 3154:266–283, 2004.

[8] K. Czarnecki, S. Helsen, and U. Eisenecker. Formalizing cardinality-based feature models and their specialization. *Software Process Improvement and Practice*, 10(1):7–29, 2005.

[9] M. Eriksson, H. Morast, J. Börstler, and K. Borg. *The PLUSS toolkit: extending telelogic DOORS and IBM-rational rose to support product line use case modeling*. ACM, 2005.

[10] Feature-Oriented Software Development Research. http://fosd.de/fide/.

[11] D. Fey, R. Fajta, and A. Boros. Feature modeling: A Meta-Model to enhance usability and usefulness. In *Software Product Lines*, pages 198–216. Springer, 2002.

[12] fmp2rsm Plug-in . http://gp.uwaterloo.ca/fmp2rsm/index.html.

[13] Generative Software Development Lab. http://gsd.uwaterloo.ca.

[14] M. L. Griss, J. Favaro, and M. d'Alessandro. Integrating feature modeling with the RSEB. In *In Proceedings of the Fifth International Conference on Software Reuse*, pages 76–85, 1998.

[15] Hydra. http://caosd.lcc.uma.es/spl/hydra/index.htm.

[16] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-Oriented domain analysis (FODA) feasibility study. Technical report, Carnegie-Mellon University Software Engineering Institute, Nov. 1990.

[17] K. C. Kang, S. Kim, J. Lee, K. Kim, E. Shin, and M. Huh. FORM: a feature-oriented reuse method with domain-specific reference architectures. *Annals of Software Engineering*, 5(1):143–168, 1998.

[18] K. C. Kang, J. Lee, and P. Donohoe. Feature-Oriented product line engineering. *IEEE Software*, 2002.

[19] N. Kano, N. Seraku, F. Takahashi, and S. Tsuji. Attractive quality and must-be quality. *Journal of the Japanese Society for Quality Control*, 1984.

[20] ProS Labs - Moskitt Feature Modeler. http://oomethod.dsic.upv.es/labs/index.php.

[21] Pure-Systems GmbH. http://www.pure-systems.com/.

[22] RequiLine. http://www-lufgi3.informatik.rwth-aachen.de/TOOLS/requiline.

[23] M. Riebisch. Towards a more precise definition of feature models. *Modelling Variability for Object-Oriented Product Lines*, pages 64–76, 2003.

[24] S2T2 - An SPL of SPL Techniques and Tools. http://download.lero.ie/spl/s2t2/.

[25] P. Schobbens, P. Heymans, J. Trigaux, and Y. Bontemps. Generic semantics of feature diagrams. *Computer Networks*, 51(2):456–479, Feb. 2007.

[26] SPLOT - Software Product Line Online Tools. http://www.splot-research.org/.

[27] XFeature. http://www.pnp-software.com/XFeature/.

[28] W. Zhang, H. Mei, and H. Zhao. Feature-driven requirement dependency analysis and high-level software design. *Requirements Engineering*, 11(3):205–220, 2006.

[29] W. Zhang, H. Zhao, and H. Mei. A propositional logic-based method for verification of feature models. *Lecture Notes in Computer Science*, 3308:115–130, 2004.

[30] T. Ziadi, L. Hélouët, and J. M. Jézéquel. Towards a UML profile for software product lines. *Lecture Notes in Computer Science*, pages 129–139, 2004.

Software Product Lines System Test Case Tool: A Proposal

Crescencio Rodrigues Lima Neto[1,3], Ivan do Carmo Machado[2,3],
Paulo Anselmo Mota Silveira Neto[1,3], Eduardo Santana de Almeida[2,3],
Silvio Romero de Lemos Meira[1,3]

[1]Center for Informatics - Federal University of Pernambuco (CIn/UFPE)
[2]Computer Science Department - Federal University of Bahia (DCC/UFBA)
[3]Reuse in Software Engineering Labs (RiSE)
{crln, pamsn, srlm}@cin.ufpe.br, {icm, esa}@dcc.ufba.br

***Abstract*—Nowadays the decision whether to work with Software Product Lines *(SPL)* or not becomes a binding requirement for the strategic planning of companies. Learning how to choose the ideal tools to test a SPL is beneficial for companies in this planning process. Even though the ascending number of available software engineering testing tools, SPL testing lacks specific tools capable of supporting the SPL Testing Process and managing the variability of test assets. This paper evaluates how to reduce the effort during the SPL Testing Process and consequently, how to make the variability of test assets manageable. We propose a software product line testing tool to build system tests from use cases that addresses challenges for SPL Testing we identified in a literature review.**

***Keywords*-Software Testing; Software Product Lines; Software Reuse; Testing Tools;**

I. Introduction

Software testing tools are available for testing in every stage of the software development life cycle, although it is not commercially viable for the tools vendors to produce a tool that suits every organization. It is inevitable, then customizations of tools is desirable [1].

Software product lines testing tools are not an exception, choose testing tools suitable for test applications and support process could be one of the most critical tasks of a project. In the SPL context, the amount of available tools decrease drastically, and the need of tools to reduce the effort during the SPL testing process is an obvious gap that need to be filled.

Software product lines is a planned, systematic and proactive reuse strategy, through exploiting the similarities within a set of products. SPL can enable organizations to achieve significant reductions in terms of development and maintenance cost and time-to-market as well [2], and remarkable quantitative improvements in productivity, quality and customer satisfaction [3], thus addressing the problems aforementioned.

In order to manage the variability, avoid the explosion of test cases due to the great number of variation points combinations and reduce the effort to test a software product line we need to implement testing tools that would allow for improvements in costs, time-to-market and quality [4].

The availability of tools makes testing a more systematic activity and can minimize the cost and time consumed, as well as the errors caused by human intervention [5]. A wide range of tools, with both commercial and academic purposes can be found. However, these tools have almost always been implemented for specific purpose and they are isolated from others, presenting its own architecture and internal structures [5]. As a consequence, difficulties in integration, evolution, maintenance, and reuse of these tools are very common. These tools often focus on automating specific testing techniques and criteria, without considering the whole testing process [5].

In this paper, we propose a SPL Test Tool for create system test cases based on use cases that supports the RiPLE-TE [6] test discipline of the RiPLE project. The RiPLE project is an effort of developing a framework to support a set of disciplines that compose the life-cycle of a SPL, namely Scoping [7], Requirements [8], Design [9], Implementation [10], Testing [11], Evolution Management [12].

We also investigate tools that can help during all tests levels of the SPL phases. Furthermore, in order to gure out the needs of the research field, we split our investigation into two steps. Firstly we analyzed test tools that have been developed for testing single systems; and secondly, we focused on tools that have been developed specifically to product lines. The question that motivates our work is: *How to handle variation points and their combination within a test case?*

The remainder of this paper is organized as follows. Section II describes the related work. Section III is an introduction to software product lines testing tools. Section IV analyzes the challenges of SPL Testing. Section V discusses the proposal. Finally, Section VI summarizes and concludes this paper.

II. Related Work

There are few studies describing and detailing tools for testing SPL projects. If we decide to narrow the scope, encompassing the search for only system test related tools, within the SPL context, the findings are worse.

According to [13] the ScenTED-DTCD (Domain Test Case Scenario Derivation) is a prototype tool focused on generating test cases scenarios that describe the test engineer's activities and the systems response if modeled within the activity diagram.

Another prototype tool can be found [14]. It generates system test scenarios from use cases. The approach is based on the automation of the generation of application system tests, for any chosen product, from the system requirements of a product line.

These studies are deemed to be good sources of information regarding tools for testing SPL. In order to develop our work, we considered every mentioned study, since they bring relevant information that comprises the knowledge to develop SPL system testing tools.

III. SPL Testing Tools

Testing is an expensive and laborious phase of the software process [15]. In SPL, this assumption continues to hold true. As a result, testing tools were among the rst software tools to be developed. These tools now offer a bunch of facilities and their use can signicantly reduce the costs of testing [15].

Finding an effective and efficient software testing tool could be a life-saver for a project or a company. There is no single test tool suitable for all possible systems and industry sectors. Deciding what criteria to apply when selecting a specific tool for a project is quite tricky [16].

A software product line is a large system, with large volume of source code, and such automated tools can aid developers and testers to scan through the large volume of product lines source code and reveal unreliable program features to the human program. Testing tools can support testing large size of software product lines to achieve its goals [17].

An important factor in SPL Testing is automated testing and the use of testing tools. This decreases the effort when reusing test assets and makes the complex testing process more manageable [18]. There are numerous mature testing tools available. Despite the seeming abundance of available testing tools SPL testing lacks efficient tool support. The problem with testing tools is that few of them directly support SPL [18].

A. Literature Review

This section presents how we identied papers reporting single systems testing tools and product lines testing tools, how we extracted data and information from the research papers, and how we analyzed them with respect to identify the requirements to compose a new testing tool focused on SPL Testing.

In order to extend the research and better understand the needs of tools for SPL Testing, we performed a research that firstly identified 26 relevant studies using dened search items related with the subject SPL testing tools. Then, we rened the search by applying some exclusion criteria to the study title. Next, we excluded studies on the basis of exclusion criteria applied to abstract and conclusion reducing the number to 15 papers. Finally, we obtained the primary studies set to be critically appraised. This set comprised 8 papers, that accordingly describe the implementation of a SPL testing tool.

In accordance with our findings, it is possible to check a huge amount of tools that support testing, but very few support SPL testing.

IV. Software Product Lines Testing Challenges

SPL Testing may be represented as two instances of the V-model, according to [6]. First, the domain assets are tested. Then, the products are tested according to the model. This raises two problems. First, complete integration and system testing in domain engineering is usually not yet specified [18]. Try to test the different combinations of components leads to exponential growth of tested configurations [19]. Second, it is hard to decide on how much we can depend on the domain testing. According to [18] it does not seem to be clear where testing belongs in the overall SPL process.

Another challenger is the organization of test assets. Management of testing becomes very difcult if testers cannot tell the difference between assets that belong to the domain and assets that belong to a product. Commonality and variability in product lines require rich traceability, from requirements to implementation and test assets, encompassing the whole development life cycle [19].

Integrating test environments and the test asset repository is a cumbersome work, that is even worse due to the lack of automated support tool, which leads to be a manually performed task. There is a need for specic testing tools that should help to manage the reusable testing assets and automate the execution of tests and the analysis of their results as well [18].

It is also evident the need of tools that integrate the whole process of software product lines testing including test environments and a test asset repository. A feasible approach for this would be to use a product lines specific integrated testing environment [20].

The product line approach requires a carefully planned testing process that can be easily adapted and used in different domains. Currently, there is a gap between the overall product line engineering processes and the assumed testing processes using the practical testing methods, because it is not clear how the processes and test assets are aligned to test

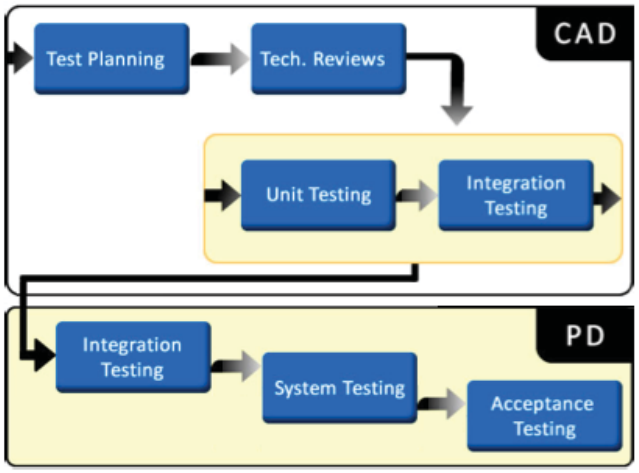


Figure 1. RiPLE-TE [6]

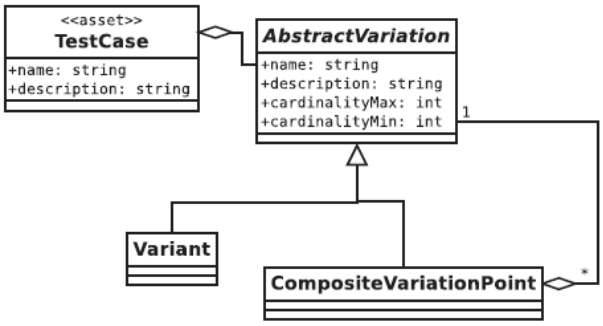


Figure 2. The tests model excerpt - from [21]

a product lines in cost-effective manner, while preserving quality [18]

In order to lessen some of these problems, this work is focused on the building of a tool to support testing activities, at the system test level, aiming at reducing the required effort. It builds on the definition of a structured process for testing SPL, the RiPLE-TE [6], as next described.

V. Towards a Tool to Support SPL Testing

In RiPLE-TE, test assets are built in parallel to the development assets. Thus it is possible to maintain traceability among these artifacts, and consequently ease the process of developing and rebuilding them, if necessary. Testing a SPL includes the core asset, product specific assets, and their interactions.

In this effect, RiPLE-TE consists of two processes, considering the particularities of each phase, as showed in Figure 1. In Core Asset Development, when assets have to be developed with a special attention to the forthcoming reuse, the process advocates that unit and integration testing levels should be performed, whereas system and acceptance testing have to be postponed to Product Development, where the assets previously developed will be reused. Hence, knowledge produced in the core asset testing can be reused in product testing, reducing the overall effort.

Every test case created in the basis of the RiPLE-TE process can represent variability in a way that reuse is encouraged. Figure 2 shows an extract of a metamodel [21] developed in order to capture the relationship among the artifacts created in a SPL project. This figure basically depicts the Test model fragment, in which test cases are considered the main artifact to be handled.

Since each use case in a SPL project is supported by one or more test cases, the variability herein in handled as the same way than in use cases elaboration. This latter expands on the abstract use case definition, in which variability is represented in the use cases. A use case is herein composed by the entity AbstractFlow, that comprises the subentity Flow, which actually represent the use case steps. Every step can be associated with a subflow, that can represent a variation point. In both, fragments compose every (*use* or *test*) case, and are thus considered when deriving some product. This way, dependencies among test cases fragments and use case fragments make sense.

In addition the model also include other involved artifacts, such as planning and management assets.

A. Software Product Lines System Test Case Tool

The literature still lacks works describing tool support for testing software product lines [11], [17], [18]. In this effect, we propose a tool focused on the elaboration of system test cases for a SPL projects, thus encouraging reusable artifacts to be produced. We will use the test model previously mentioned (and fully described in [21]) to manage the test assets and dependencies.

Additionally, Figure 3 illustrates how the system test cases are built from use cases. According to [22], a use case goal can be decomposed recursively into sub-goals, as a result is necessary to create test cases for each fragments ensuring the use cases coverage.

1a. The use case document are composed by the use cases of the SPL project.

1b. The tool allow users to select all the use cases or part of these. The selected use cases will be extracted from the document described in 1a.

2a. Each use case can generate a system test case. When the use case is more complex the tool will generate a test cases group described below.

2b. Test Cases groups will be composed by two or more system test cases.

2c. The combination of system test cases generated by the tool will compose the *Test Case Document*. The tool also allow users to select specic test cases in order to generate customized test cases documents.

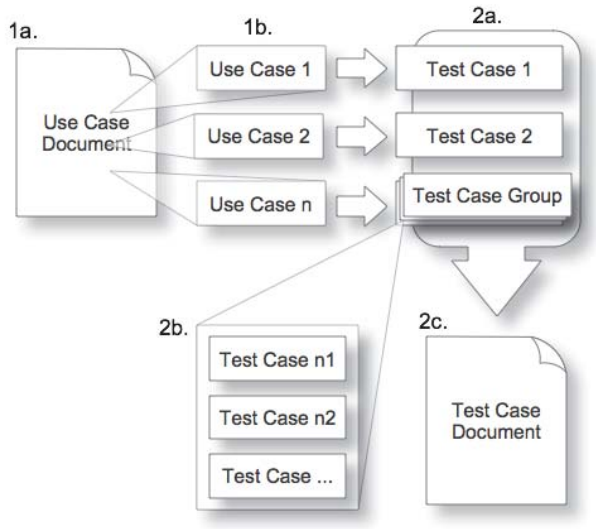


Figure 3. Proposal

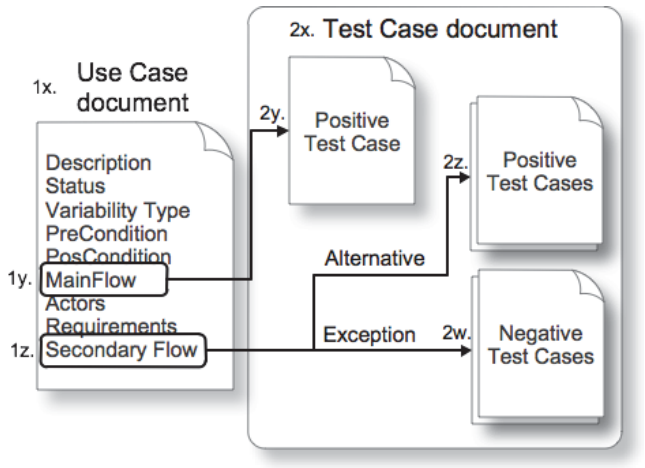


Figure 4. Test Case Composition

Next, the system test cases are formed by the following fields: an ID (unique identication number created automatically by the tool), the name of the test case, its summary and classication (positive and negative test cases), steps of the test case, pre and post conditions, variability type, use case references and screen path (describes the path of the system that should be tested).

When all the use cases were selected the application will focus at the construction of mandatory test cases. Thus the optional test cases can be built in accordance with the needs of specic products of the SPL product development phase. However, every mandatory use case, have to be validated, which consequently demands the creation of mandatory test cases. Besides test cases priority will be also classied in agreement with requirements priority dened by [8] as High, Medium and Low.

In addition to the proposal, test cases are composed by extracting information from use cases. Figure 4 illustrates the data extraction from the use case document (1x) for the construction of test case document (2x). The main flow (1y) of the use cases leads to the building of positive test cases (2y - when determined action should succeed [23]) that analyze if the main functionalities are working properly. Secondary flows (1z) of the use cases are divided in Alternative and Exception flows. Alternative secondary flows result in positive test cases (2z) that validate instructions that should succeed [23]. Finally, exception secondary flows result in negative test cases (2w - when determined action should not succeed [23]) that verifies errors messages and unsuccessful procedures.

In order to manage the variability, all the variation points are associated with requirements specified in the SPL requirement document detailed by [21]. As requirements include variability, test cases must also contain explicit variability information. Test cases that include variability are called variant test cases. Test cases without variability are called common test cases and can be applied to all applications [13].

Moreover, each variation point are related with a use case, consequently for every single use case is possible to create test cases. Hence, variation points leads to the creation of test cases in this way preserving the variability within the tests cases.

The prioritization of the variation points is established in line with the variability type and priority of the requirements. Mandatory requirements with high priority have to be examined first. For this reason we propose that tests cases of mandatory variation points with high priority should be created first.

Figure 5 explains how the test case document can support the test architect. Using the document the test architect can delegate which tests should be created by developers (Unit and Integration tests because they need source code knowledge to support White-box techniques. White-box test cases consider the entire source code of the program while grey-box test cases only consider a portion of it [24]) and which tests must be done by test analysts (System and Acceptance tests because they need part or the entire system working, will be useful to support Blackbox techniques. Blackbox test cases are those in which no details about the implementation, in terms of code, is provided. These test cases are based exclusively on the inputs and outputs of the system under test [24]).

1) Tool Architecture: [21] presents a web tool implemented using the Django[1] framework, which enabled the fast development of a functional prototype. Through Django, the metamodel mapped entities and their relationship into

[1]http://www.djangoproject.com

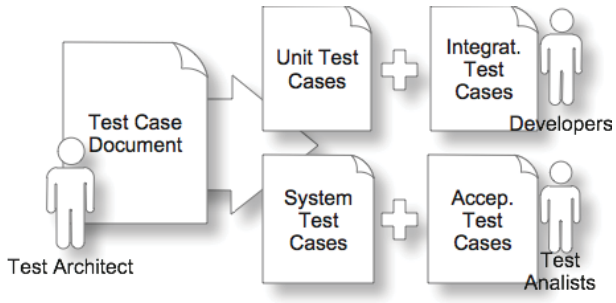


Figure 5. Using of the Test Case Document

Python[2] classes, and then it is automatically created a relational database for these entities. Finally, Django generates a Web application where it is possible to test the mapping by inserting some test data, the documentation regarding features, requirements and so on.

Currently, it is possible to use the tool to document all the assets regarding the metamodel, however the test cases derivation from use cases is not supported yet [21]. This is the focus of our proposal and to bridge this gap we propose the architecture showed in Figure 6.

In order to understand the architecture of the proposed tool is necessary to know that Django follows the Web Server Gateway Interface *(WSGI)* protocol, so we can model incoming communication as browser and server.
Figure 6 describes the architecture of the proposed tool combined with the SPL Metamodel. Details of the layers are described below:

- **Browser:** Users access the application through web browsers.
- **Template:** A Django template separates the application from the data. A template defines the presentation logic that regulate how the information should be displayed.
- **URL Dispatcher:** It defines which view is called for a given URL pattern. Basically, it is a mapping between URL patterns and the view functions that should be called for those URL patterns.
- **Model:** It describes the data structure/database schema. It contains a description of the database table, represented by a Python class. This class is called a model. Using it, is possible to create, retrieve, update and delete records in the database using simple Python code.
- **View:** It contains the business logic. View layer is itself a stack. Python view methods call a restricted template language, which can itself be extended with custom tags. Exceptions can be raised anywhere within a Python program, and if not handled they rise up as far as the environment.
- **Database:** All the data information will be recorded at the repository. Django attempts to support as many features as possible on all database backends. However, not all database backends are alike, and we have to make design decisions on which features to support and which assumptions we can make safely.

[2]http://www.python.org

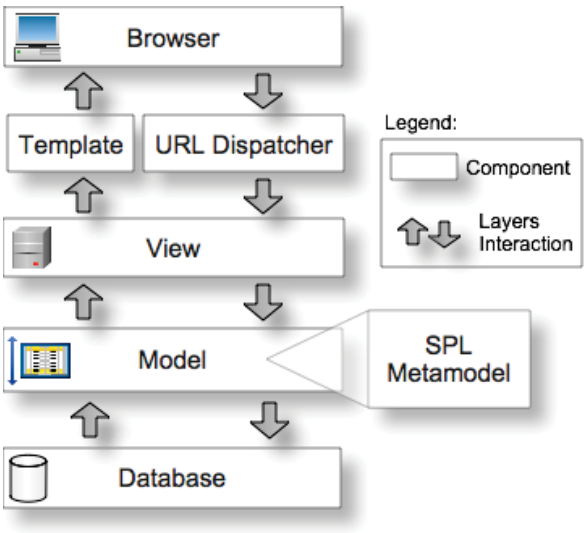


Figure 6. Proposed architecture

The view and the model layers can raise exceptions, but Python uses exceptions for control flow. So not all of these rise to the server.
The architecture of the proposed tool is suitable to support the construction of the test case document. However, to work managing source code is necessary to adapt the architecture, more details can be seen at Section VI.

VI. Concluding Remarks and Future Work

Currently, despite of the increasing interest by the research community regarding SPL testing [11], it is very difficult to find suitable tools to support SPL testing processes. It has a direct impact in the high costs this activity poses, since testing becomes a cumbersome and laborious work [17].

This work proposes a system test tool to support the test process of a Software Product Line. The tool is aimed at reducing the effort of testing, by reducing the work required to follow a SPL testing process.

The creation of system tests cases will help to manage the traceability between use cases and tests cases. It is also necessary to organize and manage the variability of the tests assets after the creation of the test case document.

During the definition of our proposal we identify the necessity of extend the architecture in order to allow support the management of source code and subsequently automatically generate unit and integration test cases.

As a future work, we are intended to conduct a series of experimental evaluations in order to analyze the effectiveness of the proposed tool.

ACKNOWLEDGMENT

This work was partially supported by the National Institute of Science and Technology for Software Engineering (INES), funded by CNPq and FACEPE, grants 573964/2008-4 and APQ-1037-1.03/08 and CNPq grants 305968/2010-6, 559997/2010-8, 474766/2010-1.

REFERENCES

[1] M. Fewster and D. Graham, *Software Test Automation: Effective Use of Test Execution Tools*. John Wiley Sons, Ltd., 1999, vol. 10, no. 2.

[2] P. Clements and L. Northrop, *Software Product Lines: Practices and Patterns*. Boston, MA, USA: Addison-Wesley, 2001.

[3] L. M. Northrop, "Sei's software product line tenets," *IEEE Software*, vol. 19, no. 4, pp. 32–40, 2002.

[4] E. S. Almeida, A. Alvaro, V. C. Garcia, V. A. A. Burégio, L. M. Nascimento, D. Lucredio, and S. R. L. Meira, *C.R.U.I.S.E: Component Reuse in Software Engineering*, 1st ed. C.E.S.A.R e-book, 2007.

[5] E. Y. Nakagawa, A. da Silva Simão, F. C. Ferrari, and J. C. Maldonado, "Towards a reference architecture for software testing tools," in *International Conference on Software Engineering & Knowledge Engineering (SEKE)*. Knowledge Systems Institute Graduate School, 2007, pp. 157–162.

[6] I. C. Machado, "Riple-te : A software product lines testing process," M.Sc. Dissertation, UFPE - Federal University of Pernambuco, Recife-PE, Brazil, Aug 2010.

[7] M. B. S. Moraes, E. S. Almeida, and S. R. L. Meira, "A systematic review on software product lines scoping," *4th Experimental Software Engineering Latin American Workshop*, pp. 63–72, 2009.

[8] D. Neiva, F. C. Almeida, E. S. Almeida, and S. R. L. Meira, "A requirements engineering process for software product lines," in *Information Reuse and Integration (IRI), 2010 IEEE International Conference on*, aug 2010, pp. 266 –269.

[9] L. L. Lobato, P. O'Leary, E. S. de Almeida, and S. R. L. de Meira, "The importance of documentation, design and reuse in risk management for spl," in *Proceedings of the 28th ACM International Conference on Design of Communication*, ser. SIGDOC '10. New York, NY, USA: ACM, 2010, pp. 143–150.

[10] V. A. Buregio, E. S. Almeida, D. Lucredio, and S. R. L. Meira, "Specification, design and implementation of a reuse repository," in *Proceedings of the 31st Annual International Computer Software and Applications Conference - Volume 01*, ser. COMPSAC '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 579–582.

[11] P. A. da Mota Silveira Neto, I. do Carmo Machado, J. D. McGregor, E. S. de Almeida, and S. R. de Lemos Meira, "A systematic mapping study of software product lines testing," *Information and Software Technology*, vol. In Press, Accepted Manuscript, 2010.

[12] M. Anastasopoulos, T. H. B. de Oliveira, D. Muthig, E. S. de Almeida, and S. R. de Lemos Meira, "Evolving a software product line reuse infrastructure: A configuration management solution," in *VaMoS*, ser. ICB Research Report, D. Benavides, A. Metzger, and U. W. Eisenecker, Eds., vol. 29. Universität Duisburg-Essen, 2009, pp. 19–28.

[13] A. Reuys, E. Kamsties, K. Pohl, and S. Reis, "Model-based system testing of software product families," *International Conference on Advanced Information Systems Engineering CAiSE*, pp. 519–534, 2005.

[14] C. Nebut, Y. Traon, and J. Jezequel, "System testing of product lines: From requirements to test cases," *Software Product Lines*, pp. 447–477, 2007.

[15] I. Sommerville, *Software Engineering*, 8th ed. Pearson Education, 2008.

[16] Q. Yang, J. J. Li, and D. Weiss, "A survey of coverage based testing tools," in *Proceedings of the 2006 international workshop on Automation of software test*, ser. AST '06. New York, NY, USA: ACM, 2006, pp. 99–103.

[17] O. O. Edwin, "Testing in software product lines," M.Sc. Dissertation, School of Engineering at Blekinge Institute of Technology, Mar 2007.

[18] A. Tevanlinna, J. Taina, and R. Kauppinen, "Product family testing: a survey," *ACM SIGSOFT Software Engineering Notes*, vol. 29, pp. 12–12, March 2004.

[19] J. D. McGregor, "Testing a software product line," CMU/SEI - Software Engineering Institute, Tech. Rep., 2001.

[20] R. Kauppinen, "Testing framework-based software product lines," *Department of Computer Science University of Helsinki Technical Report*, 2002.

[21] Y. C. Cavalcanti, I. C. Machado, P. A. M. S. Neto, L. L. Lobato, E. S. Almeida, and S. R. L. Meira, "Towards metamodel support for variability and traceability in software product lines," *5th International Workshop on Variability Modelling of Software-intensive Systems*, 2011.

[22] F. Dias, E. A. Schmitz, M. L. M. Campos, A. L. Correa, and A. A. J., "Elaboration of use case specifications: an approach based on use case fragments," *Proceedings of the 2008 ACM symposium on Applied computing*, pp. 614–618, 2008.

[23] C. Condron, "A domain approach to test automation of product lines," *International Workshop on Software Product Line Testing*, p. 27, 2004.

[24] S. Segura, D. Benavides, and A. R. Cortés, "Functional testing of feature model analysis tools: A first step," in *Proceedings of SPLiT 2008 - Fifth International Workshop on Software Product Lines Testing*, P. Knauber, A. Metzger, and J. D. McGregor, Eds., September 2008, pp. 36–39.

Scalability of Variability Management: An Example of Industrial Practice and Some Improvements

Yinxing Xue, Stan Jarzabek
School of Computing,
National University of Singapore, Singapore
{yinxing,stan}@comp.nus.edu.sg

Pengfei Ye, Xin Peng, Wenyun Zhao
School of Computer Science,
Fudan University, Shanghai, China
{072021110,pengxin}@fudan.edu.cn

Abstract— **Having set up reusable core assets for a Software Product Line (SPL), it is a common practice to apply Variation Techniques (VTs) to manage variant features. As each VT can handle only certain types of variability, multiple VTs are often employed, such as conditional compilation, configuration parameters or build tools. Our earlier study of an SPL at Fudan Wingsoft Ltd revealed potential scalability problems of multiple VTs. As a remedy to the above problems, in the follow-up study we replaced multiple VTs originally used in the Fudan Wingsoft product line, with a single, uniform VT of XML-based Variant Configuration Language (XVCL). This paper provides a proof-of-concept that commonly used variation techniques can indeed be superseded by a subset of XVCL, in a simple and natural way. We describe the essence of the XVCL solution, and evaluate the benefits and trade-offs involved in multiple VTs solution and single VT - XVCL solution.**

Keywords- Generative technique; Software Product Line; Variability management

I. INTRODUCTION

In previous paper [16], we analyzed a Software Product Line (SPL) called Wingsoft Financial Management Systems (WFMS-PL), developed by Fudan Wingsoft Ltd. WFMSes provide web-based financial services for employees and students at universities in China. Following a common practice, Wingsoft set up product architecture, identified core assets for reuse, and then applied a range of common design-time Variation Techniques (VTs), such as conditional compilation, design pattern or configuration parameters, to manage product-specific features in core assets.

Features vary in the granularity and in the scope of their impact on core assets [5][12]: *Fine-grained features* affect many core assets of an SPL, at many variation points [8]. Code of such features becomes scattered across core assets. A *Coarse-grained feature*, on the other hand, can be contained in a component (e.g., a class or package) that is included into a custom product when a given feature is needed. *Mixed-grained features* involve both fine- and coarse-grained impact.

Coarse-grained features are easier to manage than fine-grained features. Feature granularity depends to some extent on the design of core assets. Good architectural design can change feature granularity in our favor, increasing the number of coarse-grained features, and reducing the number of variation points in core assets for the features that remain fine-grained.

Variation Techniques (VT) must match feature granularity. Therefore, it is common to use multiple VTs, for example, conditional compilation to handle fine-grained features or a build tool such as Ant to handle coarse-grained features. Such VTs are easy to apply, and most of developers are familiar with them. However, as our study revealed [16], applying multiple VTs does not scale well, especially in cases of mixed-grained features. While reuse and modification of mixed-grained features is inherently difficult, applying multiple, often poorly compatible VTs aggravates the problems.

As a remedy to the above problems, in the follow-up study we replaced VTs originally used in the Fudan Wingsoft product line, with a single, uniform VT of XVCL (XML-based Variant Configuration Language) [15]. XVCL applies generative mechanisms to organize software into highly parameterized meta-components. These meta-components form SPL core assets that are adaptively reused in product derivation, automated by the XVCL Processor [9]. In this paper, we propose to use XVCL as a uniform VT to replace the original ones described in [16]. We also present an initial evaluation of benefits and trade-offs involved in adopting a uniform VT.

A practical lesson learned from our study is that in small- to medium-size product lines, applying multiple VTs may be a viable solution, as it requires less training, and variability can still be effectively managed in that way. As the product line grows in size and the impact of features on core assets becomes more complex, a company may experience problems. Then moving towards a uniform variation technique approach may be beneficial. However, this will require a more systematic approach to reuse, and training of SPL personnel.

The paper is organized as follows: Section II summarizes the findings from our earlier study of multiple VTs in WFMS-PL [16]. Section III describes the XVCL solution to WFMS-PL, and explains how it alleviates problems of the multiple VTs. We evaluate the XVCL approach in Section IV. Related work and concluding remarks end the paper.

II. PROBLEM OF MULTIPLE VARIATION TECHNIQUES

WFMS for Fudan University developed in 2003 evolved into a Software Product Line WFMS-PL [16] with more than 100 customers including major universities in China such as Shanghai Jiaotong University (WFMS for this university can be found at http://www.jdcw.sjtu.edu.cn/wingsoft/index.jsp), Zhejiang University, Chongqing University and others.

Main functionalities of WFMS include the Financial Management Subsystem (FMS) that manages all the university income and expenses, the Salary Management Subsystem (SMS) that manages salary of employees, the Reward Management Subsystem (RMS) that manages rewards for

employees and students, and the Tuition Management Subsystem (TMS) that manages student tuition fees. The TMS is a web-based portal for students to pay their tuition fee online, with functions such as login, service customization, on-line payment and history query. In addition, the TMS also provides accounting services (e.g., report generation and bill settlements) that interface universities with banking systems.

First five WFMS product variants were developed by ad hoc copy-and-paste reuse, and each WFMS was maintained as a separate product. As the number of customers was growing, ad-hoc reuse and maintenance was becoming more and more taxing on company resources. To address this problem, Wingsoft set up SPL core assets as follows: First, Wingsoft designed architecture to be shared by future products and adopted commonly available VTs, such as conditional compilation and configuration parameter files. Wingsoft did not use any advanced VTs due to the practical need for an easy-to-implement, cost-effective SPL. Such an SPL was built and reused with minimum training of the staff in SPL techniques.

We described implications of using multiple variation techniques in WFMS-PL in [16], so here we only summarize the main results. We analyzed the Tuition Management Subsystem (TMS), as it involved types of variability and variation techniques that are representative for the whole WFMS. The code of TMS is 25% of the whole WFMS system, comprising 58 Java source files, 99 JSP web pages, and several configuration files. In TMS feature model, there are 32 variant features and 9 mandatory features[1].

Wingsoft adopted simple, freely available VTs, selecting the right VT for features at hand [16]. Fine-grained features were managed by conditional compilation and commenting code in core assets. Coarse-grained features were managed by build tool Ant. Mixed-grained features were managed by configuration parameters. For the 32 variant features in TMS, conditional compilation or commenting technique was involved in 31 variant features. Ant was used by 19 variant features, overloading fields - by 13 features, configuration file - by 12 features, and design patterns - by 3 features.

Our study [16] revealed potential problems of multiple VTs in an SPL: One feature was often handled by several VTs which had to be properly synchronized. 26 among 32 variant features were managed by more than one VT, 13 features - by 3 VTs, and 3 features - by 4 VTs. The difficulty roots in the management of variant features involved in several VTs at the same time. Examples below illustrate what was involved in managing such features, and hint at complications that are bound to arise as the size of the system and the number of inter-dependent variant features grow.

Feature *PayByItem* (Fig. 1, Fig. 2 and Fig. 3) is a mixed-grained feature managed by configuration parameters that parameterize reflection, *strategy* design patterns [7], and also the build tool Ant. In Fig. 2, method *user.getPayMode()* at line 6 returns the value of the parameter *paymode* at line 2 in the configuration file (Fig. 1). According to that value, *strategy* pattern generates the specific code skeleton for feature *PayByItem*.

```
1  <webFee>
2      <paymode>PayByItem</paymode>
3      <bank-info>
4        <bankList>
5          <bank>ICBC</bank>
6          <bank>CCB</bank>
7          <bank>CMB</bank>
8        </bankList>
9        <ICBC>
10         <bankUrl>http: //mybank.icbc.com.cn/...</bankUrl>
11         <keyPath>C: /apache-tomcat-5.5.25/…</keyPath>
12         <keyPass>12345678</keyPass>
13       </ICBC>
14   </bank-info>
15   <DownloadDetail>true</DownloadDetail> …
16 </webFee>
```

Fig. 1. Managing PayByItem with configurations parameters

```
1  public class FeeOrder {
2      private Initializer initializer;
3      public init(FeeUser user, FeeInfo info,
         HttpServletRequest request) {
4          Class c;
5          try{
6              c = Class.forName( user.getPayMode() );
7              initializer = (Initializer) c.newInstance();
8              initializer.init ( . . .);
9          } catch(Exception e ) {
10             e.printStackTrace();
11         }
12 }..}
```

Fig. 2. Managing PayByItem with reflection and strategy patterns

```
1  <project name="webfee" basedir="." default="main">
2    <target name="copy-src" depends="create-folders">
3      <!-- Copy java classes of Feature PayByItem -->
4      <copy todir="${src.dir}">
5        <fileset dir="${core-src.dir}/${PayByItem}"/>
6      </copy>
7    </target>
8    <target name="copy-webpage"
9      depends="create-folders">
10     <!-- Copy webpages of Feature PayByItem -->
11     <copy todir="${web-root.dir}">
12       <fileset dir="${core-webpage.dir}/
             ${PayByItem}" />
13     </copy>
14   </target>…
15 <project>
```

Fig. 3. Managing PayByItem with Ant

Coarse-grained impact of feature *PayByItem* is managed by build tool Ant (Fig. 3). Using parameters, Ant could include/exclude source files that were relevant/irrelevant to a given feature. An Ant script sets the target path for the source code to be included in the final build. Feature *PayByItem* has a related JSP file selFeeItem.jsp, which was included by the command <**fileset dir>** at line 12.

Mixed-grained impact of features is the main source of problems for scalability of the multiple VTs to manage SPL variability. Scattered impact of mixed-grained features brings forth the difficulties to keep multiple VTs in synchronization one with another. Inter-related configuration parameters control both Ant and Java conditional compilation. There are many examples of such interactions between multiple VTs in the original TMS core assets. Its maintenance entails the accurate understanding of multiple VTs, and familiarity with variant features and core assets. As the size of the system grows and the feature dependencies increase, the above inconveniences aggravate. These observations encouraged us to experiment

[1] Details on TMS's feature particulars and feature dependency are available at this link: www.comp.nus.edu.sg/~yinxing/TMS-information.html.

with a single VT to manage the common variability situations found in TMS core assets in a uniform and traceable way.

III. SINGLE VARIATION TECHNIQUE APPROACH TO TMS CORE ASSETS

XVCL [9][15], based on Frame Technology [2], is a generative language-independent variation technique for SPLs.

XVCL encapsulates core assets in so-called *x-frames*. Coarse-grained features are contained in dedicated x-frames. Each variation point in core assets is marked with a suitable XVCL *command*, such as **<adapt>**, **<insert-before> <insert>**, **<insert-after>** and **<break>**, to enable customizations. SPL variant features are formally mapped into all the relevant variation points in core assets by means of XVCL parameters and commands. The SPeCification x-frame, called **SPC**, sets values of XVCL parameters according to feature selection. XVCL Processor interprets x-frames starting from the **SPC** (Fig. 4), traverses x-frames, propagates customization information (parameters) to them, adapting visited x-frames accordingly, and emitting code for a custom product. XVCL mechanisms allow us to manage features with fine-, coarse- and mixed-grained impact on core assets. Due to its language-independence, any type of SPL artifacts including Java code, JSP files, DB scripts, WORD files, test cases or even UML models in XMI can be consistently customized for any legal selection of features required in a custom product.

A. TMS core assets instrumented with XVCL

Fig. 4 provides a snapshot of the WFMS core assets in XVCL representation, and Fig. 5 expands some x-frames to highlight the working mechanism of XVCL [2]. The **SPC** specifies which features we need in a custom WFMS product by setting values for XVCL parameters that correspond to selected features. Values of those parameters propagate to x-frames below, navigating configuration and detailed customizations of core assets and features accordingly. **Level 2** x-frames define architecture-level customizations, in terms of configuration of core assets for a custom WFMS product. Some of the coarse-grained feature impacts are also addressed at Level 2. **Level 3** x-frames contain the actual code of core assets and features, instrumented with XVCL commands to enable customization of fine-grained features.

Features we want to select for a custom product are assigned non-empty string values, while features to be de-selected are assigned empty string values. Thus, **SPC** shown in Fig. 5 selects features *IDCard* and *SSO* (related to *Login*), and feature *PayByItem* (related to *Paymode*) for a custom product. It deselects feature *Direct*, *PayByYear* and *PayByYearOrder*.

<**select**> commands mark variation points in x-frames below **SPC**. The value of an XVCL parameter that controls <**select**> identifies an <**option**> to be processed. <**select** PayByItem> in x-frame **OnlinePayment** at Level 2 illustrates a simple variation point affected by one feature only, namely *PayByItem*. If feature *PayByItem* is selected, then the Processor emits feature code to the custom product; otherwise, <**select**> has no effect.

[2] Details on TMS's XVCL solution are available at this link: www.comp.nus.edu.sg/~yinxing/TMS-XVCL-solution.html.

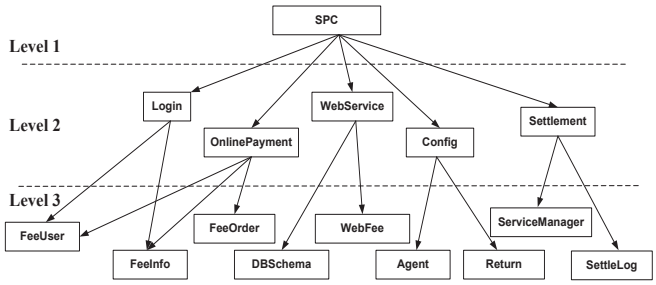


Fig. 4. Overview of WFMS core assets in XVCL

<**select** Login> in x-frame **FeeUser** at Level 3 marks a variation point affected by three features, namely *IDCard*, *SSO* and *Direct*. Notation **@v**, where v is an XVCL parameter, means reference to v's value, as assigned in respective <**set**> command. The value of Login, <**set**> to be a concatenation of the three XVCL parameters corresponding to these features, controls <**select**>, directing processing to the <**option**> corresponding to the particular combination of selected features. Note that <**option** "IDCard+SSO"> is processed whenever the two interacting features *IDCard* and *SSO* are selected. Symbol '+' is a separator. By the multiple features in <**option**>, XVCL can handle the interacted features.

XVCL parameters formally link together customizations of all the core assets affected by selected features, at all the relevant variation points. XVCL parameters set in **SPC** create a bridge between features and WFMS core assets in XVCL.

B. One variation technique instead of many

Our example of Fig. 5 also illustrates how a single variation technique can successfully provide capabilities of many variation techniques. In x-frame **FeeOrder, @PayMode** c = **new @PayMode** replaces configuration files and applications of *Strategy* pattern in the original WFMS core assets in Fig. 1, Fig. 2 and Fig. 3. Here, we need a parameterizable name of the class. Java generics support parametric types, but not class names. In the original WFMS core assets, *Strategy* pattern and configuration parameter stored in a configuration file were used to mitigate the problem (Fig. 2). *Strategy* pattern reads the name of a required class from the configuration file.

In the original WFMS core assets, architecture-level configurations of core assets and coarse-grained features were done by Ant. For example, if we select feature *PayByItem,* Ant's command <**fileset dir>** in Fig. 3 includes file selFeeItem.jsp into the custom product. The same is achieved by **<adapt>** placed under **<select>** in x-frame **OnlinePayment**. Of course, Ant has more capabilities than XVCL's <**adapt**>, but in this context only Ant's asset configuration capabilities are used.

In x-frames **DBSchema** and **FeeUser**, we see how XVCL's parameters and <**select**> replace conditional compilation and commenting out feature code. For example, feature *InitPayMode* affects **DBSchema** and **FeeUser** and is managed by conditional compilation and commenting out technique. Manual modification of the conditional compilation or comments has to been done to include/exclude features. In XVCL on the other hand, variation points are inter-lined and customizations are automated.

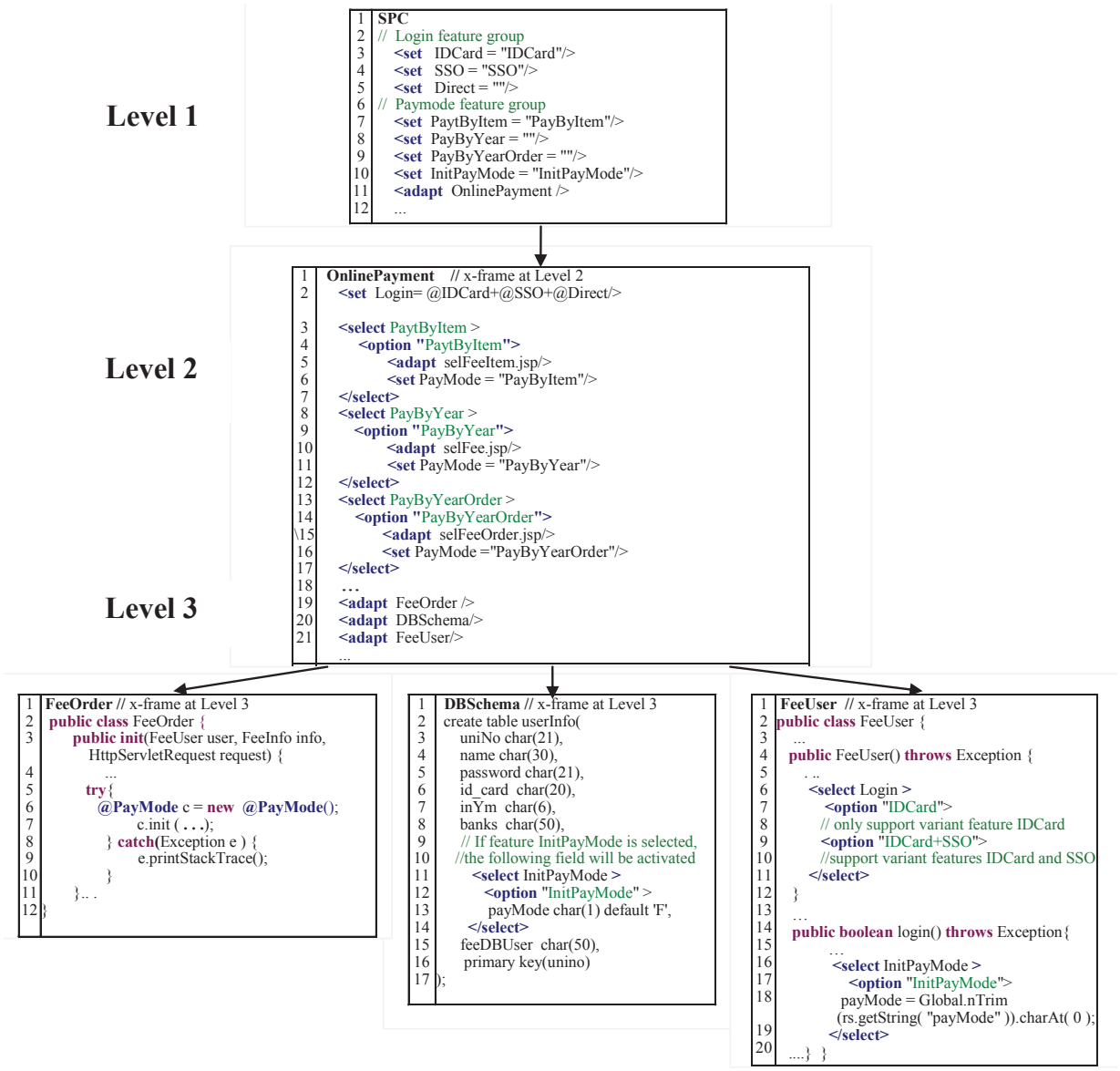


Fig. 5. Detailed view of WFMS core assets in XVCL

IV. Evaluation

What are the implications of replacing multiple variation techniques with a single one on SPL productivity? To answer this question, we conducted lab studies and collected inputs from Fudan Wingsoft Ltd. regarding the original WFMS core assets developed by Wingsoft using multiple variation techniques, and core assets in XVCL. Below, we comment on productivity during domain engineering (i.e., building and evolving core assets), and product derivation.

A. Domain engineering effort

The original WFMS core assets were built by gradual re-engineering of existing WFMSes. Core components and their interfaces were stabilized first, and then variation techniques were used to prepare them for ease of customization, as described in Section II. While it is difficult to precisely determine the effort to build core assets, we obtained some relevant information from Wingsoft engineers who were involved in re-engineering. Selecting suitable variation techniques for various features was not difficult for experienced engineers. Also, each step of applying variation techniques was quite simple. New staff joining the Wingsoft team had little difficulty to understand the variation techniques used in WFMS core assets and their role. However, some problems could be observed during evolution of the WFMS core assets. When multiple variation techniques were used together to implement a variant feature, it might not be clear how to find all the relevant variation points, and understand the exact interplay between variation techniques. Still, given the size of WFMS core assets and relatively small number of features, the solutions adopted by Wingsoft team were considered to be adequate for the purpose.

To get insights into the effort of unifying multiple variation techniques with XVCL, one PhD student and one developer re-engineered the original WFMS core assets into XVCL representation. PhD student was an XVCL expert, and the

developer was a WFMS expert, also participating in maintenance of the original WFMS core assets. It took two weeks for them to replace multiple variation techniques with XVCL in core assets for TMS subsystem. Applying XVCL was greatly simplified, as core assets were already in place, and they preserved most of the variation points. The main task was to work out overall XVCL controls and then to replace multiple variation techniques with XVCL commands at respective variation points.

Evolution of core assets involves adding new features and modifying features. The effort to evolve core assets depends on the number of variation points involved in change, and the complexity of finding, analyzing, changing variation points and tracing the impact of change. While the number of variation points in both solutions is almost the same, we assume that evolution of XVCL solution is easier than evolution of the original solution. This is due to uniform treatment of features, formal links between all the variation points relevant to a given feature, and feature query system [10].

B. Product derivation and maintenance effort

Deriving new products includes reuse of existing features, modifying features, and implementing extra features required by product customers. Similarly, the effort of each such task depends on the number of variation points involved in product customization, and the complexity of finding, analyzing, customizing variation points and tracing the impact of change.

Table I summarizes statistics relevant to product derivation effort. "Managed variation points" means variation points that have to be revised manually when reusing or modifying features. "Managed variation points" is a subset of all the variation points at which one feature affects core assets. For example, among core assets affected by feature *InitPayMode* are Java files and DB schema files. To reuse this feature in the original WFMS-PL, all affected files need to be manually changed. In the XVCL solution, once we <**set**> value of XVCL parameter **InitPayMode** in **SPC** (Fig. 5), all the customizations for feature *InitPayMode* spark from there, can be found by feature queries, and automatically performed by the XVCL Processor. Therefore, feature *InitPayMode* requires only one managed variation point in XVCL solution.

TABLE I. MANAGED VARIATION POINTS

	#variation points	#managed variation points	# files containing managed variation points
Original WFMS core assets	275	126	31
XVCL WFMS core assets	275	40	6

As another example, core assets affected by feature *Settlement* include seven Java files, four JSP scripts, one configuration file, and one file containing DB schema, totally 13 variant points. To reuse feature *Settlement* in the original WFMS-PL, we must customize code at 8 managed variation points handled by conditional compilation, comments and Ant. The location of managed variation points as well as relationship among them is not formally captured, therefore must be communicated via external documentation or re-discovered when needed. In XVCL solution, for the same feature there are also 13 variation points, but only 3 managed variation points (XVCL parameters for **Settlement** and for two dependent features). All the variation points are inter-linked via relevant XVCL parameters <**set**> in **SPC**, and reuse of the feature is automated by the XVCL Processor.

C. Other inputs from Wingsoft

Besides the above comparison study, we also collected some feedback and comments on the XVCL solution from interview with several Wingsoft engineers.

Comments on code readability. Both XVCL representations and the original final-boolean conditional compilation and commenting out applied variation techniques to embed fine-grained feature code in the code of core assets at relevant variation points. Wingsoft engineers reported about 30% of code in class FeeOrder, 20% of code in FeeInfo and 35% of code in FeeUser was managed by final-boolean conditional compilation and commenting out (the respectively similar percentages managed in XVCL representations).

Comments on copyright protection. In the original WFMS core assets, run-time binding variation techniques such as design pattern and configuration parameters are widely used. Therefore, Wingsoft engineers often included unnecessary feature code into a custom product release because of the characteristics of runtime variation binding and also time involved in feature removal. When extra functionality is contained in files that are released in readable form (e.g., JSP or XML configuration files), this practice can sometimes create copyright problems, as other customers may use extra functionality that was not meant for them or not included in their licenses. In XVCL, unwanted features are never included into a custom product, as the job of feature manipulation is consistently and automatically done by the XVCL Processor. Other than protecting copyrights, such precise and flexible control over feature inclusion/exclusion to/from custom products also matters in situations when we need to build highly optimized products, for example embedded software.

D. Evaluation summary and comments on relation work

Overall, it was felt that for small-to-medium systems such as WFMS (around 50KLOC), adopting multiple variation techniques is still practical. Variation techniques used in the original WFMS are simple and known to most of engineers. They came into engineers' mind naturally, could be applied on the fly during core asset design, with minimum disruption of conventional programming. Multiple traditional VTs provide an elementary infrastructure for SPL support. Handled by the experienced engineers, the original WFMS core assets serve well for the derivation of almost 100 product variants.

As the size of core assets and the number of variant features grows, and feature interactions get more complicated, problems may show up. Feature reuse and maintenance may become more complex because of the many variation points at which feature code needs be understood. Manual customizations become time-consuming and error-prone, even for skilled domain engineers. Then, it may be worth to consider migrating to a uniform variation technique such as XVCL.

In XVCL, for a feature reused as-is we need small number of managed variation points, at which we <**set**> XVCL

parameters for that feature and its dependent features (in **SPC**). All the variation points for a given feature are formally linked to XVCL parameter representing that feature. The ability to locate and analyze traces of customizations for each feature helps developers reuse and modify features with less errors and unwanted side-effects as compared to working with the original WFMS core assets. Reuse is automated by the XVCL Processor.

However, the adoption of XVCL is not without pitfalls, some of which XVCL shares with other variation techniques. Much of the code of features still remains tangled with core assets, affecting readability. This is a big problem, but so far alternative approaches based on specification-based variation points such as AOP [13] or FOP [3] failed to provide an effective solution to fine-grained feature management in SPLs [11] [12] .

Industrial tools such as GEARS [4] and pure::variants [14] could certainly manage the WFMS SPL. However, we do not have hands-on experience with those tools or specific studies to provide detailed comparison. GEARS can handle configurable software artifacts – such as source code, test cases and requirement documents. Its capability is similar to the XVCL. Pure::variants captures the problems (*family model*) and the solutions (*variant model,* which records the customized feature models for product variants) separately and independently, to reuse the solutions and the feature models in new projects.

The new emerging academic tool FEATUREHOUSE [1], in virtue of the FST, the direct annotation in artifacts is avoided and the readability is not undermined. The trade-off is that it has to integrate the various adapters and computation rules for the different languages. Since no annotation inside the artifacts for feature code at arbitrary granularity, we find that FEATUREHOUSE has to adopt hook a method to deal with the fine-grained impact of features. Compared with XVCL, FSTCOMPOSER in FEATUREHOUSE is flexible in supporting additional features. It cannot really change an existing fragment. But XVCL is more flexible in what can be variable.

V. CONCLUSION

This study was conducted jointly by Fudan Wingsoft Ltd., and researchers at Fudan University and National University of Singapore (NUS). Our earlier study of a Wingsoft Software Product Line (SPL) revealed that applying multiple variation techniques poorly scale to larger SPLs. In particular, it may become increasingly difficult to find and understand feature code scattered through core assets (so-called fine-grained features), and to coordinate changes at multiple inter-related variation points. Multiple variation techniques also hinder readability, reuse and evolution of core assets heavily affected by fine-grained features.

In this paper, as a remedy to the above problems, we presented an approach based on a single, uniform variation technique of XVCL, capable of managing both fine-grained features, as well as features whose impact requires customizations at the product architecture/component level. We evaluated the XVCL-based product line representation in lab experiments, and in Fudan Wingsoft Ltd, a company that initially used multiple VTs and then applied XVCL.

An SPL in our study was small, with only 39 features. Feature dependencies were few and feature impact on core assets was not very complicated. Still, we could sense some of the above problems. However, as the impact of features on core assets accumulates and gets more complex, understanding and synchronizing multiple, poorly compatible variation techniques may become difficult. Other SPLs may contain much larger code base in core assets, with hundreds or thousands inter-dependent features [6]. We believe that in case of such SPLs, the problems reported in our study are bound to become more severe, and a single variation technique approach even more attractive.

ACKNOWLEDGMENT

This work was supported by Fudan University grant (the NSF of China under Grant No. 60703092), and National University of Singapore grant R-252-000-336-112.

REFERENCES

[1] Apel, S., Kästner, C. and Lengauer, C. "FEATURE-HOUSE: Language-independent, automated software composition. ", ICSE 2009: 221-231

[2] Bassett, P., Framing software reuse - lessons from real world, Prentice Hall, 1997

[3] Batory, D., Sarvela, J.N. and Rauschmayer, A. "Scaling Step-Wise Refinement," Proc. Int. Conf. on Soft. Eng.,, ICSE'03, Portland, Oregon, May 2003, pp. 187-197

[4] BigLever. GEARS. http://www.biglever.com/, 2009.

[5] Clements, P. and Northrop, L. Software Product Lines: Practices and Patterns, Addison-Wesley, 2002

[6] Deelstra, S., Sinnema, M. and Bosch, J. "Experiences in Software Product Families: Problems and Issues during Product Derivation," Proc. Software Product Lines Conference, SPLC 2004, Boston, Aug. 2004, LNCS 3154, Springer-Verlag, pp. 165-182

[7] Gamma, E., Helm, R., Johnson, R. and Vlissides, J., Design Patterns. Addison-Wesley Professional, 1995.

[8] Jacobson, I., Griss, M. and Jonsson. P., Software Reuse: Architecture, Process and Organization for Business Success. Addison Wesley Professional, 1997.

[9] Jarzabek, S. Effective Software Maintenance and Evolution: Reuse-based Approach, Taylor & Francis CRC Press, 2007

[10] Jarzabek, S., Zhang, H., Lee, Y., Xue, Y. and Shaikh, N., Increasing usability of preprocessing for feature management in product lines with queries. ICSE Companion 2009: 215-218

[11] Kästner, C., Apel, S. and Batory, D. "A Case Study Implementing Features Using AspectJ," Proc. Int. Software Product Line Conference, SPLC'07, Kyoto, Japan, Sept.2007, pp.223-232

[12] Kästner, C., Apel, S. and Kuhlemann, M.: Granularity in Software Product Lines. In ICSE'08, 2008.

[13] Kiczales, G, Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J-M. and Irwin, J. "Aspect-Oriented Programming," Europ. Conf. on Object-Oriented Programming, ECOOP'97, Finland, Springer-Verlag LNCS 1241, 1997, pp. 220-242

[14] Pure systems. pure::variants. http://www.puresystems.com, 2009

[15] XVCL (XML-based Variant Configuration Language) http://xvcl.comp.nus.edu.sg

[16] Ye, P., Peng, X, Xue, Y. and Jarzabek, S. "A Case Study of Variation technique in an Industrial Product Line," 11th Int. Conf. on Software Reuse, ICSR09, USA, Sept. 27-30, 2009, pp. 126-136

RiPLE-TE: A Process for Testing Software Product Lines

Ivan do Carmo Machado[1,2], Paulo Anselmo da Mota Silveira Neto[2], Eduardo Santana de Almeida[1,2], Silvio Romero de Lemos Meira[2,3]

[1] Federal University of Bahia (UFBA) - Salvador, Bahia, Brazil
[2] Reuse in Software Engineering (RiSE) - Recife, Pernambuco, Brazil
[3] Federal University of Pernambuco (UFPE) – Recife, Pernambuco, Brazil

Email: {ivanmachado,esa}@dcc.ufba.br, {pamsn,srlm}@cin.ufpe.br

***Abstract*—Software Product Lines Testing has received special attention in recent years, due to its crucial role in quality and also due to the high cost this activity poses. In this effect, to make testing a feasible activity, some improvements are required. This paper presents a process for testing product lines, designed based on the gaps identified by a systematic mapping study, performed in order to understand the current scenario in this research field. An experimental study was performed in order to evaluate the proposed process in terms of understanding the role of a structured testing process in a SPL project.**

Keywords- Software Product Lines; Software Testing; Software Process; Software Reuse.

I. INTRODUCTION

Testing can be considered one of the most effective methods of quality assurance [1]. However, the state of software testing practice is not as advanced in general as software development techniques and, the same holds true for Software Product Lines (SPL) [2]. Although a number of studies can be found in recent literature on SPL Testing, including definition of processes, methodologies, modeling strategies, and other topics, it is not clear how to effectively apply testing in a coordinated, timely and cost-saving manner, together [3][4].

From an industry point of view, the growing SPL adoption points to the need of more efficient and effective testing approaches, since testing is still a difficult, expensive and challenging process, more explicitly due to the presence of variability, which hinders the way software is tested [5].

Such statements are based upon the results of a systematic mapping study performed by the authors in order to understand the current state of the art and practice of this field [3]. This study involved the investigation of 120 research papers, dated from 1993 to 2009, published by the most important vehicles in the computer science field.

The study describes that existing research on SPL testing addresses a diverse range of topics [3]. Special attention goes to the huge lack of *empirical research*, seeing that proposed techniques are mostly novel and have usually not yet been implemented in practice. Although *solution proposals* are very common in the literature, they are usually not process-centered, that can be largely applied to different context and projects. In addition, reported *proposals* do not care about *evaluation research*. Researchers are not also concerned about *experience reports* on their personal expertise using particular approaches. Practitioners in the field should report results on the adoption, in *real projects*, of the techniques proposed and reported in the literature. In addition, authors should express opinions about the desirable direction of SPL Testing research, expressing their experts' viewpoint.

Our findings are in line with a similar study [4], which also surveyed existing research on SPL testing in order to identify useful approaches and needs for future research.

Based on the findings [3], we decided to initially work in the definition of a structured process to support testing in SPL projects. The proposed process is composed of a set of activities, tasks and roles, encompassing the both SPL phases [6]: *core asset development (CAD)* and *product development (PD)*. We believe that such effort can be an initial endeavor to filling out the gaps of existing approaches, enhancing the SPL testing practice.

An empirical evaluation was performed on the basis of the proposed process. It consisted of an experiment, performed on the unit test level of the process. It involved 30 undergraduate students. Statistical techniques were applied to analyze gathered data. Our decision about including only one test level was based on the idea of incremental construction of knowledge about the role of testing procedures for SPL. We believe that solving every step once at a time could raise better results and accordingly we could extract more detailed feedback.

The remainder of this paper is structured as follows: Section II outlines the research field. Section III describes the proposed approach. Section IV presents the evaluation performed. Section V gives an overview about related work and finally Section VI concludes the paper and outlines future work derived from this research.

II. TESTING IN SOFTWARE PRODUCT LINES

Among the aspects that differentiate testing in single systems development (SSD) and SPL, ***variability*** should be mentioned as the most important issue to be handled, since its exploitation can lead to a huge combinatorial explosion of configurations a system can support. It directly reflects in the testing activities, because the large amount of variability in a product line increases the number of possible testing combinations.

Indeed, testing all combinations is almost impossible in practice. This is the main reason for some authors to consider testing as the bottleneck in SPL, since the cost of testing product lines is becoming increasingly more costly than in SSD [7]. High cost makes testing an attractive target for improvements [8].

In this effect, a systematic testing approach can save significant development effort, increase product quality and customer satisfaction, and also lower maintenance costs. Hence, techniques that consider variability issues and coordinate testing activities are desirable.

A systematic approach for testing has to consider the both SPL phases, *CAD* and *PD*, in which in the prior the assets are designed and built considering the possible variable parts to be further reused, in the latter. In PD phase, products are instantiated, on the basis of assets from the core asset base. This strategy can foster reuse of assets. The systematic reuse of assets enables efficiencies in development time and cost. Creating document templates and abstract test cases may serve, for example, as a feasible strategy [8].

It is just as important to mention that the different test levels should be considered. The goals of *unit*, *integration*, *system* and *acceptance* testing levels should be combined with the goals of the SPL phases.

Evidences collected from the studies analyzed and reported in our previous work [3] points out that these mentioned aspects constitute the core elements when considering testing SPL.

III. RIPLE-TE: A PROCESS FOR TESTING SPL

In line with the results of the mapping study, through exploiting the gaps reported in the literature, we have devoted effort towards designing a structured process to support the test discipline in SPL projects. The design is based on the RiPLE - *RiSE Product Line Engineering* - project. This project is an effort[1] of developing a framework to support a set of disciplines that compose the lifecycle of a SPL, namely Scoping, Requirements, Design, Implementation, Testing and Evolution Management. RiPLE-TE is the name given to the test discipline in this framework.

Testing with RiPLE-TE comprises the core assets, product specific assets, and the interactions among these. The process considers particularities of each of these SPL phases.

- In CAD, when assets have to be developed with a special attention to the forthcoming reuse, attention goes to the execution of *unit* and *integration* test levels.

In a first moment testing should be concerned about evaluating assets from its initial development effort. Hence, in RiPLE-TE the initial effort is devoted to *unit testing*. Components are here considered the *units*. Then, in order to ensure that a component may be further reused, it should be tested, under planned conditions. Although coupling and cohesion are considered the cornerstones in modular software development, provided by component-based development, in practice it is usual to work with tightly coupled units. In these cases, after performing unit tests in a component, and ensure that it fulfills what it was specified to, *integration* tests are then performed.

These levels are responsible for detecting different types of faults. While unit testing independently tests methods, classes, and the interaction among these pieces that comprises a component, whereas integration testing is responsible for testing the interaction among components interfaces and the integration between modules.

- In PD, *integration*, *system* and *acceptance* testing is performed, based on the assets previously developed and tested. In special cases *unit* tests can also be performed in PD, as explained next. The purpose of such a division is linked to the role of testing in every phase, as advocated by other approaches, e.g. [8].

Integration testing is the first level to be performed. The point is why to perform it again. Given that the purpose is to avoid repetition and thus consequently reduce the overall testing effort, in CAD it is only performed the integration between tightly coupled units, regardless integrating the whole set of components. It regards the behavior of the core asset base, in which there are several components, attending to a diverse range of variations, that not necessarily integrate with each other, but indeed should be ready for instantiation in PD. This is when integration testing in PD takes place.

In PD, the integration test of the components that will comprise a product is performed, in order to ensure the workability of the interconnected modules as well.

Next, *system testing* is performed. This level is then focused on evaluating the product as a whole, intended to detect system architectural and/or end-to-end defects. At this point, testers evaluate the system against system requirement specification.

Acceptance tests are carried out after system testing, in order to gather customer feedback on the product just instantiated. It is more a demonstration to user that intends to show that the product does what is expected.

As mentioned before, *unit tests* may be performed in PD. It may happen whenever a new requirement or feature that does not belong to the core asset base yet is probably to be included in a specific product. As this new artifact is built, unit tests have to be performed. Then, the remaining levels should also be performed, considering the new artifact. After that, the new "asset" can be included in the core asset base, and be used in next products, thus motivating reuse. This way, we handle the *interactions between CAD and PD*, as previously stated.

RiPLE-TE is indeed not focused on evolution issues, as the probable "propagation" of assets from PD to CAD afore-mentioned. The boundary between testing and evolution is rather supported by another discipline in RiPLE, the RiPLE-EM (Evolution Management) [9].

The main workflow of RiPLE-TE is presented in Figure 1, which represents both SPL phases and associated levels. Feedback is a closure activity common to all these levels.

[1] RiPLE is a RiSE project - http://labs.rise.com.br/

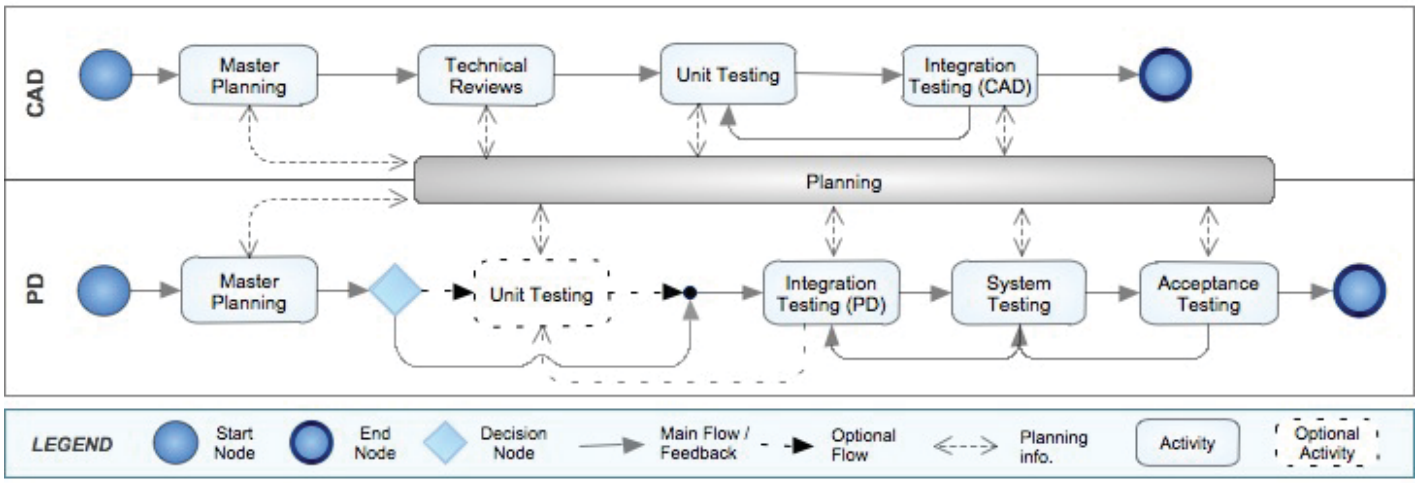


Figure 1. RiPLE-TE CAD and PD Workflow.

This Figure shows the activity *master planning* as the initial point in both CAD and PD. It refers to the development of a *Master Test Plan*, artifact responsible for orchestrating the whole testing activity. This plan is based on [10], and defines: *how testing will be done; who will do it; what will be tested; the coverage criteria;* and *schedule associated with.* As all levels should be planned, not only in the beginning, Figure 1 includes a support activity called planning, linked to all activities, to illustrate the need of performing the activities in a coordinated manner, following a plan that can be continuously updated.

Furthermore, it is desirable that project members can detect and get rid of errors/defects early in the life cycle, even before any code is available [11]. In CAD the activity *technical review* is responsible for reviewing the main assets of a SPL project that directly impacts the testing activities, such as: *Feature Model, Product Map, Requirements (and Use Cases), Feature Dependency Model,* and *the Architectural Document (Design)*. This activity should be performed before starting the tests.

In overall, testing in CAD tries to minimize the PD testing. It can be achieved through preserving variability in core assets. Knowledge produced in the CAD testing can be reused in PD testing, which may reduce the effort of building assets. This will be detailed next.

A. Managing Test Assets Variability within RiPLE-TE

In all activities mentioned from both CAD and PD testing, each level encompasses four main tasks: *planning*, *design*, *execution* and *reporting*. Figure 2 shows the process workflow, which also includes the notion of Change Requests (CR), which is handled by RiPLE-EM. In decision node 1, if any error was found during execution that needs to be supported by a CR, RiPLE-EM discipline is invoked.

The feedback flow represented in the Figure referring to the decision node 2 asks whether the coverage criteria defined in planning was achieved. If yes, then the flow gets to the end node. Otherwise, planning is revisited and the main flow is performed again.

Although the remaining flows are presented as sequentially initiated, this is an iterative and incremental activity, with feasible feedback connections enabling refinements.

Due to limitation constraints in this paper, the complete and detailed list of steps, artifacts, input, outputs, and roles associated can be found at the RiPLE website[2] and also in [12], the M.Sc. dissertation this work is associated with. Our proposal indeed does not introduce new terminologies for testing, but rather tailors existing patterns [10][13].

The management of assets in RiPLE-TE is defined under the idea that every test case created may represent variability. Our previous effort was to define a meta-model in order to capture the relationship among the artifacts created in a SPL project [14]. In the metamodel, the RiPLE-TE works mainly in the relationship of use cases and test cases.

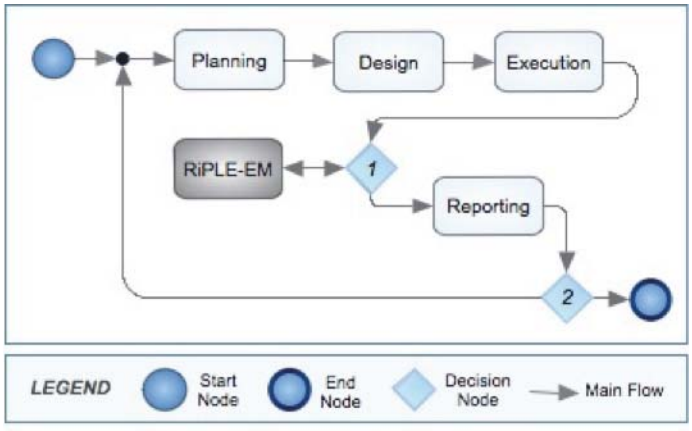


Figure 1. Testing Workflow.

In the test model, test cases are considered the main artifact to be handled. Every use case supports one or more test cases. The use case elaboration expands on the abstract use case

[2] http://riselabs.dcc.ufba.br/riple/

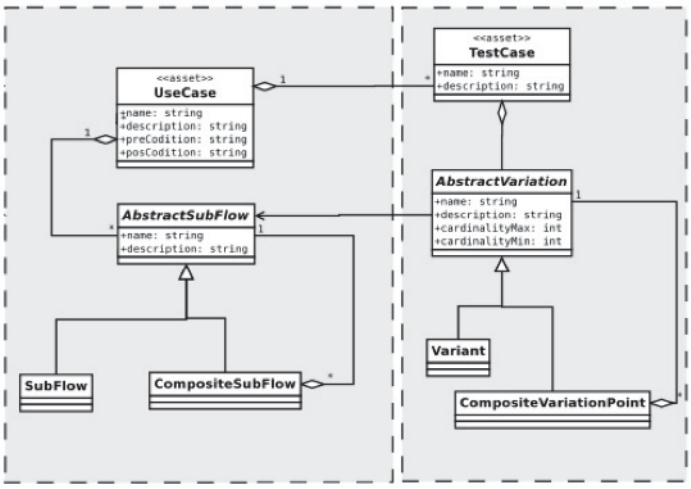


Figure 1. An excerpt of the metamodel [14].

definition, in which variability is represented in the use cases. A use case is composed by the entity *AbstractFlow* that comprises the sub entity *Flow*, which actually represent the use case steps. Every step can be associated with a sub flow that can represent a variation point. Figure 3 shows an excerpt of the metamodel.

Reuse is encouraged in a sense that, both use cases and test cases are composed by fragments. Each fragment represents a recurring set of interactions required to achieve a sub-goal. As each fragment deals with some specific sub-goal, it can be customized and fit variation point needs. This strategy is built on the work described in [15]. Dependencies among fragments in test cases and use case can be tracked.

A tool has been under development that will support the RiPLE-TE, but so far we have only counted on manually performed tasks.

IV. INITIAL EVALUATION

This initial evaluation was focused on trying to understand the role of unit testing in SPL, using the RiPLE-TE, according to the process defined in [16].

A. Definition

Based on the GQM [17], the experimental study goal was to analyze *the RiPLE-TE Process, Unit Test level for the purpose of evaluation with respect to its effectiveness and to identify which professional skills have impact on the test activity results from the point of view of the potential users in the context of a SPL testing project at the University.*

We were intended to answer the following questions:

- ***Q1.*** *Do the quality of detected defect improve when the process is followed?*
- ***Q2.*** *Does the rate of defect detection increase when the process is followed?*
- ***Q3.*** *Does the test coverage rate increase when the process is followed?*
- ***Q4.*** *Which professional skills have influence on the results of the testing activity?*

Since the literature [3][4] does not provide metrics directly associated to SPL testing, we applied known practices from SSD testing, with some adaptations to reflect SPL issues.

- ***M1. Test Case Effectiveness (TCE):*** the ratio of the amount of defects reported (D_{tot}) to the total number of test *cases* (N_{tc}). *TCE* metric validates the effectiveness of functional test cases. This metric refers to *Q2* and *Q4*. Defined as: $TCE = (D_{tot}/N_{tc}) \times 100$.
- ***M2. Quality of Defects Found (QDF):*** the number of valid defects found, normalized to Difficulty (DD) and Severity (SV) *values*. Every defect is valued with a coefficient (*k* and *r*) according to its DD and SV. *M2* refers to *Q1* and *Q4*. Defined as: QDF = k.f(DD) + r.f(SV).
- ***M3.Test Coverage (TC):*** gives the fraction of all features (or requirements/use cases) covered by a selected number of test cases or a complete test suite. Coverage (C_{ov}) was, in this study, generated by the *Eclemma* code coverage tool, and JUnit framework. *M3* refers to *Q3* and *Q4*. Defined as: TC = C_{ov}.

B. Preparation

Design. *One factor with two treatments*. We compared the two treatments against each other [16]. Factor in this experiment was the *RiPLE-TE unit testing process* and treatments were: (1) *Testing with the process*; and (2) *testing without it*. The participants were divided in two groups, in which one performed testing using the process and the second performed in an ad-hoc fashion.

Experiment materials. Objects used in this experiment was as follows: *Consent Form, Background Questionnaire, Test Assets and Component Source Code, RiPLE-TE Documentation - including guidelines and usage samples, Error Reporting Form, Feedback Questionnaire*. A copy of common artifacts was provided: *master test plan, unit test plan template, feature model, project code - components to be tested and specification - requirements and use cases*. *Error reporting forms* were also provided.

All participants signed a **consent form**, as a means of agreeing to join the study, and filled in a questionnaire, with their background information, the **background questionnaire.** They provided information about their experience with software development, SPL, testing, and the tools required by the experiment, and expertise in industrial projects.

There were designed three types of **feedback questionnaire**: (a) addressed to the group that did not use the process; (b) another to the group that applied the RiPLE-TE. The questionnaires were designed to provide information on the use of the approach, through gathering information about the participants' satisfaction; (c) designed to gather feedback on the subjects that did not use the process, regarding their opinion about the possibility of finding more defects, if they had used the process. This was answered after the last training session, when the RiPLE-TE was addressed.

Hypotheses. The Null Hypotheses are stated considering that there is no benefit of using the process (*RP*), if compared to ad-hoc testing (*AH*), in terms of effectiveness. They are:

H_{01}: $\mu TCE_{AH} \leq \mu TCE_{RP}$

H_{02}: $\mu QDF_{AH} \leq \mu QDF_{RP}$

H_{03}: $\mu TC_{AH} \leq \mu TC_{RP}$

Conversely, alternative hypotheses state the opposite values. Due to space constraints, we omitted them.

Variables. Dependent variables were the metrics TCE and QDF. The independent variables were the background information of the participants, due to our intention of evaluating the effectiveness of the proposed approach and to identify which skills could interfere in the results.

Project. It was chosen a SPL project in the domain of conference management. The goal of the project is to develop the *RiSE Chair* product line, targeted at the submission/ management of papers in conferences, journals, and others, including the control over the review life cycle. The RiSE Chair platform was based on commonality and variability among systems such as: *EasyChair*, *CyberChair* and JEMS[3]. The project is composed of 41 features extracted from the scoping analysis of such systems. The code was developed in the J2EE platform, with Spring and Hibernate, implementing variability in 8 core components.

Participants. 30 undergraduate students taking a V&V course, from Computer Science department at Federal University of Bahia, Brazil. This course is targeted at last year undergraduate students. All students had previously enrolled in courses on OOP, Java, and Testing.

C. Execution

The experiment was conducted from November to December in 2009, according to the definition and planning documented. From the amount of components of the developed SPL, we selected one feature, with its variants, and dependencies, to our experiment.

Participants were said for not to implement new features, but rather analyze the components they were given and implement the test cases to evaluate existing assets.

D. Analysis

Test case effectiveness. In terms of valid defects found, in group 1 (G-1) (without run the process), the *mean value* was 6.188 with an *standard deviation (sd)* of 3.187, while in group 2 (G-2) (with the process), the *mean* was 3.857, with a *sd* of 3.505. Regarding designed test cases, in G-1 the *mean* was 10.38 with a *sd* of 5.137 and 8.143 with a *sd* of 3.670 in G-2. By applying the TCE formula, the *mean* in G-1 was 0.725 with a *sd* of 0.540. In G-2 the *mean* was 0.425 with a *sd* of 0.353. *Median* of G-1 was slightly greater than in G-2.

Quality of defects found. What we mean by defect category refers to defects found that, although described in many different ways, they expressed similar problems in the code. We arranged these similar problems into categories. Then, we identified 12 groups of defects, and classified them according to associated Difficulty (DD) and Severity (SV). The classification was done based on a discussion performed with experts, considering their industry expertise. Each group of defect was assigned to a coefficient. As there were no baseline values to these coefficients, we performed PCA calculations [18], to identify such values. These formed the elements to calculate the score of quality. As result, the *mean* in G-1 was 8.868, and a *sd* of 3.896; in G-2 the *mean* was 6.048, and a *sd* of 5.302.

Test Coverage. While in G-1 the *mean* was 0.663 with a *sd* of 0.136, whereas in G-2 the *mean* was 0.630 with a *sd* of 0.364.

Hypothesis Testing. Regarding *TCE*, t-test (unpaired, two-tailed) was applied and resulted in a p-value higher than 0.05, which indicates that the Null Hypothesis H_{01} could not be rejected. H_{02} could not also be rejected, since calculated p-value was higher than 0.05. Thus, we can conclude that there was no gain using the process instead of an *adhoc* fashion, regarding *QDF*. TC_{ov} presented apparent similar values, but when applying t-test in this sample resulted in a very high *p-value*, which lead us to conclude that means are extremely different. In TC_{ov}, then, we could not draw conclusions.

Interpretation. Considering absolute values, the results presented insights that enabled us to infer that unit testing in SPL does not have impressive differences if compared to SSD, since practitioners which did not have experience in SPL projects had slightly better results than others who applied a formalized process. However, the model extracted from the multivariate regression analysis (MRA) [18], which correlated all variables, did not return satisfactory results, in a sense that, much was collected but a very small amount of variables impacted the results.

The insights served as a first step towards defining, and in long-term, refining, a process for testing SPL. Moreover, although the results have presented results very different from expected, in terms of the use of a formalized process for SPL, this experiment served for us to sketch models using MRA, and to define baselines for future experiments.

V. RELATED WORK

In [19] an approach providing a method to derive scenarios to be tested is proposed. It deals with textual use cases as the central development artifact. Use cases are extended with tags, which describe the variability. Test specifications containing variability are derived from these use cases. It handles how to manage variability and how to instantiate test cases for a specific product.

In [20], a method for automatic generation of test cases associated to specific products is provided, from sets of incomplete and generic scenarios associated to a product line. They propose a two-step process, from test requirements to product-specific test cases. From use cases, they structure scenarios to produce reusable test patterns common to an entire product line and represented using the UML. When the final design is available and a product is chosen as test target, test cases for that particular product are synthesized.

In [21], an approach to support the systematic reuse of assets for system and integration testing of SPL is provided. It describes test cases from requirements, by using UML activity diagram to represent all possible use case scenarios for a use case.

These are representative approaches to the field, but they have limitations that prevent their larger application. They do not provide elements of software process, encompassing the

[3] http://submissoes.sbc.org.br/

whole life cycle and the SPL phases. Besides, little is reported regarding validation. Only [21] did it. However, replications would be difficult since no details but rather only the results are described. It does not invalidate the proposed process, but it may be infeasible to implement it in contexts other than what was reported. Hence, more practical evidence is required.

In our proposal, we use the test case generation methods from these approaches, since we considered them as well defined practices, but we also give details that are not provided by them, in terms of activities sequence, inputs and outputs, and so on.

VI. Concluding Remarks and Future Work

In this paper, the RiPLE-TE process for testing SPL has been introduced as a process-centered strategy to coordinate the testing activities. It was designed based upon results gained from a systematic mapping study performed aimed at providing evidences about this research field and sketch a set of open rooms for improvement [3].

The RiPLE-TE process was based on best practices reported in the literature, and focused on a direction that no existing approaches had been concerned about, the coordination of test activities in the whole lifecycle and the SPL phases. This way, our proposal works out as a complementary solution to existing ones.

In addition, based upon the underlying assumption that effective processes are required due to the additional complexity of testing SPL because of assets variability [3], we designed an experiment in order to investigate such assumption through evidences. Hence, our initial evaluation was focused on understanding the role of RiPLE-TE process, unit test level, in a SPL project. This served as an initial step towards clarifying peculiar characteristics of different test levels in a SPL project. A formal basis of experimentation was applied in this work [16].

We have been working towards incrementally building knowledge about the role of testing procedures in SPL. Thus, as future work we are intended to empirically investigate other test levels, based on this proposed process and also making comparisons with approaches that support other levels. Besides, we are also intended to investigate how variability affects test cases and explosion of test cases, and how we manage that and how we organize SPL test cases. Our model for representing test assets and their interaction with other artifacts [14] was a first step towards such understanding.

Acknowledgements

This work was partially supported by the National Institute of Science and Technology for Software Engineering (INES), funded by CNPq and FACEPE, grants 573964/2008-4 and APQ-1037-1.03/08, CNPq grants 305968/2010-6, 559997/2010-8, 474766/2010-1 and FAPESB grant 783/2010.

References

[1] R. Kolb and D. Muthig, “Challenges in testing software product lines,” CONQUEST - 7th Conference on Quality Engineering in Software Technology, pp. 81–95, 2003.

[2] A. Tevanlinna, J. Taina, and R. Kauppinen, “Product family testing: a survey.” ACM SIGSOFT Software Engineering Notes, vol. 29, no. 2, p. 12, 2004.

[3] P. A. M. S. Neto, I. C. Machado, J. D. McGregor, E. S. Almeida, and S. R. L. Meira, “A systematic mapping study of software product lines testing,” Information and Software Technology, vol. 53, no. 5, pp. 407 – 423, 2011.

[4] E. Engström and P. Runeson, “Software product line testing - a systematic mapping study,” Information and Software Technology, vol. 53, no. 1, pp. 2 – 13, 2011.

[5] M. Jaring, R. L. Krikhaar, J. Bosch,“Modelingvariability and testability interaction in software product line engineering,” in 7th International Conference on Composition-Based Software Systems, 2008, pp. 120–129.

[6] P. Clements and L. Northrop, Software Product Lines: Prac- tices and Patterns, Boston, MA, USA: Addison-Wesley, 2001.

[7] R. Kolb and D. Muthig, “Making testing product lines more efficient by improving the testability of product line architectures,” in ROSATEA: ISSTA workshop on Role of software architecture for testing and analysis, 2006, pp. 22–27.

[8] J. D. McGregor, “Building reusable testing assets for a software product line,” in 14th International Conference on Software Product Lines: Going Beyond, ser. SPLC’10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 505–506.

[9] M. Anastasopoulos, T. H. B. Oliveira, D. Muthig, E. S. Almeida, and S. R. L. Meira, “Evolving a software product line reuse infrastructure: A configuration management solution,” in VaMoS - 3rd Workshop on Variability Modeling of Software-Intensive Systems, Seville, Spain, 2009, pp. 19–28.

[10] IEEE Standard for Software Test Documentation - 829:1998. IEEE Computer Society, 1998.

[11] C. Denger and R. Kolb, “Testing and inspecting reusable product line components: first empirical results,” in International Symposium on Empirical Software Engineering, 2006, pp. 184–193.

[12] I. C. Machado, “RiPLE-TE: A software product lines testing process,” M.Sc. Dissertation, CIn - Informatics Center, UFPE - Federal University of Pernambuco, Recife-PE, Brazil, Aug 2010.

[13] J. D. McGregor, “Testing a software product line,” CMU/SEI - Software Engineering Institute, Pittsburgh, PA, USA, Tech. Rep. CMU/SEI-2001-TR-022, ADA401736, 2001.

[14] Y. C. Cavalcanti, I. C. Machado, P. A. M. S. Neto, L. L. Lo- bato, E. S. Almeida, and S. R. L. Meira, “Towards metamodel support for variability and traceability in software product lines,” in VaMoS - 5th Workshop on Variability Modeling of Software-Intensive Systems, Namur, Belgium, 2011, pp. 49– 57.

[15] F. G. Dias, E. A. Schmitz, M. L. M. Campos, A. L. Correa, and A. J. Alencar, “Elaboration of use case specifications: an approach based on use case fragments,” in SAC - ACM Symposium on Applied computing, Fortaleza, Ceara, Brazil, 2008, pp. 614–618.

[16] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wessle ́n, Experimentation in software engineering: an introduction. Norwell, MA, USA: Kluwer Academic Publishers, 2000.

[17] V. R. Basili, G. Caldiera, and H. D. Rombach, “Goal question matric paradim,” in Encyclopaedia of Software Engineering, vol. 2. John Wiley & Sons, Inc., 1994.

[18] I. T. Jolliffe, Principal Component Analysis. Springer, 2002.

[19] A. Bertolino and S. Gnesi, “Pluto: A test methodology for product families,” in Software Product-Family Engineering, 5th International Workshop, PFE, Siena, Italy, 2003, pp. 181– 197.

[20] C. Nebut, S. Pickin, Y. L. Traon, and J. Jézéquel, “Reusable test requirements for uml-modeled product lines,” in REPL: Proceedings of the Workshop on Requirements Engineering for Product Lines, Essen, Germany, 2002, pp. 51–56.

[21] A. Reuys, S. Reis, E. Kamsties, and K. Pohl, “The scented method for testing software product lines,” Software Product Lines - Research Issues in Engineering and Management, pp. 479–520, 2006.

An Agile Scoping Process for Software Product Lines

Marcela Balbino[1], Eduardo Santana de Almeida[2, 3], Silvio Meira[1, 3]

[1] Federal University of Pernambuco (UFPE) – Recife, Pernambuco, Brazil
[2] Federal University of Bahia (UFBA) – Salvador, Bahia, Brazil
[3] Reuse in Software Engineering (RiSE) – Recife, Pernambuco, Brazil

Email: {mbsm, srlm}@cin.ufpe.br, esa@rise.com.br

***Abstract* – The constant changes and the search for new benefits in the software industry make possible the emergence of new research areas. In this context, a trend that has presented important benefits is the area of agile software product lines. Software Product Lines (SPL) and Agile Methods (AM) both search to satisfy the customer, increase quality and decrease time-to-market and costs. The integration between SPL and AM is a challenge, but can increase the benefits that these approaches offer. In this sense, this paper proposes an agile scoping process for SPL.**

Keywords – Scoping, Software Product Lines, Software Process.

I. INTRODUCTION

The main aspect for software development processes adoption, with objects or not, is the idea of reuse. The idea is to build software across the use of existing artifacts or knowledge.

The implications for decreasing development time and costs and quality increase make the reuse approach highly attractive. In this context, one of the reuse approaches with significantly and increasing success is Software Product Lines (SPL) [5].

In order to obtain success in the SPL development, the planning should be made carefully. The planning performed in the initial phase of software product lines development is called *scoping*. Product lines scoping is the phase by which information used in software systems development within a domain is identified, captured and organized with the purpose of making it reusable when building new products [2]. The literature has highlighted the importance and benefits of scoping for the product line success [13][14][15][21].

On the other hand, another success practice in industry is agile methods that encourage strong business involvement in development activities, focus only on the requirements at hand with incremental planning and design. Initial evidences based on the results of ongoing research suggest that agile methods are advantageous, given the right context [3][6]. In spite of the differences, when compared to SPL, AM has the same overall objective: improve software development productivity. In this sense, recently, the community started to investigate if these approaches can be combined and how the agile aspects can be integrated in product lines processes for maximizing the benefits searched by product lines and agile methods [7][10] [11][18][19][20][24]. However, even with the initial efforts, specific areas have been few explored on this perspective, such as scoping. In this sense, this paper presents a scoping process for SPL based on some agile principles.

II. PROCESS OVERVIEW

For more than one decade, the SPL scoping has been investigated [5][8][13]. However, the studies do not address the issue "*how to perform systematic and agile scoping*" which starts a new research direction.

The importance of scoping to be systematic in SPL is based on the idea that it should manage variabilities and commonalities among several applications, factor which increases its complexity compared to traditional software development. Thus, in order to develop a SPL is needed systematic scoping to obtain positive results. On the other hand, as SPL scoping is totally defined up-front, it demands high effort and costs. In this sense, introducing agile aspects in SPL scoping enables to add agility for scoping decreasing effort and costs.

In this context, a systematic and agile scoping process with tasks, guidelines, inputs, outputs and well defined roles, incorporating agile practices in their lifecycle is important to decrease the risks related to product lines and decrease the effort and costs up-front.

Based on this scenario, this work presents RiPLE-SC, an agile scoping process for SPL. It is part of the RiPLE (The RiSE Process for Product Line Engineering), project that is being developed within Reuse in Software Engineering (RiSE) Labs[1] to develop a process covering the full SPL lifecycle.

RiPLE-SC is the first discipline of the RiPLE as can be seen in Figure 1 and has a direct relationship with the process of Requirements Engineering (RE), Risks Management (RM) and Evolution Management (EM) from RiPLE.

The RE activities are responsible for refining the scope, thus, the scope definition is a pre-requisite to start the RiPLE-RE which has the *Product Map* artifact (a matrix of features and products generated by the RiPLE-SC) as a mandatory input. The RM activities are related to the identification and management of the risks related to scoping as well as all the risks identified during the RiPLE execution. If some risk is identified, it is catalogued by the RM and techniques for risk management are applied to mitigate it.

The communication between SC and EM is performed by *Change Requests* and maintenance tasks in the artifacts created during the scoping phase. If some artifact of SC is incomplete

[1] http://labs.rise.com.br.

or inconsistent and it is discovered in some of the phases of the scoping process, the SC requests a change to EM. The EM analyzes the change and when approved, sends a task to SC.

In the composition of RiPLE-SC, there are phases related to tasks, roles and work products (inputs and outputs). The roles defined in the process are: **scoping expert** - responsible to conduct the scoping process, driving the workshops performed in the process; **customer** - it has critical role in the process and their presence is essential for that the product lines products present their real needs; **domain expert** - provides their knowledge on the domains and the products related to it; **market analyst** - provides knowledge on market analysis and can help in the identification of the domains and products more relevant for a determined market segment; **developer** - important for the *assets scoping phase*, where they define estimative related to effort; **architect** - responsible to indicate which features will constitute the reference architecture; and **product line manager** - responsible by providing the organizational goals.

Regarding the phases, they are iterative, incremental and clearly integrated with agile aspects, such as: review meetings to obtain customers' feedback; creation of user stories to obtain the real needs from the customers; pre-scoping meeting to gather the different visions of the stakeholders; and so on. The phases are discussed in details in the next section.

III. PHASES AND GUIDELINES

RiPLE-SC consists of four main phases that are performed in an iterative way to enable frequent feedback: ***(A) Pre-Scoping***, ***(B) Domain Scoping***, ***(C) Product Scoping*** and ***(D) Assets Scoping***. Besides, we believe that during the achievement of theses phases it is very important the use of *reflection meetings*, meetings that should be performed at regular intervals in order to discuss, what is good, what should be improved and what should be performed to improve, adjusting the behavior of the team adequately.

A. Pre-Scoping

This phase is composed of two tasks: *pre-scoping meeting* and *analyze market*. In it, relevant characteristics that will influence the next stages of the scoping process such as project vision, operational and organizational context, stakeholders and roles, business goal and market potential are identified.

1) Pre-Scoping Meeting

This task has as objective to identify general information related to: customer, team and organization. Moreover, this task aims to provide an initial contact between the customer and the project team and exposes the agenda that will drive the application of the process.

The identification of the information necessary from the customer, project team and organization can be obtained with the use of questionnaires and semi-structured interview that drive the *pre-scoping meeting*. The meeting is conducted by the scoping expert through informal conversation with the stakeholders, factor that encourages the customer collaboration in the project and the collaboration inside the team. The active involvement of the customer and the cooperation among all the stakeholders is essential for the success of agile projects [9]. Besides, the communication face-to-face is considered by the Agile Manifesto as the most efficient and effective method of conveying information in a project team.

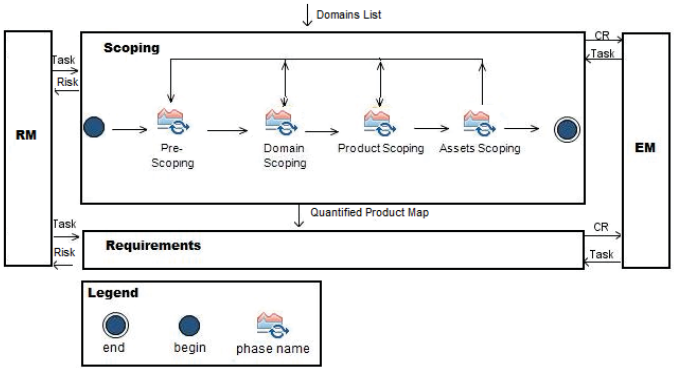


Figure 1. Relationship of the SC with RM, EM and RE.

Four steps should be performed in the *pre-scoping meeting*: *identify organizational context*, *identify operational* context, *analyze stakeholders* and *identify business goals*. For performing these steps, the *pre-scoping meeting* receives as inputs the *stakeholders' information*, which are related to information such as "what" are the expectations of the stakeholders for the SPL, "what is the profile of the project team" and in "which" operational and organizational context the project will be inserted; and a *domains list*, an artifact in which are defined the domains that will be analyzed in the product line. As output is produced, the *SPL vision*, where relevant information for the project, identified in the *pre-scoping meeting*, are documented.

a) Identify Organizational Context

It must capture context aspects of the organization for identifying how the activities performed during the identification of the scope will be influenced. Thus, the following aspects should be considered: **1.** *structure*, it is an aspect that can influence the planning of meetings and workshops. In the context of small organizations, the execution of an agile approach, i.e., focused on workshops integrated by all the projects stakeholders, makes possible the face-to-face communication, the collaboration among the stakeholders and consequently the decrease of documentation. On the other hand, in complex organizations, dates are harder to find and important experts might miss due to other obligations and sometimes it is not possible that all people meet at the same location [13], thus for mitigating these problems, we recommended the use of different ways such as workshops and individual interviews for communication and coordination; and **2.** *maturity*, it directly influences the problem understanding and consequently the agility in which the scoping process is performed. The knowledge of both, process and domain is fundamental for the best scoping result. Therefore, the maturity will determinate the need for trainings on the process and the investigation of documentation about the domains before performing the scoping process.

b) Identify Operational Context

The operational context has impact on the scoping planning, influencing the overall scoping process [13]. They drive decisions that affect all further scoping tasks. These decisions are exposed in the following aspects: **1.** *business constraint*, (e.g. time-to-market and resources) can affect the process lifecycle and the detail level of some assets. Furthermore, it influences on the choice of stakeholders, tools

and in the scope size. In projects where the time and the resources are limited, the team should commit oneself with a smaller scope; and **2.** *process*, the identification of the process used by the organization determines if there is a relationship between it and reuse practices. In cases where the organization presents reuse practices inserted in the current process, the adoption of SPL practices is easier, as well as the understanding of why specifics scoping tasks are performed.

c) Analyze Stakeholders

A stakeholder is someone who has a defined interest in the outcome of the project. In the RiPLE-SC, the choice of the stakeholders and their respective roles is performed according to the profile analysis of each stakeholder in the initial context of the product line.

A product line can start from the scratch, i.e. it can be introduced while some products are already under development, or its core assets can be reengineered from legacy systems. Thus, the initial context influences on the identification of the stakeholders and consequently in the scope definition, because it can present different information sources. In this choice is also considered the maturity of the stakeholders on the domains that will be analyzed in the SPL.

d) Identify Business Goals

In this step, stakeholders' goals are identified. RiPLE-SC considers that scoping is not only an economical activity, but also a social activity and, therefore, should be defined with the participation of each relevant stakeholder group for the SPL, making possible the identification of different views and business goals. In this sense, there are an arbitrary number of business goals for the product line development. However, according to the study on software reuse measurement based on experts performed by the RiSE Labs, the business goals most considered in organization that adopts practices of product lines are, respectively: *reduce costs*, *improve the productivity*, *improve the time-to-market*, *improve the quality*, *increase the company portfolio* and *gain new markets*.

2) Analyze Market

The systematic research and the analysis of external factors such as market segments and potential determine the success of a domain in the market. It involves the gathering of business intelligence, competitive studies and assessments, market segmentation, customer plans, and the integration of this information into a cohesive business strategy and plan [5]. Thus, the task *analyze market* aims to obtain information of the market segments in which the domains are inserted for identifying issues that can determine their success in the marketplace.

This task uses as input the *domains list* that will be analyzed in the product line and the *SPL vision*, producing as output the *business plan*. The *business plan* is a strategic plan that has information about the domains potential in the market, the market segments related, their sales channel and the factors that determine the success of these domains in the marketplace. The market expert performs this task.

B. Domain Scoping

A key question in developing a domain scoping process is: which set of factors can determine the domains and sub-domains of more potential? The answer for this question is to consider dimensions such as: maturity, volatility, market potential, reuse potential, risks, experience, coupling and existing code potential. These dimensions directly impact on the potential of a domain or sub-domain, for example, mature domains make possible easy access to documentation and the existence of reusable code is more probable. In this sense, the domain scoping phase of the RiPLE-SC combines the fundamental aspects for a success domain analysis.

The domain scoping of RiPLE-SC aims to analyze and discuss the domains and sub-domains among the projects members, considering points which aid in the definition of the most relevant ones. Moreover, this phase proposes the knowledge dissemination of the domain and sub-domain among the team, decreasing the need for documentation.

This phase is composed by the task *analyze domains*, which has as inputs: an already prepared list of domains, the *SPL vision*, the *stakeholders' knowledge*, relevant information of each stakeholder about the dimensions discussed, and the *business plan*. It produces as output: the *domains and sub-domains list*, which is composed by the domains and sub-domains of more potential for the SPL

1) Analyze Domains

In the RiPLE-SC, the task *analyze domains* as well as the other tasks of the process is iterative and incremental. Therefore, in each iteration, new information can be searched and thus evidenced the need of inclusion or exclusion of some domain. For example, it is common that in the definition of the products, new domains are identified and thus an analysis of these domains should be performed. It is possible also that with the prioritization of the features, a domain is considered dispensable and excluded.

The domain analysis of the RiPLE-SC uses the workshops culture in its execution, i.e., all steps of the task *analyze domains* are performed in the workshops format, mediated by the scoping expert. The workshops are a fundamental resource from agile methodologies for maintaining the integration in a team. Moreover, it allows that the stakeholders of the project discuss several aspects expressing their viewpoints.

The domain analysis workshops proposed in the RiPLE-SC makes possible the communication face-to-face and the collaborative work between business and technical people. It has as goal to identify the domains and sub-domains more relevant for the product line. In addition, this analysis requires a clear understanding of the stakeholders that integrate the workshops of the domains, because the agility can be negatively influenced when the stakeholders do not have experience with them. Thus, before the beginning of the domain analysis, the participants should explore information on the domains that will be analyzed. This information can be obtained from: project plans, user manuals, modeling, data dictionary, existing applications and knowledge from domain experts.

Four steps should be performed in the task *analyze domains*: *review domains*, *identify sub-domains*, *analyze sub-domains* and *prioritize domains* and *sub-domains*. These steps are discussed in the next sections.

a) Review Domains

In this step, the objective is to discuss general characteristics of the domains, making possible to align the knowledge of the stakeholders regarding them. The review of

each domain is performed in pre-determined timeboxes. The timeboxes are time interval which cannot be exceeded. With them is possible to limit the time of the analysis and maintain the focus on the workshop. In the review, the stakeholders should express their understanding on the domains, enabling them to identify new domains, exclude existent ones or maintain the initial set of domains.

b) Identify Sub-Domains

A domain can be composed by a big number of systems. Furthermore, the identification of "what" is *in* and *out* in a given domain depends on the vision of specialists and stakeholders in general, because each one has their particular interests. Thus, the division of the domains in technical sub-areas (sub-domains) makes possible a deepened analysis of each individual area of the domain, facilitating the choice of the areas where the reuse is adequate and where the economical potential is most suitable.

The sub-domains definition is performed in brainstorms sessions. These brainstorms provide an opportunity for the stakeholders, together, gather information about their needs and expectations regarding to key sub-domains that are of particular concern to them. In this step, is important also that the team is aligned with the customer' needs and the product line as whole. Thus, the knowledge of domain experts and market experts is determinant to drive the team in the choice of which sub-domains are more relevant for SPL and their customers.

c) Analyze Sub-Domains

In this step, each sub-domain is individually discussed by the stakeholders concerning the relevant dimensions previously defined for the analysis. Thus, considering the different visions of the stakeholders, a conclusion about each dimension evaluated in each sub-domain is performed. The following dimensions are considered in the analysis: **1.** *experience*, it indicates the level of knowledge that the participants of the workshop have on the sub-domain; **2.** *risks*, they are identified and analyzed to determine the negative impact on the sub-domain. In the analysis, the risks are prioritized according to the perceptions of the team about their severity; **3.** *volatility*, it determines if the sub-domain changes with the time. **4.** *maturity*, it determines if the sub-domain is stable; **5.** *code potential*, the code existence facilitates the understanding of the sub-domain and aids in the development; **6.** *market potential*, it identifies which sub-domains can obtain greater economical return in the market segments; **7.** *reuse potential*, it determines the possibility of the sub-domains of composing a generic reuse infrastructure; and **8.** *coupling*, it identifies if the sub-domains can be dealt in an independent way. The dimensions considered in the evaluation of the sub-domains were defined with base on relevant aspects identified in [21].

d) Prioritize Sub-Domains

The prioritization is performed based on the results obtained in the sub-domains analysis, the business goals and considerations made by the customers in the task of pre-scoping meeting.

This step is performed by different stakeholders and is possible that the different viewpoints result in conflicts. Thus, it is recommendable an initial search for common interests and the negotiation of the divergent ones. In this case, discussion sessions moderated by the scoping expert can be made, enabling the alignment of the interests for all the team. With these sessions is constructed the final list of domains and sub-domains more relevant for the product line.

C. Product Scoping

In today's competitive business environment, it is extremely important to offer for customers exactly the products that they want. SPL has the potential to enable companies to offer a large variety of products while still being able to manage the complexity caused by this increased number of products. In this sense, the product scoping phase aims to define the product portfolio that optimally satisfies customer demands, characteristic of highest priority in agile process, where the customers' role is critical [22] and at the same time restricts the number of products offered.

In this phase, the products and features associated with them are identified and evaluated with respect to their potential for introduction in the product line. In this identification and evaluation, the market analysis as well as the real needs of the customers are considered.

In order to define the product scoping, five tasks should be performed: *construct user stories*, identify features, *features review meeting, identify products*, *construct product map* and *validate product map*.

1) Construct User Stories

In a product line started from scratch or in its evolution, the utilization of user stories as basis for identification of customers' needs and consequently features of product is very important. It is well known the fact that customers have difficulties of expressing their expectations before using the final product [23]. Thus, user stories should be used in these cases, because in our vision this is the most natural way for customers express their needs. However, in cases where is necessary to construct a product line of pre-existents products, i.e. in cases of reengineering, the construction of user stories is unusual, and the features are identified from the analysis of documents associated to products or through the code.

RiPLE-SC defines user stories as brief descriptions on how customers will use the system [9]. The stories present short names, business value and are written in a short way or graphical description.

Each story can be written in index cards because they are relatively small, easy to move and order [9]. If there is the need to report the progress to others parts of the organization in a traditional format, electronic media can be used.

2) Identify Features

The goal of this task is to determine the features which will be present in the further products of the SPL. It is performed across a workshop whose collaborative participation of the team is essential. This workshop should be moderated by the scoping expert, and all participants are expected to be fully engaged. During the workshop is possible to identify and discuss several issues with base on different perspectives. In this sense, different views of the stakeholders will be confronted and analyzed enabling a better scoping result.

For identifying the features, RiPLE-SC uses as base: **1.** user stories, in this case an evaluation of the user stories by representative stakeholders is essential to determine if the stories are feasible and complete, and thus to identify potential

features for the product line and create competitive products; and/or **2.** the abstraction of the previous knowledge obtained from e.g. books, user manuals, design documents and code.

In the RiPLE-SC process, functional and non-functional features are considered. The functional features are related to aspects as "what" the system has to do. The non-functional features are associated with quality attributes, which the products should address. These attributes will serve as architectural drivers for the product line and present impact on the product line architecture. Therefore, the identification of non-functional features is of extreme relevance for the architecture of the product line.

3) Feature Review Meeting

The feature review meeting is related to feature evaluation and negotiation between the organization (domain expert and marketer) and customers. Domain experts and market experts are indicated for this task because they have high knowledge of the domains, and also know potential market segments and their needs. In this context, it is possible to obtain the customer's feedback enabling they to evaluate the work performed and define if the releases are aligned to their real needs. With the review meeting, new features can be included, excluded or reprioritized in the list.

4) Identify Products

The task *identify products* has as objective to find appropriated products for the product line. This task receives as inputs the market experts' knowledge, the SPL vision, the features list and the business plan. The output of this task is a list that presents the products of more potential for the SPL.

The choice of products for a product line is critical for the organizations, because the market is competitive and requires products diversity, factor that can have negative consequences, such as: increase of costs, complexity and time-to-market, causing decrease of benefits for the product line [12]. Thus, in this choice it is highly important to consider a set of products aligned with the goals of the organization and needs of a specific group of customers or market segment.

5) Construct Product Map

After the identification of the products and their features, these are organized in a product map. In this map, columns and rows are used to represent products and features, respectively. Moreover, in this map, each column is composed of two other columns: the first indicates if the feature is a possible further feature of the product; the second indicates if the feature is required in the product. Thus, it is possible to determine which features will integrate or not each product. In addition, in the product map, each feature is related to a scope as follows: *mandatory*, features that are required by all members of the product line; *optional*, features that are part of some products, i.e., can or cannot be selected by the SPL products; or *out of scoping*, features that are part of only one product.

6) Validate Product Map

The product map validation is performed in meetings with the participation of customers, domain experts and market experts. These meetings have as objectives: **1.** identify if any feature was forgotten or allocated improperly in some product; and **2.** verify the scope defined for the features.

With this task, the product map is consolidated enabling the choice of the assets that will integrate the reuse infrastructure for the SPL.

D. Assets Scoping

The goal of the assets scoping is to determine the appropriate features that should be built for reuse. Thus, the assets scoping establishes the reusability of features relevant for the development of the reference architecture. This phase is based on a quantitative analysis of the benefits of making the feature reusable. In this sense, our process determines the definition of metrics for measuring the benefits that specific feature has for the product line. It aims to optimize the product line according to specific benefits.

It is well known that projects have different visions from different stakeholders and that benefits can differ according to the business goals of them. In this context, we consider Goal Question Metric (GQM) [4] as a way for deriving metrics based on different business goals expected by the stakeholders. The choice of performing assets scoping focused on GQM was influenced by the study performed in [1], where ten experts were surveyed regard some factors related to software reuse measurement and was identified that GQM is the most utilized method for reuse measurement.

This phase is composed by the tasks: *create metrics, apply metrics* and prioritize product map, as next described.

1) Create Metrics

The metrics creation involves two well-defined steps: *refine and operationalizate business goals*, *develop benefit and characterization metrics*.

a) Refine and Operationalizate Business Goals

This task initiates with the step *refine and operationalize business goals*. Initially, the goals, prior produced in the pre-scoping phase, are analyzed and refined. The refinement is made because in the phase of pre-scoping, the goals are identified generically. Thus, the goals are refined according to their relevance for the customers, organization and domains.

After the refinement, it is performed the operationalization of the goals according to four distinct levels: goal, question, characterization metrics and benefits metrics. The operationalization flow starts with the goal level, where the optimized description of the goal is performed following the schema that has the form of <Purpose><Issue><Object> <Context>, such as ***Minimize*** *the* ***effort*** *needed for* ***the development of new applications*** *from the viewpoint of* ***software engineers*** *in* ***the company***.

In the question level, additional aspects are elicited and the goal is defined more preciously. After the aspects have been elicited, the goal is expressed in characterization metrics that will be used in the benefits metrics. The characterization and benefit metrics are next detailed.

b) Develop Benefit and Characterization Metrics

The characterization metrics are used for identifying the need of a specific feature to integrate a product in the product line. The benefit metrics describe the potential of introducing a specific feature into reuse infrastructure considering a defined business goal.

2) Apply Metrics

In this task, the benefit and characterization metrics are applied for the product map in two steps: *apply characterization metrics* and *apply benefit metrics*.

a) Apply Characterization Metrics

In general, this task is performed by the stakeholders who provided the different goals, e.g., the developer can be useful to estimate effort, while the architect can help in the definition of which are the impacts of some features in the reuse infrastructure.

b) Apply Benefit Metrics

After the application of the characterization metrics, the benefits of the assets for the product line are evaluated. In this step, the metrics are applied based on the goals previously considered.

3) Prioritize Product Map

After assigning the values for the benefit metrics, the task prioritize product map is performed. In this task, the features with more potential for the product line are selected.

The product map prioritization is performed in a meeting which involves the customer, manager, marketer and domain expert, where considerations and negotiations are performed.

As RiPLE-SC is intended to satisfy the customer's real needs, their feedback is indispensable for definition of the SPL scope. Thus, the base to perform the prioritization of the product map is the customers' need.

IV. Related Work

In [21], we performed a systematic review on scoping approaches for SPL where relevant approaches were analyzed and clustered. Therefore, in this section, we will make a brief discussion about the main related work.

Helferich et al. [8] focus on Product Portfolio Planning (PPP) based on Quality Function Deployment. Their work demonstrates how QFD-PPP can be used to identify different customer groups and their needs, derive systematically product portfolio (i.e. members of a product line) and derive common and variable product functions.

The scoping customization is inserted in [13]. This work highlights that scoping should be customized for concrete situation and activities, because representative stakeholders, artifacts and execution time for the tasks can change according to several factors, such as: resources, organizations factors, etc.

In the context of agile product lines, one approach proposes agile scoping [24], where the goal is to define a product map, i.e. a matrix that relates products and features using active participation and collaboration among the project stakeholders. During the approach, tasks of Collaboration Engineering and practices of AM are used in parallel.

In our work, we defined a systematic agile scoping process for SPL making possible the incremental scope definition and agility, favoring the up-front costs and effort decrease.

V. Conclusions

During the last decade, several efforts were conducted to achieve effective ways of dealing with the software industry competitive needs. In this context, Agile Product Line Engineering (APLE) is a new research area that aims to join the benefits of product lines and agile methods and which was motivated by the need to rapidly deliver high quality software that meets the changing needs of stakeholders.

In this paper, we explored this combination in the scoping phase. Scoping is the first activity to start a software product line and is considered key for its success. It aims to decide which features are *in* and *out* in the product line. We believe that our work can be used as important base to develop the next generation of scoping approaches covering agile issues. As future work, we intend to evaluate our process in an industrial case study in the medical domain.

Acknowledgement

This work was partially supported by the National Institute of Science and Technology for Software Engineering (INES), funded by CNPq and FACEPE, grants 573964/2008-4 and APQ-1037-1.03/08, CNPq grants 305968/2010-6, 559997/20 10-8, 474766/2010-1 and FAPESB grant 783/2010.

References

[1] Almeida, E. S. *Software reuse measurement – what the experts think about it*. Workshop to Introducing Reuse in the Enterprises, Invited Talk, Brazil, 2009.

[2] America, P., Thiel, S., Ferber, S., and Mergel, M. *Introduction to Domain Analysis*. Technical Report, ESAPS project, 2001.

[3] Boehm, B. Get ready for agile methods, with care. *IEEE Computer*, 2002.

[4] Briand, L., Differding, C., and Rombach, D. Practical guidelines for measurement-based process improvement. *Software Process Improvement and Practice Journal,* 2(4), 1996, 253-280.

[5] Clements, P., and Northrop, L. *Software Product Lines: Practices and Patterns*. Addison Wesley, 2001.

[6] Erickson, J., Lyytinen, K., and Siau, K. Agile modeling, agile software development, and extreme programming: the state of research. *Journal of Database Management,* 16(4), 2005, 88–100.

[7] Hanssen, G. K., and Fægri, T. E. Process fusion: an industrial case study on agile software product line engineering. *Journal System and Software*, 2008.

[8] Helferich, A., Herzwurm, G., and Schockert, S. QFD-PPP: product line portfolio planning using quality function deployment. In *Proceedings of the SPLC*, France, 2005, 162-173.

[9] Hunt, J. *Agile Software Construction*, 2006.

[10] Ghanam, Y., and Maurer, F. An iterative model for agile product line engineering. In *Proceedings of the SPLC*, Ireland, 2008, 377-384.

[11] Ghanam, Y., Maurer, F., Abrahamsson, P., and Cooper, K. A report on the XP workshop on agile product line engineering. *ACM SIGSOFT,* 2009.

[12] Jiao, J., Zangh, Y., and Wang, Y. A heuristic algorithm for product portfolio planning. *Computers & Operations Research*, 34(6), 2005.

[13] John, I., Knodel, J., Lehner, T., and Muthig, D. A practical guide to product line scoping. In *Proceedings of the SPLC*, U.S., 2006, 3-12.

[14] John, I., and Eisenbarth, M. A decade of scoping – a survey. In *Proceedings of the SPLC*, U.S., 2009, 31-40.

[15] John, I. Using documentation for product line scoping. *IEEE Software*, 2010.

[16] McGregor, J. D. Agile software product lines, deconstructed. *Journal of Object Technology,* 7(8), November, 2008, 7-19.

[17] McGregor, J. D. Agile software product lines - a working session. In *Proceedings of the SPLC*, Ireland, 2008, 364.

[18] Mohan, K., Ramesh, B., and Sugumaran, V. Integrating software product line engineering and agile development. *IEEE Software*, 2010.

[19] Moraes, M. B. S., Almeida, E. S., and Meira, S. R. L. A systematic review on software product lines *scoping*. In *VI Experimental Software Engineering Latin American Workshop (ESELAW)*, São Carlos, Brazil, 2009, 63-72.

[20] Nerur, S., Mahapatra, R., and Mangalaraj, G. Challenges of migrating to agile methodologies. *Communications of the ACM*, 48 (5), 2005, 72-78.

[21] Nuseibeh, B., and Easterbrook, S. Requirements engineering: a roadmap, In *Proceedings of the FOSE*, Limerick, Ireland, 2000, 35-46.

[22] Noor, M. A., Rabiser, R., and Grünbacher, P. Agile product line planning: a collaborative approach and a case study. *Journal of Systems and Software*, 2007, 81(6), 868-882.

[23] Schmid, K. A comprehensive product line scoping approach and its validation, In Proceedings of the *SPLC*, USA, 2002.

An Approach for Identifying and Implementing Aspectual Features in Software Product Lines

Mohamed A. Zaatar
College of Computing & IT
Arab Academy for Science & Tech.
Cairo, Egypt
mzaatar@acm.org

Haitham S. Hamza
Dept. of IT
Cairo University
Giza, Egypt
hshamza@acm.org

Abd El Fatah Hegazy
College of Computing & IT
Arab Academy for Science & Tech.
Cairo, Egypt
abdheg@yahoo.com

Abstract

Software Product Lines (SPL) exploits reuse by identifying, modeling, and systemically reusing software features to develop different but related software systems. Successful reuse of a product line depends greatly on the modularity of the features that characterize the product line. Traditionally, features in SPL are grouped along the dimension of commonality *and* variability. *However, this single-dimension grouping overlooks the crosscutting nature of some features in the system, which negatively impacts the reusability and modularity of the product line architecture. In this paper we address this particular problem by investigating the concept of* **Aspectual Feature** *(AF) as another grouping dimension that can be used in SPLs. To this end, the paper proposes the Aspectual Product Line Engineering (APPLE) approach for identifying, modeling, and implementing AFs to enhance the reuse of SPLs. A tool support for implementing the APPLE approach is also presented and demonstrated through a case study.*

1 Introduction

Software product line (SPL) engineering has emerged as an effective and practical technology to exploit systematic reuse in developing software applications. An SPL can be defined as "*a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a particular way*" [1]. A core activity in engineering product lines is the *Domain Engineering* (DE) process used to engineer reusable assets necessary to develop a family of applications within a defined domain [2] [3].

Domain Analysis (DA) [4] is a key activity in DE that focuses on identifying and modeling common and variable features for a set of software systems, and exploiting these features into reusable artifacts that can be used for engineering various related products and applications. This activity is known as *Commonalities and Variabilities* (C&V) analysis. Identifying, modeling, and encoding C&V features play a key role in facilitating the reuse of these features. Dependency, coupling, and cross-cuttings among features may limit their reuse, or at least make reuse more difficult and error prone. Conventional SPL development approaches group features along the dimension of *commonality* and *variability* without considering the interrelationships among these features.

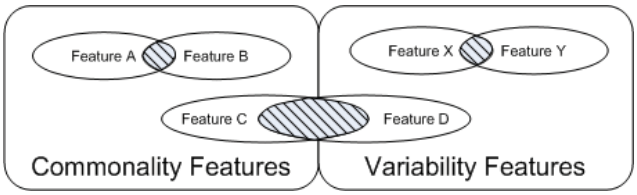


Figure 1. Aspectual Feature Existence

In practice, features do not live in isolation. They typically cross-cut and interact with each other to implement the key functionalities of the system. Figure 1 shows some examples of features that may cross-cut each other within the commonality (A and B) or variability layers (X and Y), or even across the two layers (C and D).

Dealing with cross-cutting features in SPL is not new. In the literature, two research directions were investigated in this context. The first direction focuses on identifying crosscutting features within the variability features (e.g., Features X and Y in Figure 1. [6] [7] [8] [9]. However, this work ignores other types of crosscutting features within the commonality or across the commonality and variability layers. The second research direction focuses on developing

and end-to-end development process to handle crosscutting features in the SPL life-cycle. Example of this direction is the approach reported in [12]. However, this approach is complex as it relies on data-mining and natural language processing to identify crosscutting features.

We argue that identifying cross-cut features is circuital in developing truly reusable software product lines, and there should be a practical approach for identifying various types of such features. Accordingly, in this paper, we investigate the concept of ***Aspectual Feature*** (AF) as another grouping dimension that can be used in conjunction with the traditional *commonality* and *variability* dimension used in most SPL techniques. Recent work reported in [6], [7], [8], and [9] focuses on To this end, we propose a new approach, namely, the ***Aspectual Product Line Engineering (APPLE)*** approach based on the theory of Formal Concept Analysis (FCA) for identifying, modeling, and implementing AFs to enhance the reuse of SPLs. In addition, we present a CASE tool for implementing the APPLE approach and demonstrate its use via a case study.

The rest of the paper is organized as follows. Section 2 presents the proposed APPLE approach and describes its main activities. In Section 3, a case study for applying the APPLE approach is presented. Related work is discussed in Section 4. Conclusions are presented in Section 5.

2 The Proposed Approach

In this section we present the proposed *Aspectual Product Line Engineering* (APPLE) approach, and provide an overview about its main activities and implementation. A brief overview about the Formal Concept Analysis (FCA) theory is first presented.

2.1 Formal Concept Analysis (FCA)

Formal concept analysis (FCA) provides means to identify meaningful groupings of objects that share common attributes as well as provides a theoretical model to analyze hierarchies of these groupings [18]. A formal context $C = (A, O, R)$ in FCA consists of a set of objects O, a set of attributes A, and a binary relation $R \subseteq O \times A$ between objects and attributes, indicating which attributes are possessed by each object. The formal context can be used to generate a set of concepts where a concept C is defined as a pair of sets (X, Y) such that:

$$X = \{o \in O | \forall a \in Y : (o, a) \in R\} \quad (1)$$

$$Y = \{a \in A | \forall o \in X : (o, a) \in R\} \quad (2)$$

Where X is said to be the extent of the concept C and Y is said to be its intent.

The set of all concepts of a formal context and the partial ordering can be represented graphically using a concept lattice. A concept lattice consists of nodes that represent the concepts and edges connecting these nodes. The concept lattice is the basis for further data analysis.

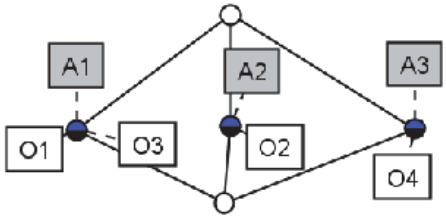


Figure 2. Lattice for the Context of Table 1.

Table 1 shows an example of a simple context. The lattice of this context is illustrated in Figure 2.

Table 1. Sample Formal Context.

	A1	A2	A3
O1	X		
O2		X	
O3	X		
O4			X

2.2 The APPLE Approach

APPLE aims at providing a practical approach for detecting and propagating aspects from analysis to implementation without imposing heavy-weight analysis techniques those are not in the mainstream development process. As we discuss below, in the proposed approach, typical SPL activities such as Use Case analysis and Feature-Oriented Domain Analysis (FODA) are employed. The use of FCA helps to achieve this goal as it is a semi-automated technique that does not require much knowledge from typical analysts and developers.

In the following we present the various steps of the APPLE approach (Figure 3). A summary of the various activities, their input, and outputs are summarized in Table 2.

Step 1: *Domain Analysis:* As with typical SPL development approaches, the first step is to perform functional and domain analysis in order to identify the required functionalities and the common and variable features in the systems under consideration. In this step, the following two parallel activities are performed:

1.1 *UML Domain Analysis:* This activity aims at analyzing the functional and non-functional requirements of the systems within the specified domain. The output of this activity is a set of Use Cases (UCs) describes the requirements of the systems.

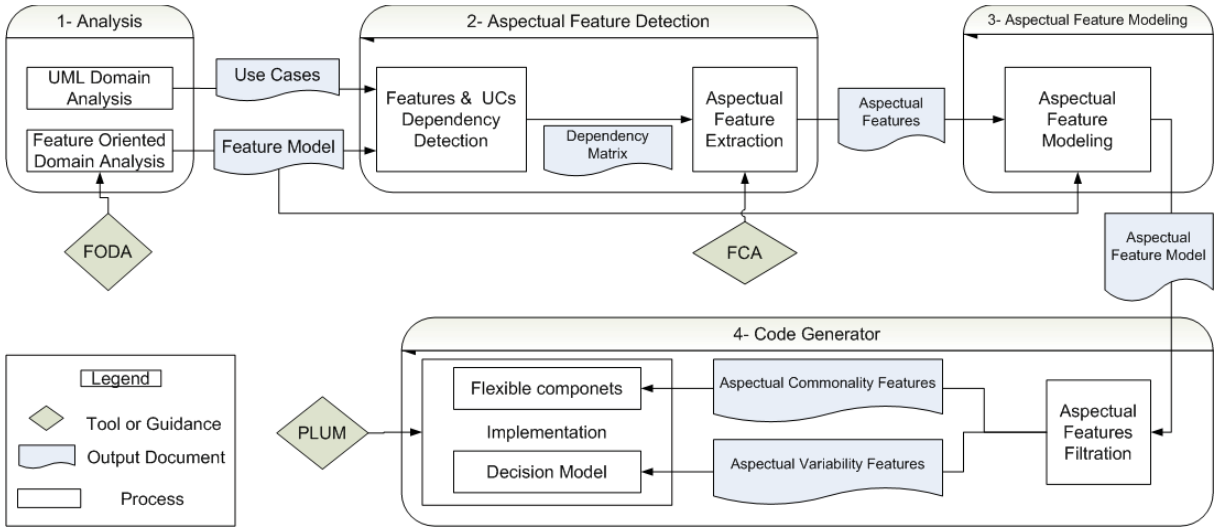


Figure 3. Main Activities In The Proposed APPLE Approach.

1.2 *Feature Oriented Domain Analysis:* In this activity, common and variable features among the systems are identified using the FODA method [5]. Identified features are modeled using conventional feature models (FM) [5].

Step 2: *Aspectual Features Detection:* This step focuses on identifying aspectual features (AFs) based on the UCs and FM developed in Step 1 above. To do so, the following two activities are performed:

2.1 *Features and UCs Dependencies Detection:* In this activity, the relationship between the identified UCs and features are carefully analyzed in order to identify their dependencies. We follow an approach similar to that reported in [8][9]. The result of this activity is a dependency matrix where the rows and columns represent the features and the UCs, respectively.

2.2 *Aspectual Feature Extraction:* The matrix obtained from the previous activity is processed using FCA in order to identify actual Aspectual Features in the systems. A lattice is constructed and its structure is analyzed (via a tool) in order to identify crosscutting and non-crosscutting features.

Step 3: *Aspectual Feature Modeling:* In this step, the conventional feature model constructed in Step 1 is now revised and updated based on the results of Step 2.2 above. The result is a new feature model with a set of features classified along two dimensions. The first dimension differentiates between common and variable features (from the original FM), whereas the second dimension differentiates between crosscutting and non-crosscutting features (based on activity 2.2). It is worth noting that, at this point crosscutting and non-crosscutting features include both common and variable features. This particular point set the APPLE approach from most existing research that focuses only on identifying crosscutting features within variabilities only. The result of this step is an updated FM named Aspectual Feature Model (AFM).

Step 4: *Code Generation:* In this step, the actual implementation of the SPL is performed. In our approach, we adopt the PLUM (Product Line Unified Modeler) as a tool for generating the code for the product line under development [17]. In PLUM, common features are encoded in what is called Flexible Components (FCs), whereas variable features are presented in the so-called Decision Model (DM). DM is used to set the values of the variable features according to the particular product in the SPL family. Accordingly, we handle the common and variable aspectual features differently. That is, common AFs are handled in the FCs, whereas variable AFs are handled in the DM. The following are the two activities used in APPLE to implement AFs:

4.1 *Aspectual Features Filtration:* In this activity, we isolate common and variable aspectual features in order to prepare these features for appropriate implementation.

4.2 *PLUM Implementation*: In this activity the actual code is developed and generated as appropriate. In particular, common AFs are implemented in the FCs using appropriate Aspect-Oriented Pro-

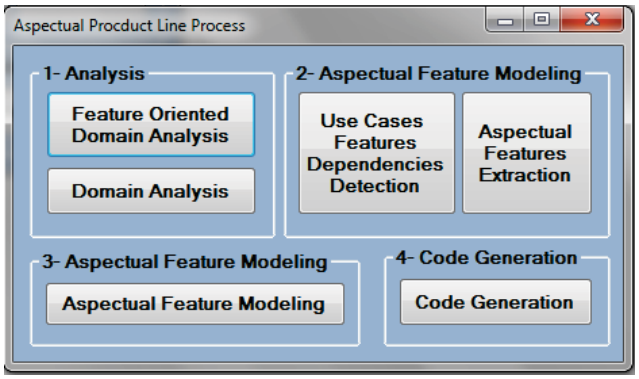


Figure 4. Tool Interface for APPLE.

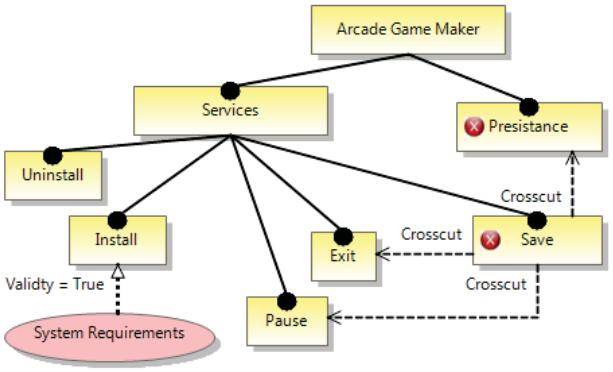


Figure 5. Aspectual Feature Model

gramming (AOP) languages (e.g., AspectJ). Variable AFs are handled in the DM using a sub-class of the Decision Class in PLUM Decision Meta Meta Model.

A CASE tool is developed in order to semi-automated most of the activities in the proposed APPLE approach. The tool supports the four steps shown in Figure 3. The GUI of the developed tool is illustrated in Figure 4. In the lattice generation it extracts and identifies the aspectual features by exploring the scattering and tangling dependencies. Each feature have to be scattered and at least one of the corresponding use cases are tangled to be an aspectual feature. A customization done on the FCA generation to show the aspectual features by analysis of the dependencies and draw hashing on the aspectual features. The tool includes add-in for UC modeling, and Feature Modeling.

The conventional feature model is extended to include the aspectual feature types. Figure 5 shows a sample aspectual feature model with two aspectual features, namely `Save` and `Persistence`.

Aspectual features are marked with an X in the left-hand side as shown in the figure. Each aspectual feature crosscuts

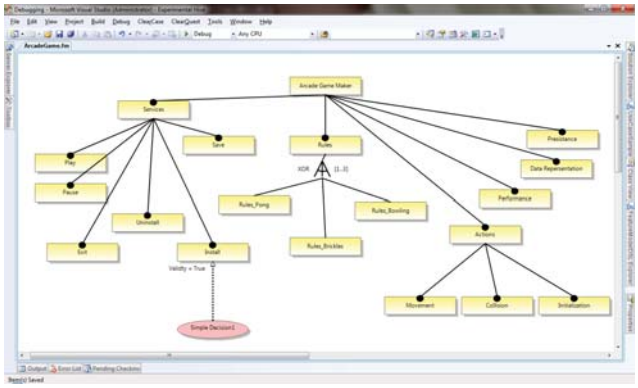


Figure 6. Arcade Game Maker Feature Model.

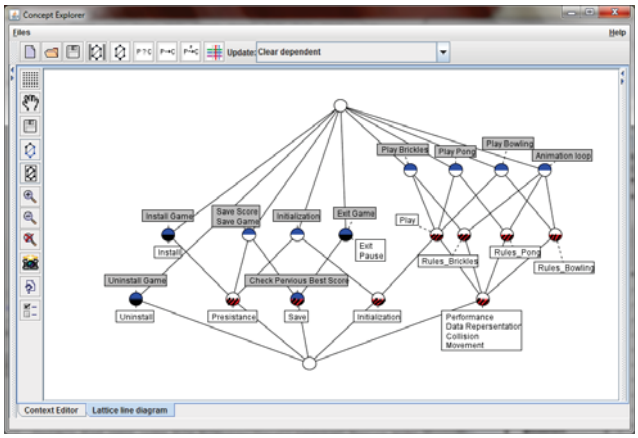


Figure 7. Arcade Game Maker FCA Lattice.

one or more features. The modified aspectual model illustrates this crosscutting nature by dotted line. For example, as shown in Figure 5, the aspectual feature `Save` crosscuts three other features: `Pause`, `Exit`, and `Persistence`.

3 Case Study: The Arcade Game Maker

To demonstrate the APPLE approach, we applied it to the Arcade Game Maker Product Line [8][16]. Arcade Game Maker is a SPL that aims at generating three related, but different games. Each game is a Single-player Game and has some Rules. A Scoring point is obtained by hitting some hurdles. The installer component of the software can install and uninstall the game. The User should be able to interact with the game. Some interactions with the game is to play, save Game and exit. Saving Score is one of the features available in the game. Examples for some functional requirements in this system include: *Play, Pause, install, and Uninstall*. Examples for some non-functional requirements include: *Performance and Persistence*. Most of the func-

Table 2. Summary of Main Activities in the APPLE Approach.

Step	Activities	Input	Output
1- Domain Analysis	UML Domain Analysis	Requirements	Use Cases (UCs)
	Feature Analysis (FODA)	Domain Requirements	Feature Model (FM)
2- Aspectual Features Detection	Features and UCs Dependencies Detection	UCs and FM	FCA Lattice
	Aspectual Feature Extraction	FCA Lattice	Aspectual Features (AFs)
3- Aspectual Feature Modeling	AF Modeling	AFs	Aspectual Feature Model (AFM)
4- Code Generation	Aspectual Features Filtration	AFM	Aspectual Variability and Commonality Features
	PLUM Implementation	Filtered Features	PLUM Code

tional requirements are mandatory and some are related to the particular game that will be generated.

First, the UC and the feature models are constructed using conventional UC analysis and FODA (Figure 6). Next, the dependencies between the identified UCs and features are captured using the technique reported in [8][9]. The results of these analysis steps are used to generate the FCA lattice given in Figure 7. Based on analysis of the generated lattice, an updated Aspectual Feature Model is generated. The identified AFs are filtered and implemented in the FCs and DM of the PLUM tool as discussed above.

4 Related Work

The use of FCA in SPLs is not new. In [13], Niu and Easterbrook reported the use of FCA to examine and refine the functional and quality requirements in software product lines based on the concept of clustering. In [15], Tonella and Ceccato proposed the use of FCA to mine the aspects from the execution traces of the software. These research efforts exploit FCA in SPL at the code level only. In our work; however, we exploit FCA early in the analysis phases.

In [10], Salinas and Suesaowaluk have developed an aspect-oriented approach for SPLs in the context of web applications. They focused on identifying and managing the variabilities from the analysis to the coding phases. In [14], Silva *et al.* , proposed an approach for the mapping between the SPL-AOV graph and the feature model to present a graph that contains both the features and the requirements of the system.

In [8], Conejero and Hernández proposed an approach based on their previous work reported in [9] to identify the effects of crosscutting features. The approach uses matrix operations to identify the relationships between a set of UCs and features by identifying the so-called scattering and tangling matrices. However, they do not show how crosscutting features can be identified from these matrices. The APPLE approach presented in this paper, does not only identify crosscutting features, but it also discovers the relationships between UCs and features without complex manual matrices operations as those used in [8].

Loughran *et al.* proposed in [12] the NAPLES (Natural language Aspect-based Product Line Engineering of Systems) approach. NAPLES aims at providing an end-to-end SPL development approach from requirements to code generation. Similar to the APPLE, NAPLES attempts to analyze aspects in addition to conventional commonality and variability analysis for product lines. In NAPLES, natural language processing and data mining (EA-Miner) techniques are applied to the requirements documents and the textual assets related to the domain of the systems under development in order to identify early aspects, viewpoints, and the commonalities and variabilities. We argue that NAPLES can be complex for use in mainstream SPL development life-cycles. APPLE, on the other hand, makes use of Use Case and FODA, which are more natural to the typical SPL developers.

Table 3 gives a comparison between the proposed APPLE approach and the matrix-based and NAPLES approaches.

5 Conclusions

In this paper, the Aspectual Product Line Engineering (APPLE) approach is proposed to exploit the concept of aspectual features (AFs) for developing modular software product line systems. The key objective of APPLE is to identify, model, and implement the crosscutting features by analyzing the relationships between the requirements and features of the system early in the analysis phase. Based on this analysis, AFs are identified and handled efficiently to facilitate the reuse of the SPL. The APPLE approach is

Table 3. Comparison Between Various SPL Approaches.

Property	Matrix-based Technique [8]	NAPLES [12]	APPLE (This Paper)
Supported Development Phase(s)	Analysis	Analysis Code Generation	Analysis Design and Architecture Code Generation
Aspectual Feature Modeling	No	No	Yes
C & V	No	Yes	Yes
Crosscutting Feature Detection Technique	Matrices	Data Mining	FCA

presented and its activities are described. A CASE tool to semi-automate the various activities of the APPLE approach is also presented and its use is demonstrated through the Arcade Game Maker software case study.

References

[1] P. Clements and L. Northrop. Software Product Lines: Practice and Patterns. Addison-Wesley, 2007.

[2] H. Mili et al. Reuse-based software engineering. John Wiley & Sons, Inc. 2002.

[3] G. Arango, "A brief introduction to domain analysis," 1994, ACM.

[4] J. Neighbors, "Software Construction using components," PhD Thesis, Dept. of Information and Computer Science, U. of California, Irvine, 19981.

[5] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. "Feature-Oriented Domain Analysis (FODA) Feasibility Study," *Chapter 1, pp 21-31. Carnegie Mellon University, SEI,* 1990.

[6] R. Bonifácio, P. Borba, and S. Soares, "On the Benefits of Scenario Variability as Crosscutting," *Proc. of the 2008 AOSD workshop on Early Aspects, EA '08,* pp. 1-6, Belgium, 2008.

[7] R. Bonifácio and P. Borba, "Modeling Scenario Variability as Crosscutting Mechanisms," *Proc. of the 8th ACM Int. Conf. on Aspect-oriented Software Development, AOSD '09,* pp. 125-136, Virginia, USA, 2009.

[8] J. M. Conejero and J. Hernández, "Analysis Of Crosscutting Features In Software Product Lines," *Proc. of the 13th Int. Workshop on Early Aspects, EA '08,* pp 3-10, Germany, 2008.

[9] K. van den Berg, J. M. Conejero, and J. Hernández, "Analysis of Crosscutting Across Software Development Phases Based On Traceability," *Proc. of the 2006 Int. Workshop on Early Aspects at ICSE, EA '06,* pp 43-50, China, 2006.

[10] G. H. A. Salinas and P. Suesaowaluk, "An Aspect-Oriented Product Line Framework to Support the Development of Software Product Lines of Web Applications," *Proc. of 24th South East Asia Regional Computer Conf.,* pp 13-18, Thailand, November 2007.

[11] Finkelstein, A., Sommerville, I.: The Viewpoints FAQ. BCS/IEE Software Engineering Journal 11(1), pp.24 (1996)

[12] N. Loughran, A. Sampaio, and A. Rashid, "From Requirements Documents To Feature Models For Aspect Oriented Product Line Implementation," *Workshop on MDD in Product Lines (held with MODELS 2005),* Montego, vol. 3844 of Lecture Notes in Computer Science, pp 262-271. Springer, 2006.

[13] N. Niu and S. Easterbrook, "Concept Analysis For Product Line Requirements," *Proc. of the 8th ACM Int. Conf. on Aspect-Oriented Software Development, AOSD '09,* pp. 137-148, Virginia, USA, 2009.

[14] L. Silva, T. Batista, S. Soares, and L. Santos, "On the Role of Features and Goals Models in the Aspect-Oriented Development of Software Product Line," *Information Sciences and Technologies Bulletin of the ACM Slovakia, Special Section on Early Aspects, R.Chitchyan, S. Zsachaler (Eds.),* Vol. 2, No. 1, pp. 60-65, 2010.

[15] P. Tonella and M. Ceccato, "Aspect Mining through the Formal Concept Analysis of Execution Traces," *Proc. of 11th Working Conf. on Reverse Engineering,* pp. 112-121, Delft, The Netherland, 2004.

[16] Arcade Game Maker, http://www.sei.cmu.edu/productlines/ppl/

[17] Product Line Modified Modeler (PLUM), http://www.esi.es/Projects/plum/

[18] R. Wille, "Restructuring Lattice Theory: an approach based on hierarchies of concepts, in: Ivan Rivali, ed., Ordered Sets, pp. 445-470, 1982.

Automating the Detection of Complex Semantic Conflicts between Software Requirements

An empirical study on requirements conflict analysis with semantic technology

Thomas Moser, Dietmar Winkler, Matthias Heindl and Stefan Biffl
Christian Doppler Laboratory "Software Engineering Integration for Flexible Automation Systems"
Vienna University of Technology, Vienna, Austria
{*firstname.lastname*}@tuwien.ac.at

***Abstract*—Keeping requirements consistent already at early project stages is a main success factor for software development projects. However, manual requirements conflict analysis takes significant effort and is error-prone. Requirement engineers and other project participants such as technical architects use different terminologies (due to different domain backgrounds) which makes automation of conflict analysis difficult. In this paper we propose semantic approach as foundation for automating requirements conflict analysis and introduce the automated ontology-based reporting approach OntRep. We evaluate the effectiveness of OntRep referring to (a) different types of conflicts and (b) different levels of conflict complexity in a real-world industrial case study. Major results were that OntRep had considerably higher recall and precision of conflict detection for all conflict types compared to a manual approach. Regarding complexity, the comparison with manual results shows that recall and precision of OntRep is considerably higher for complex conflicts.**

Keywords—requirements conflict detection, requirements consistency checking, ontology, case study, empirical evaluation.

I. INTRODUCTION

Modern software and systems engineering projects are complex due to (a) the high number and complexity of requirements (e.g., mutual dependent or contradicting requirements), and (b) geographically distributed project stakeholders with different backgrounds and domain terminologies. A major goal of requirements engineering is to achieve consistent requirements descriptions in order to create a common and agreed understanding on the set of requirements between all project stakeholders. Initial observations at our industry partners showed that establishing consistency of requirements takes significant effort and is error-prone when performed manually. Thus, automation approaches for requirements conflict analysis are needed that (a) increase the effectiveness and quality of analyzing requirements from different stakeholders for symptoms of inconsistency, e.g., contradicting requirements, and (b) reduce the conflict analysis effort. Unfortunately, current automation approaches for conflict analysis suffer from the following challenges and limitations:

- High level of *incompleteness* [2] of conflict detection, leaving complex requirements uncovered.
- *Focus on executable code.* Some approaches need executable code to identify requirements conflicts [2], but in early stages - where conflict analysis is important - executable code may not available.
- *Limitation to syntactical comparison.* Potential conflicts are identified by using trace dependencies between requirements that have been explicitly captured before, e.g. by information retrieval approaches like "keyword matching" [6][7]. These approaches allow syntactical comparison of requirements, but do not cover semantics, even if different terms are used in these requirements definitions.

Thus, using semantic technologies seems to be a promising approach to address these challenges: ontologies provide the means for describing the concepts of a domain and the relationships between these concepts in a way that allows automated reasoning [10]. In this paper, we propose OntRep [4], an automated ontology-based reporting approach for the analysis of complex semantic conflicts between requirements based on ontologies and reasoning mechanisms. The main criteria for the evaluation of OntRep are: correctness and completeness of identified requirements conflicts and the effort to develop a project or domain ontology. OntRep aims at lowering the effort for requirements conflict analysis, while keeping requirements consistency high. Initially, OntRep automatically categorizes requirements into a given set of categories using ontology classes modeled in Protégé and mapping the terms used in the requirements to these classes. Using this foundation, OntRep analyzes the content of the requirements and identifies conflicts between requirements. Therefore, conflict analysis is not only based on traditional keyword-matching-approaches, but also considers different terminologies used for describing semantically equivalent concepts. We empirically evaluate OntRep [4] in a real-life project at our industry partner to investigate performance and quality with respect to (a) different types of conflicts and (b) conflicts on different levels of complexity.

The remainder of the paper is organized as follows: Section 2 summarizes related work on requirements conflict analysis and natural language processing; Section 3 introduces the OntRep approach and motivates research issues. Section 4 describes the performed empirical study. Section 5 presents the results which are discussed in Section 6. Finally, Section 7 concludes and suggests further work.

II. Related Work

This section presents related work on requirements conflict analysis and natural language processing [4].

A. Requirements Conflict Analysis

"Requirements conflict with each other if they make contradicting statements about common software attributes […] Given that there may be up to n^2 conflicts among n requirements […], the number of potential conflicts, […], could be enormous, burdening the engineer with the time-intensive and error-prone task of identifying the true conflicts" [2].

The Trace Analyzer by Egyed and Grünbacher [2] analyzes the footprints of test cases to detect requirements conflicts. If two requirements execute overlapping lines of code, a potential conflict may exist. A prerequisite for the Trace Analyzer is to have executable code, which is often not available in early project phases, when conflict analysis is a major goal. Heitmeyer *et al.* [5] describe a formal analysis technique, called consistency checking, for the automated detection of syntactic errors, such as type errors, non-determinism, missing cases, and circular definitions, in requirements specifications. The approach does not find semantic conflicts.

Information retrieval approaches [6], such as the RETH approach [7] use keyword-matching techniques to identify general requirements interdependencies. These captured interdependencies can be used to identify requirements conflicts. However, these techniques do not allow identifying conflicts or other interdependencies between requirements, if they use different terms for similar concepts. Thus, these approaches are less effective in practice, because they cannot identify the full set of interdependencies between requirements. The extended Bakkus-Naur-Form (EBNF) [12] is a syntax for requirements, which is used to improve the understandability of requirements for humans and machines.

B. Natural Language Processing

Natural language processing (NLP) techniques are useful to extract structure and content of requirements given in natural language for transformation into the structure of an ontology. NLP generally refers to a range of computational techniques for analyzing and representing naturally occurring texts [1]. The core purpose of NLP techniques is to achieve human-like language processing for a range of tasks or applications [8].

Most important NLP models used in this research are part-of-speech (POS) tagging and sentence parsers [1]. POS tagging involves marking up the words in a text as corresponding to a particular part of speech, based on both its definition, as well as its context. In addition, sentence parsers transform text into a data structure, which provides insight into the grammatical structure and implied hierarchy of the input text [1]. Standford parser/tagger[1] and OpenNLP[2] are the core set of NLP tools used in this paper. Furthermore, we use WordNet, a large lexical database in English [9]. Nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms (synsets), each expressing a distinct concept. Synsets are interlinked by means of conceptual-semantic and lexical relations. WordNet is a useful building block for requirements analysis. These NLP technologies can be used for our purpose, namely to improve the effectiveness of requirements management activities like categorization, conflict analysis, and tracing.

[1] http://nlp.stanford.edu/software/lex-parser.shtml

[2] http://opennlp.sourceforge.net/

III. Analysis of Conflicts with an Ontology

Due to the limitations of requirements analysis approaches that address only links between requirements based on syntactic equality, we explore an approach based on semantic equality: OntRep [4] links similar concepts sharing the same meaning even if their syntactic representations are different. As ontologies are versatile for representing knowledge on requirements and for deriving new links between requirements, we introduce an ontology-based approach for reporting analysis results on a set of requirements. The goal of the ontology-based reporting approach OntRep is making requirements management tasks such as requirements conflict analysis more effective based on the automation of selected steps in these tasks. We focus on complex semantic, which we define as conflicts regarding a set of more than two requirements and constraints, which based on observations at our industry partners are hard to identify manually. In the following subsections, we provide an overview on the approach and motivate research issues.

A. Basis for Requirements Conflict Analysis

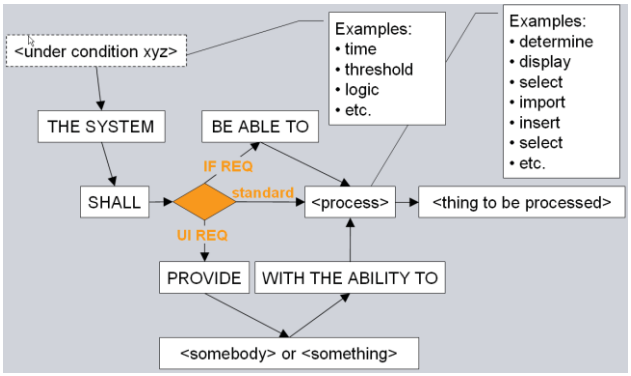


Figure 1. EBNF requirements structure (sentence level) [12].

For formally specified requirement semantics, in our case following an EBNF template (see Figure 1), semantic analysis can identify inconsistencies and conflicts using a set of assertions that should hold true for all available facts. These assertions are based on available requirements, while available facts are based on the environment and properties of the target system. Typically, requirements following this EBNF template specify under which conditions *(under condition)* a target system should provide a certain functionality *(process)* regarding a specific object *(thing to be processed)*, e.g., the system configuration, to a certain entity or role *(somebody or something)*, e.g., to an user or to the administrator. In this paper, we focus on (a) functional requirements (following an EBNF template) and (b) a set of requirements constraints regarding technical, requirement-specific or documentary issues.

We derive three types of conflicts, which can be detected with OntRep:

- *Not well-formed requirements* (conflicts between requirements and EBNF grammar).
- Conflicts between *functional requirements and technical and/or requirement constraints*.
- Conflicts between *two functional requirements*.

B. Automated Conflict Analysis

We developed a prototype tool for the OntRep approach as a plug-in to Trac[3], which can be extended by Python plug-ins.

Phase A: Linking of natural language to semantic concepts. In a first phase, natural language texts (technical constraints, requirements constraints, documentation guidelines and glossary knowledge) have to be linked to semantic concepts as preparation for further analysis and reporting [4]. The following 8 steps are used to build this project knowledge:

1) *Define project-specific concepts* in Protégé. Each concept is defined as an ontology class in Protégé. It is important to define project-relevant "semantic" concepts and not formal ones in order to enable the automated assignment, e.g., "Security". Typically, these concepts can be defined based on a project glossary that contains important project-specific terms.
2) *Provide input data* to be categorized: Constraints, guidelines and glossary knowledge are typically represented as natural language text. For our research we export these natural language texts and import them into OntRep prototype tool.
3) *Remove irrelevant stop-words*, like "and", "any", "but", which cannot be used for assignment. This step is performed automatically using a standard stop-word list[4].
4) Bring all remaining words into their *root form* (e.g., "jumping" to "jump"): this process is called "stemming" based on algorithms like "Porter Stemmer" [11].
5) Get all *synonyms and hyponyms* of the analyzed words in the requirements by using the natural language processing library "WordNet" [9]. For example, "house" is a synonym for "building", "dog" is a hyponym of "animal".
6) *Heuristic-based assignment* of each requirement to the defined concepts depending on the number of hits for (a) synonyms, (b) hyponyms, and (c) substring matches. The heuristic checks if the hits for synonym, hyponym and substring matches meet the given threshold values. If this number is equal or higher than the number of thresholds that must be met, the word will be related to that concept, otherwise not. If several concepts reach these thresholds, the requirement will be assigned to all of these concepts.
7) Save the element as an *individual of the ontology class*, if it is not already in the class. This can only be checked if one or more of the elements attributes have been declared as primary keys (uniquely identifying the element). If the element has already been saved in another class as well (which could be the case), declare that the new element is the same as the already existing one with the "*owl:sameAs*" property.
8) Manually check the *validity and correctness* of the imported fact, both regarding its assignment to the right concepts as well as regarding its meaning.

[3] http://trac.edgewall.org/

[4] http://www.textfixer.com/resources/common-english-words.txt

Phase B: Mapping of Requirements and Semantic Concepts. In the second phase, analysis and reporting approaches build on the mapping of requirements to semantic concepts. For formally specified requirement semantics, in our case following an EBNF template (see Figure 1), semantic analysis can identify inconsistencies and conflicts using a set of assertions that should hold true for all available facts. These assertions are based on the available requirements, while the available facts are based on the environment and properties of the target system (which were imported in the first phase). The following 4 steps describe the semantic requirements conflict analysis:

1) *Import of requirements*. The requirements are represented as tickets in Trac. For our research prototype we export these requirements via CSV from Trac and import them into the OntRep prototype tool.
2) *Parsing of requirements*. If the requirements are formally described using a specified grammar (e.g., EBNF), the information contained in the textual requirement descriptions can be semantically analyzed in order to identify possible inconsistencies and/or conflicts. Based on the specified grammar, certain parts of the requirements are extracted for further usage. In our case, this primarily affects the "thing to be processed" and the "obligation" ("shall" or "shall not") specified in EBNF.
3) *Linking of requirements to semantic concepts*. The "things to be processed" of the specific requirements, which were extracted in the previous step, now are linked to semantic concepts modeled in the first phase of the OntRep approach. If there is no semantic concept available for a specific "thing to be processed", a new semantic concept is created, in order to later identify conflicts between two functional requirements. Additionally, the "obligation" for each requirement is stored to enable requirement conflict detection.
4) *Requirement conflict detection*. After all requirements have been successfully parsed and linked to their related semantic concepts, every semantic concept enables checking for consistency, validity, and correctness of the related requirements using ontology-based reasoning. From the point of view of semantic analysis, we focus on two different kinds of conflicts in this paper, namely on logical inconsistencies between facts, as well as numerical discrepancies of facts. In the following, we present an example for these kinds of conflicts.

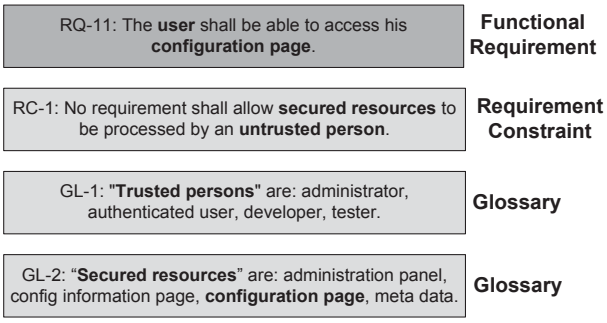


Figure 2. Logical requirement conflicts.

Figure 2 depicts an example for logical conflicts between requirements and constraints. Requirement RQ-11 links the

semantic concept "User" to the semantic concept "Configuration page". Additionally, there exists a requirement constraint specifying that no links may exist between the semantic concepts "Secured Resource" and "Untrusted Person". In the project glossary the semantic concept "Configuration Page" is identified as sub-concept of the "Secure Resource" concept, and the semantic concept "User" as sub-concept of the "Untrusted Person" concept since it is not a sub-concept of the "Trusted Person" concept which is defined as the negation of the "Untrusted Person" concept. Using these facts, the OntRep approach successfully identifies a logical conflict between RQ-11 and RC-1, relying on the facts specified in GL-1 and GL-2.

Figure 3 depicts an example for conflicts between requirements as well as for numerical discrepancies between requirements and constraints. There are three requirements which are linked to the semantic concept "Notification", RQ-12, RQ-17, and RQ-21. RQ-12 and RQ-17 are additionally linked to the semantic concept of "Messages per Second", but with a different parameter value, therefore the OntRep approach successfully identifies a conflict between these two requirements. Additionally, the semantic concept of "Notification" is linked to the semantic concept "SSL Encryption", for which a technical constraint is defined, specifying a link to the semantic concept "Messages per Second" with a parameter value of 3. While this holds true for RQ-17, RQ-12 requires at least a parameter value of 4 for the "Messages per Second" semantic concept; therefore the OntRep approach detects another conflict between RQ-12, RQ-21, and TC-1.

C. Research Issues

The underlying idea of this research is to use advanced semantic technologies, like ontologies and reasoning mechanisms, to increase the effectiveness of the analysis of complex requirements conflicts. Based on our findings in [4], the main research questions of this paper are:

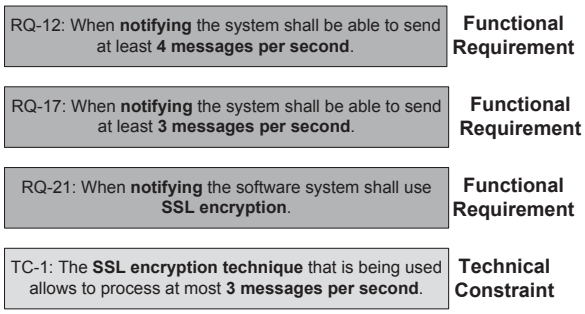


Figure 3. Numerical discrepancies of requirements and constraints.

RQ1) How effective is OntRep in finding different types of conflicts compared to manual analysis by experts? We mentioned 2 kinds of conflicts in section 3.B: numeric discrepancies and logical requirements conflicts. These conflicts may occur between different elements: (a) between requirements, (b) between a requirement and a constraint, and (c) between a requirement and a formal guideline. For these 3 types of typical conflicts we analyze the correctness and completeness of conflicts found with OntRep and compare the results to a manual approach for each conflict type.

RQ2) How effective is OntRep to identify conflicts of different degrees of complexity compared to manual analyses by experts? A conflict may consist of two or more elements, e.g., a conflict may exist between two or three requirements. We define these conflicts as "simple conflicts" because we assume they are easy to identify. On the other hand, we define conflicts consisting of more than 3 elements to be "complex conflicts". These complex conflicts might be more difficult to identify completely, especially when performing conflict analysis manually. We assume that OntRep reaches a higher completeness of complex conflict identification compared to the manual approach. For each complexity level we analyze the correctness and completeness of each identified conflict.

Based on our previous experience with tool support for quality assurance [4], we assume that OntRep can help increasing effectiveness and quality of requirements conflict analysis. Additionally, we also assume that OntRep reduces the effort for requirements conflict analysis.

IV. Study Description

This section summarizes study settings, i.e., goals, design, process, material, and participants, and identifies a set of threats to validity.

Goals. The goal of the pilot study was to analyze the effectiveness and quality of the OntRep conflict analysis approach compared to traditional manual approaches with focus on different types and complexity levels of conflicts.

Study Subject. As described in [4] the case study project is a software development project at our industry partner with the goal to design and implement a web application that serves as a platform for communication and networking between technology experts. This type of project seemed well suitable to apply and evaluate the OntRep prototype, because OntRep has been implemented as Trac plug-in and was easy to integrate with the project's tool infrastructure.

Participants were six project managers for individual conflict detection, randomly assigned to two teams for preparing an agreed team conflict list. One requirement engineering expert, i.e., one of the authors, provided control data for all tasks.

Material. We used a list of requirements (i.e., a spreadsheet) containing 23 functional requirements, 11 constraints (7 technical and 4 business constraints), and 4 formal documentation rules (documentation guidelines). To focus on individual conflict classes, i.e., conflict between requirements (CRR), conflict of a requirement with a constraint (CRC), conflict of a requirement with a formal guideline, i.e., ill-formed requirement (CRG), we seeded an overall number of 22 conflicts according to different conflict types and two complexity levels. Conflicts related to up to 3 elements are rated as simple, conflicts with more than 3 elements involved are considered as complex conflicts. In addition we used a data capturing sheet, guidelines to support the participants in identifying conflicts, and questionnaires to capture the background of participants prior to the study and feedback after afterwards.

Variables & metrics. Independent variables were the number of seeded conflicts, the defect types, and the number of

requirements per defect type. Dependent variables are the number of identified conflicts and for conflict identification. We used recall and precision as study metrics.

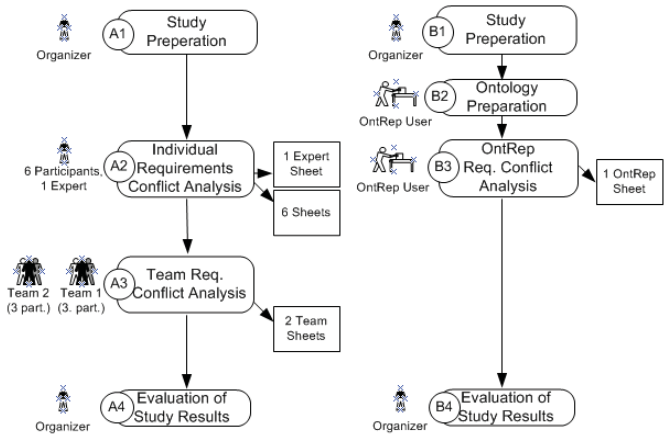


Figure 4.Evaluation Setting: Manual (l), OntRep (r) [4].

Study Design and Study Process. We applied the standard practices of empirical software engineering research according to Freimut *et al.* [3] and Wohlin *et al.* [13] to investigate the performance of OntRep in comparison to manual conflict detection. Figure 4 illustrates the relevant steps for evaluating requirements conflict analysis. This study was part of a larger overall study, reported in [4], where requirements categorization and tracing aspects were addressed in addition to conflict analysis. In context of this paper steps A2, A3, B2, and B3 are important for conflict detection. The other steps are described in [4] as part of the overall study.

A2) *Individual requirements conflict analysis*: The participants read through the task description, i.e., supporting guidelines, and were asked to (a) identify conflicts according to different conflict types and complexity levels and (b) report them within the data capturing sheet. Results of the individual conflict detection step were 6 individual results, provided by project managers and control data provided by one requirements engineering expert, i.e., one of the authors. Furthermore, we captured the individual effort of every participant.

A3) *Team requirements conflict analysis:* Afterwards, the participants harmonized their individual results within 2 randomly assigned groups of 3 team members. Again, the effort needed for this task was captured, resulting in 2 team sheets.

B2) *Ontology preparation:* A tool expert constructed one ontology class in OntRep for each category and then imported the given requirements from Trac as CSV into OntRep.

B3) *OntRep requirements conflict analysis:* Then, we provided the requirements as CSV-input to OntRep. Further, the tool expert had to model the constraints as facts and the formal guidelines as rules in the ontology. We captured the effort for this preparation. Afterwards, automated conflict identification was executed. The results were summarized in a final report.

Finally, we analyzed and evaluated the following results: (a) 6 spreadsheets for conflict analysis from each of the 6 individual participants, (b) 2 team spreadsheets, (c) 1 conflict analysis spreadsheet from a requirement engineering expert, and finally (d) 1 conflict analysis spreadsheet created with the OntRep approach. The results were evaluated with descriptive statistics in Excel and R and are described in the following section. We applied the Mann-Whitney-Test at a significance level of 95% (2-sided) for statistical testing.

Threats to Validity. Following the standard practice of empirical software engineering [13], we identified a set of internal and external threats to validity. We addressed *internal threats to validity* [3] of the study by two measures: a) intensive reviews of the study concept and materials, and b) a test run of the study conducted by a test person in order to make sure that the guidelines, explanations, and task descriptions are understandable for the participants and to estimate the required effort/time frame. Regarding *external validity*, we performed this initial case study in a professional context at a software development company. The participants had medium requirements management know-how and advanced software engineering know-how (captured by background questionnaire prior to the study). In addition, we had an expert in Requirements Engineering as control group. Nevertheless, the small number of participants might hinder the generalization of results. Therefore, we suggest replicating the study in a larger context in future work.

V. RESULTS

The following subsections describe the results of requirements conflict analysis with OntRep.

A. Conflict analysis results by conflict types

The first research question was how effective OntRep is in finding different types of conflicts compared to manual analysis? We defined 3 types of typical conflicts: Conflicts between a requirement and a constraint (CRC), e.g., when a particular requirement requires a response time that is not feasible with the chosen technology (the chosen technology constrains response times); Conflicts between a requirement and a documentation guideline (CRG), e.g. a requirement may conflict with the documentation guideline "all requirements must use the obligation word 'shall'"; Conflicts between requirements (CRR), e.g. the requirements "The system shall update the index at least 30 times per hour" and "The system shall update the index at least 20 times per hour" results in a conflict.

The individual recall and precision results for detecting CRC conflicts manually are low: approx. one third of existing conflicts has been identified and only one third of found conflicts were identified correctly. Group harmonization was quite effective for this type of conflict, because both recall and precision were improved, i.e., the conflicts found by each individual in the group have been merged in a discussion session, which results in a recall of approx. 60%; and some false positives were eliminated, which results in a precision of 50%. The expert performed better than the individuals regarding recall and precision, but the group results regarding recall were better. In this case the discussion of conflicts in the group was valuable.

Comparing the expert's CRC recall value with the CRG and CRR results shows that CRC results are clearly lower than the others. This is probably due to the fact that CRC conflicts were complex only, and thus hard to find, whereas the CRG and CRR conflict bulks consisted of either simple only or both simple and complex.

Mann-Whitney Test at significance level 95% (2-side) did not show any significant differences (p-values >0.098(-)) regarding study groups, i.e., individuals, groups, experts, and OntRep and individual conflict classes, i.e., total, CRC, CRG, and CRR. The main reason is a low number of involved data sets. Note that expert (Exp) and OntRep include one data set and we had 2 groups and an overall number of 6 individuals.

The CRG results are similar to the CRC results: rather low recall and precision values of individual conflict analysis and an improvement by group harmonization. A slight difference is that the expert performed much better than both the individuals and groups. He reached pretty high recall and precision values (more than 80%). The CRR results of the expert are similar to his CRG results, but totally different regarding individual and group results: The individual results were rather good (40% recall and 56% precision, but then worsened by group harmonization (30% recall and 24% precision).

Table 1. Overall Recall and Precision per Conflict Type.

	Total		**CRC**		**CRG**		**CRR**	
	Rec	Prec	Rec	Prec	Rec	Prec	Rec	Prec
Indiv.	32%	46%	29%	31%	30%	66%	40%	56%
Group	48%	49%	57%	49%	50%	68%	30%	24%
Expert	68%	88%	43%	100%	80%	89%	80%	80%
OntRep	100%	100%	100%	100%	100%	100%	100%	100%

Table 1 shows that OntRep has 100% values for both recall and precision for every conflict type. This means that OntRep found all of conflicts (7 CRC, 10 CRG, 5 CRR conflicts and no false positives). Although these results look like rigged for making OntRep look good, we think that the defined conflict types and the seeded conflicts are typical and realistic. Thus, the study is well-balanced and the comparison of OntRep with manual analysis results is valuable and meaningful.

B. Conflict analysis results by complexity

After analyzing the results regarding the different conflict types, we studied how effective OntRep is to identify conflicts of different degrees of complexity compared to a manual analysis. We defined two levels of complexity based on the number of elements involved in a conflict: simple and complex. The threshold number of elements to make a conflict complex is 3. We chose this threshold subjectively, because we think that analyzing 3 elements cognitively is still feasible, but gets more complicated and difficult to analyze conflicts with 4 or more involved elements. We seeded 12 simple and 10 complex conflicts (see Table 2).

In comparison to OntRep, the manual conflict analysis approach resulted in a lower completeness, no matter which type of conflict: the only conflict types for which the manual results come close to the OntRep results are the simple CRG and complex CRR conflicts; and there it is only the expert result that is similar to the OntRep result, the average results of individuals and groups are far away and false positives are introduced. They have been reduced slightly during group harmonization.

Table 2. Conflict identification per conflict type and complexity level.

	Simple Conflicts (12) Ind. / Group / Expert / OntRep (std. dev.)				**Complex** Conflicts (10) Ind. / Group / Expert / OntRep (std. dev.)			
Type **CRC**: To find: **0 simple and 7 complex** conflicts								
Fully (f)	n/a	n/a	n/a	n/a	**0.8** (2.0)	**2.0** (1.4)	0.0	**7.0**
Partially (p)	n/a	n/a	n/a	n/a	1.2 (1.6)	**2.0** (0.0)	**3.0**	0.0
Recall (f+p)	n/a	n/a	n/a	n/a	29%	57%	43%	100%
False Pos.	3.2 (3.8)	0.5 (0.7)	0.0	0.0	1.2 (1.5)	3.5 (0.7)	**0.0**	**0.0**
Precision	n/a	n/a	n/a	n/a	63%	53%	100%	100%
Not Found	0.0	0.0	0.0	0.0	5.0 (2.1)	3.0 (1.4)	4.0	**0.0**
Type **CRG**: To find**: 10 simple and 0 complex** conflicts								
Fully	2.3 (1.2)	3.0 (0.0)	**8.0**	**10.0**	n/a	n/a	n/a	n/a
Partially	0.7 (1.2)	2.0 (1.4)	0.0	0.0	n/a	n/a	n/a	n/a
Recall (f+p)	30%	50%	80%	100%	n/a	n/a	n/a	n/a
False Pos.	2.7 (3.8)	2.0 (2.8)	1.0	**0.0**	0.7 (1.0)	0.5 (0.7)	0.0	**0.0**
Precision	53%	71%	89%	100%	n/a	n/a	n/a	n/a
Not Found	7.0 (2.0)	5.0 (1.4)	2.0	**0.0**	0.0	0.0	0.0	**0.0**
Type **CRR**: To find: **2 simple and 3 complex** conflicts								
Fully	0.0 (0.0)	**0.0** (0.0)	0.0	**2.0**	0.5 (0.8)	0.5 (0.7)	**2.0**	**3.0**
Partially	0.8 (0.4)	1.0 (0.0)	1.0	0.0	0.7 (0.8)	0.0 (0.0)	1.0	0.0
Recall (f+p)	42%	50%	50%	100%	39%	17%	100%	100%
False Pos.	**2.2** (2.3)	**4.5** (0.7)	1.0	**0.0**	**0.0** (0.0)	**0.0** (0.0)	0.0	**0.0**
Precision	28%	18%	50%	100%	100%	100%	100%	100%
Not Found	1.2 (0.4)	1.0 (0.0)	1.0	**0.0**	1.8 (0.8)	2.5 (0.7)	0.0	**0.0**

Complex CRC conflicts seem to be very hard to identify manually: from 7 existing conflicts only 0.9 conflicts on average have been identified completely, 2 conflicts have been identified partially; i.e. less than 45% of conflicts have been identified manually. On the other hand, false positives have been introduced. We expected that the number of false positives is reduced during group harmonization, but the opposite was the case: the amount of false positives increased from average individual results to the avg. group results. This is due to the complexity of the conflicts and the discussion that was caused by it during group harmonization.

Comparing completeness of simple and complex conflicts shows that the differences between manual and OntRep results increase with the level of complexity: e.g. regarding complex CRC conflicts the difference between manual and automated results is higher than regarding simple CRG conflicts where 8 conflicts have been found manually (as optimum) and 10 conflicts have been found completely with the OntRep.

Comparing correctness of simple and complex conflicts shows that correctness of identified complex conflicts is higher than of simple ones: For example, regarding CRR 2.6 false positives have been introduced for simple conflicts in average. On the other hand, no false positives have been identified for complex conflicts. The OntRep results for all types of conflicts are complete: all 22 conflicts of the defined conflict classes in

the given data were identified, due to the 3 reasons for 100% conflict identification described in section 5. We did not find any significant differences regarding the Mann-Whitney Test at a significance level of 95% (2-side). The necessary efforts for both the manual and automated approach are reported in [4].

VI. Discussion

The results of conflict analysis with OntRep seem convincing for the conflict types. The automated conflict analysis depends on the following factors:

- *Requirements structure*: OntRep analyzes different sections of each requirement to map them to the according concepts and to identify conflicts. Thus, a certain structure (EBNF) is necessary. When documents are used, this syntax is not used so frequently, but the importance of such a grammar increases when requirements databases are used instead of documents, because requirements have to be understandable and clear even without the context that usually exists in a requirements document.
- *Completeness of glossary*: the detection of logical conflicts depends on the terms captured as glossary terms in the ontology. OntRep recognizes conflicts between requirements that use different terms only if these terms are appropriately defined and their dependencies are clear. Usually, there is a project glossary existing in a project that can be imported into the ontology without big effort, so that all relevant terms are available for analysis.
- *Quality of ontology*: another prerequisite is correct modeling of the ontology, which requires expert knowledge. If the user does not specify it correctly, the tool cannot find the conflicts. If a constraint is missing in the ontology, the tool does not find any conflicts linked to that constraint. If a constraint has been specified wrongly, the tool might find conflicts with that constraint it was not intended to be found by the user. So if all facts have been modeled correctly, the tool will find all corresponding conflicts.

Despite the given prerequisites and limitations of OntRep, we think that this study is relevant, because its focus was on (a) evaluating the general technical feasibility of applying semantic technology to automated requirements conflict analysis, and (b) compare the results with the results of a manual approach for a practical-relevant set of conflict types.

VII. Conclusion and Further Work

Keeping requirements consistent is a main success factor for software development projects. That is why conflict analysis activities are important for requirements managers and project managers. However, the manual conduct of these activities takes significant effort and is error-prone, especially with an increasing number of requirements. Another issue is that participants with different domain backgrounds and terminologies have to work together in large distributed projects.

In this paper we proposed semantic technology as foundation for automating requirements conflict analysis and introduced the automated ontology-based reporting approach OntRep based on a project ontology and a reasoning mechanism. We used requirements formulated in EBNF as input to the proposed OntRep approach, which supports automated requirements conflict analysis. We evaluated the effectiveness of the OntRep conflict analysis approach referring to (a) different types of conflicts and (b) different levels of conflict complexity in a real-world industrial case study with 6 project managers in 2 teams. In addition a requirements expert and an OntRep user performed the same tasks to enable comparing the quality of results. Regarding the given conflict types the results of the evaluation are similar: OntRep found all conflicts in the requirements during the empirical study, while manual conflict analysis identified 30 to 80% of the conflicts for each conflict type and produced more false positives.

Further work will focus on the replication of this pilot study in a larger context. In addition, we want to increase the number of requirements and conflicts to be analyzed in order to get (a) more accurate numbers regarding recall and precision of the automated conflict analysis approach (b) more meaningful data regarding efforts of OntRep and manual conflict analysis approach for a higher number of requirements and conflicts.

Acknowledgment

We want to thank Alexander Wagner for the implementation of OntRep. This work has been supported by the Christian Doppler Forschungsgesellschaft and the BMWFJ, Austria.

References

[1] F.Y.Y. Choi, “Advances in domain independent linear text segmentation”, Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference., Morgan Kaufmann Publishers Inc., Seattle, Washington, 2000

[2] A Egyed, P Grünbacher, “Identifying Requirements Conflicts and Cooperation: How Quality Attributes and Automated Traceability Can Help”, IEEE software, 2004

[3] B. Freimut, T. Punter, St. Biffl, M. Ciolkowski, “State-of-the-Art in Empirical Studies”, Report: ViSEK/007/E, Fraunhofer Inst. of Experimental Software Engineering, 2002

[4] M. Heindl, T. Moser, D. Winkler, St. Biffl, „Requirements Management with Semantic Technology“,Vienna University of Technology, Technical Report IFS-QSE-10-03, http://qse.ifs.tuwien.ac.at/publication/IFS-QSE-10-03.pdf, March 2011

[5] C.L. Heitmeyer, R.D. Jeffords, B.G. Labaw, “Automated consistency checking of requirements specifications”, 2nd International Symposium on Requirements Engineering (RE '95), York, England, 1995

[6] J. Jackson, “A Keyphrase Based Traceability Scheme”, IEE Colloquium on Tools and Techniques for Maintaining Traceability during Design, pp.2-1-2/4, 1991

[7] H. Kaindl, “The Missing Link in Requirements Engineering”, ACM SigSoft Software Engineering Notes, vol. 18, no. 2, pp. 30-39, 1993

[8] E.D. Liddy, “Natural Language Processing”, 2 ed. Encyclopedia of Library and Information Science. NY. Marcel Decker, Inc., 2001

[9] G. A. Miller, “WordNet: A Lexical Database for English”, Communications of the ACM Vol. 38, No. 11: 39-41, 1995

[10] C. Pedrinaci, J. Domingue, A. K. Alves de Medeiros, “Core Ontology for Business Process Analysis”, 5th European Semantic Web Conference, 2008

[11] C.J. van Rijsbergen, S.E. Robertson, M.F. Porter, “New models in probabilistic information retrieval”, British Library Research and Development Report, no. 5587, 1980

[12] C. Rupp, Requirements Engineering und –Management, Hanser, 2002

[13] C. Wohlin, M. Höst, et al., “Controlled Experiments in Software Engineering”, Journal for Information and Software Technology, 921-924, Elsevier Science BV, 2001.

A Process Oriented Approach to Model Non-Functional Requirements Proposition Extending UML

Aneesh Krishna
Department of Computing, Faculty of Science & Engineering
Curtin University of Technology, WA 6102, Perth, Australia
a.krishna@curtin.edu.au

Abstract

Non-functional requirements (NFRs), sometimes termed quality, or quality of service, attributes or requirements, have been a topic of interest within systems engineering, software engineering, and requirements engineering for a considerable period of time. NFRs have been for too long overlooked during the development of software systems. This has led to numerous cases of failure resulting in over budgeting or cancellation of projects. This paper proposes a method to extend the UML diagrams to model NFRs. A car rental case study is provided to illustrate the use of the proposed method.

1. Introduction

Functional Requirements (FRs) determine what functions a software product should offer. Multiple proposals have been developed to analyze and model them, and are nowadays widely used. For Non-Functional Requirements (NFRs), the problem is more complex and due to that they have been ignored for a long time in the early phases of the software development process. NFRs are defined as constrained on the emergent properties of the overall system. Hence, software products are sometimes refined late in the development process to satisfy NFRs, this as a result leads to over budget or cancelled projects [1]. Software engineers now acknowledge the importance of dealing with NFRs in the early phases of the development process [1]; however, there is lack of credible proposals to be used in the software development. In this paper we propose a solution to integrate the analysis and modelling of NFRs into UML [2], which is the most popular language for modelling purposes in object-oriented paradigm. Our goal is to offer a method to analyze, refine and integrate NFRs in existing diagrams to provide justifications for the design.

Chung et al [1] developed a framework to represent and analyze non-functional requirements. This framework is an interesting one; having potential to fully analyze the dependencies between non-functional requirements. This work uses the concept of 'claim' in the diagram. Claims are linked to interdependency, and provide a rationale to design decision. However, having multiple claims can make diagrams complicated and harder to read. Cysneiros et al [3] first proposed a method to integrate NFRs into ER and UML data models. This strategy uses the notion of lexicon, namely, the Language Extended Lexicon (LEL), as an anchor to integrate NFRs into OO models. The idea of LEL is to register the vocabulary (symbols) of a project, and determine the relations between the symbols. The concept of LEL is not widely accepted/used in software industry. Imposing its usage would hugely increase the workload of the systems analysts. Therefore, another solution is needed to link NFRs to UML diagrams that would have practical usefulness in industry. This work also considers that functional and non-functional aspects should be carried through two independent cycles with convergence points, while efforts should be carried out to merge the two aspects. Cysneiros's proposal can be used to integrate NFRs into any UML diagram, but no support is provided to deal with diagrams other than Class Diagrams. These reasons form the basis for our proposal in this paper.

This paper is structured as follow: Section 2 formally formulates the proposal, providing new notations to deal with NFRs through an example. Finally, Section 3 concludes the paper.

2. Incorporation of NFRs into UML

The proposal considers that NFRs and FRs should not be carried out through two independent cycles, but should be part of a common, global strategy. This decision had a huge impact on the process designed. The process can be visualized in Figure 1.

The main steps of the proposal are the following:

1. First, we determine what the FR and NFRs are.
2. From the FRs, we shall draw the use cases, and then associate the NFRs with the different parts of the system they control. When associating them, two cases may arise: either the NFR is specific to an FR (e.g. 'the authentication must be fast'), or it is global and thus not linked to a single FR (e.g. 'all screens of the system should look similar'). In the first case, the NFR is linked with a use case, else with the entire system.
3. The NFRs will then be refined using the use cases. In this step, the systems analysts will use their experience to find missing NFRs that should be included

in the diagrams. This is quite important, because the stakeholders do not always think about every single NFR (some might seem obvious, other irrelevant at first), or some of them are simply hard to come up with. This should prevent NFR from emerging at later stages of the project.

4. Using the FRs and NFRs that have been extracted, a requirement graph will be drawn. This graph will show the decompositions of the NFRs into subgoals, how to satisfy the sub-goals with operationalizations, and the way all these elements interact with each other. The focus here is solidly on FR that can impact NFRs, so not every FR will be included in this graph.
5. Study the interdependencies between the different requirements, their subgoals and their operationalizations. If necessary, perform trade offs (for example, using uncompressed format is good for performance, but not for space usage), and more generally, refine the requirements graph by studying the impact of each interdependency on the related requirements (see subsection 3.4).
6. Finally, use all the information gathered to draw other UML diagrams. This paper highlights only the application of this to sequence, class and deployment diagrams.

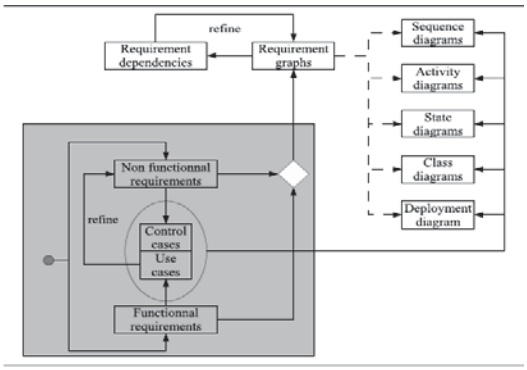


Figure 1: Diagram of the proposed strategy

Use cases controlled by NFRs: Incorporation of NFRs into use case diagrams is done by using a solution similar to [4]. A prerequisite for this is a complete or near complete list of the system requirements. FRs are subsequently used to produce use cases. Each NFR is documented with three pieces of information: the interested stakeholder, the NFR's type and NFR identification code (ID). Representing an NFR in the diagram, these three pieces of information are particularly important. First, we need a non ambiguous way to link a use case to an NFR, which is done by stating the ID of the NFR. Then, including both the stakeholder and the type of the NFR in the representation provides the systems analysts' with useful information without overloading the schema.

The general representation of such a diagram is presented on Figure 2.

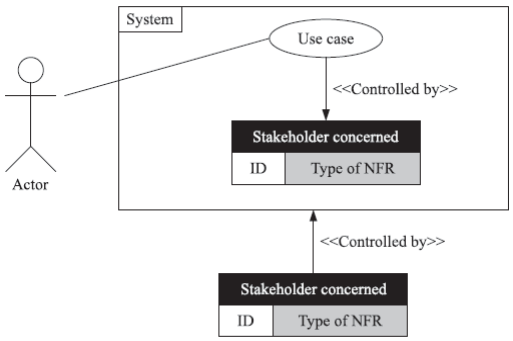


Figure 2: Actors and use cases controlled by NFRs

Construction of the NFRs graph: The aim of this step is to decompose an NFR into sub-softgoals [1]. Then, sub-softgoals that can't be subdivided find alternatives to operationalize it. Therefore, the following process has to be completed for each NFR represented in the use cases/control cases diagram. Add the object that the NFR is related to. Given that the NFR is not linked to a use case in this diagram, it is important to know on what the NFR acts (e.g. if we consider the informal high-level NFR 'Good Performance for accounts', the object is 'Account'). Next, the analyst has to specify the priority of the NFR. This is important to algorithmically determine what subset of the NFRs can actually be satisfied. The mark 1 means that the NFR can easily be ignored, while 5 mean that the NFR has to be satisfied no matter what. With the conclusion of the process, the NFR is renamed 'Formulated NFR' as shown in Figure 3.

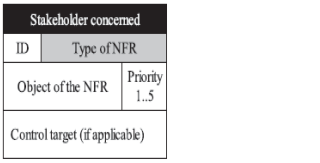


Figure 3: Representation of a high-level NFR

Sub-softgoals are subdivided until the analysts decide that no more decomposition is feasible or interesting. If possible, the analysts must try to avoid dividing a subgoal using only 'helps' contributions. If a softgoal does not have a sub-softgoal (or a group of sub-softgoals) which 'satisfies' it, it will be hard to determine if the higher level softgoal can actually be met. Once no more division is needed, possible operationalizations are associated to each leaf of the constructed graph. To do that, one must think about the most intelligent solutions to respond to the need expressed by a softgoal. All these alternatives will be represented on the graph using a unique ID and linked to the corresponding softgoal using the same connectors than before.

At the end of this phase, the analyst should have a graph

of formulated NFR, each with its own set of sub softgoals and operationalizations. The next step is to study the interactions between graphs and to incorporate the necessary FR.

Studying interdependencies in the requirements diagram: At this point of the process the analysts concentrate on the interdependencies between the different requirements. First all the NFRs graphs which are related are grouped. Obviously, the NFRs controlling the entire system must appear on each diagram. For the others it is quite easy: the NFRs linked to the same use cases, or having the same 'object' (Figure 3) can interact or be in conflict. Once all the related NFRs are on the same diagram, one uses the use case diagram and the associated documentation to add all the related FRs to the diagram. FR can also have an ID.

What remains is the selection of the operationalizations which will be used to design the system. In this work, we don't propose any algorithm (this forms part of the future investigation on this topic) to solve this problem, but such an algorithm should:

- analyze the impact of choosing each operationalization (which requirements are satisfied or helped, which operationalizations or requirements are hurt or broken)
- assure that every FR (and NFR of priority 5) is satisfied (one of its operationalization is chosen)
- give a mark to each alternative subset (using the weight of the contributions and the priority of the NFRs)
- return the highest-ranked solutions

The analyst then chooses the most appropriate subset for the design of the system. The chosen operationalizations are marked with the letter "**C"** in the 'Expected status' box, while the others with the letter "**R",** "Rejected". The 'Expected status' of the Formulated NFRs and softgoals will then be updated according to this choice.

UML diagrams: This section demonstrates how to integrate NFRs and their operationalizations into UML diagrams. The focus is on sequences, deployment and classes, but the techniques can work on other diagrams.

The following links can be used in these diagrams, as shown in Figure 4

- The «implements» link means that the element has been added to the design of the product in order to satisfy the operationalization. If the operationalization is not considered anymore, this element can be removed.
- The «contributes to» link means that an element previously existing in the diagram is used by the operationalization. It is important to highlight such an element: if it has to be modified to comply with its primary function, the systems analysts must study the impact of the change on the realization of the operationalization.
- The «controlled by» link defines that one knows that the implementation has to be guided by a particular NFR but no specific solution has been chosen yet. Most of the time, such a link must come with an annotation.

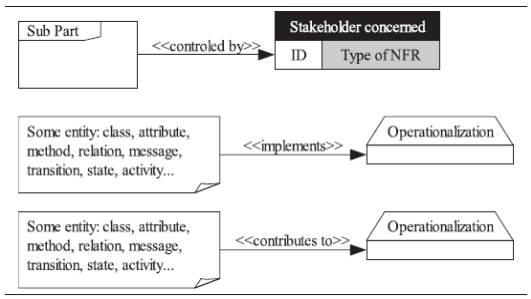


Figure 4: Links to be used in UML diagrams

Sequence Diagram: Once the sequence diagrams are ready, the analysts will incorporate the NFRs into them. Usually, a sequence diagram will be related to one of the use cases that has been designed earlier, and this use case might be related to one or more NFRs. If this is the case, then for each of them, the analyst has to ask himself: where does this NFR impact the sequence diagram, and how? The analyst can choose the level of detail for the diagram as shown on Figure 5. Therefore the analyst can:

- state that a subsequence is «controlled by» an NFR,
- link some specific previously existing messages to a operationalization («contributes to»),
- add new messages (and maybe new classes) to the diagram which «implements» an operationalization.

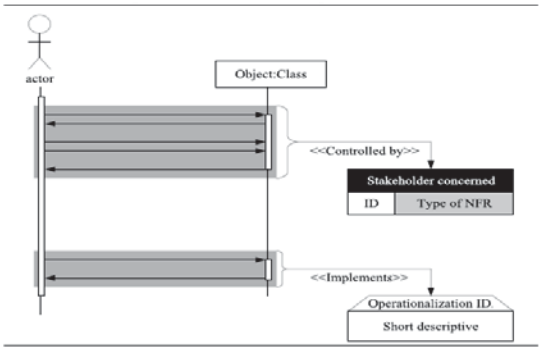


Figure 5: Representation of an operationalization

Class Diagram: The approach to incorporate NFRs into class diagram is quite simple and is shown in Figure 6.

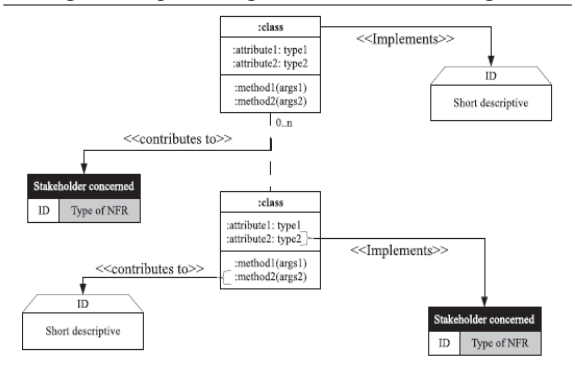


Figure 6: Incorporating NFRs into class diagrams

Verification: Once the NFRs or their operationalizations have been included in every necessary diagram, one has to verify that everything planned on the NFR graphs is actually considered in the design of the product. This justifies the existence of the box 'Current status' at the bottom of the representations of NFRs and operationalizations in the NFR graphs. For each operationalization chosen, one has to check if it appears on diagrams with links of type «implements» or «contributes to». If it is the case, then the operationalization has been 'satisfied', or else it was 'not handle'. Then update the 'current status' of all NFR softgoals that the operationalization links to. After considering the operationalization, if an NFR softgoal that was expected to be 'satisfied' or 'weakly satisfied' is not, it has to be verified if this NFR is linked to any other diagram with relation «controlled by». If so, the NFR has not been forgotten, however, no technical solution has been indicated on the design. If it is the case, it is up to the analyst to decide if this is good enough to mark to softgoal as 'weakly satisfied'/'satisfied' or if the design needs to be refined. These verifications should in fact be done automatically by a framework implementing our proposal. After these verifications, the operationalizations which were chosen but are not satisfied and the NFR softgoals where 'expected status' is different from 'current status' should be highlighted for the analyst to know immediately what is missing.

Car Rental Case Study: This case study demonstrates the application of the method for designing an e-commerce application for online car rentals. The example concentrates on core aspects of such a system with emphasis on business processes that relate to rental and provision of information to customers regarding the company. The complete explanations of the approach (along with case study) can be had by visiting http://www.computing.edu.au/~aneesh/NFR (details withheld due to space limitations).
Use cases enriched with NFRs are shown in Figure 7.

The sequence diagram of the system is depicted in figure 8. The functionality depicted is limited to the searching process for a car.

3 Conclusions and Future Work

This paper presented a novel approach to dealing with Non-Functional Requirements and to model them in UML diagrams. This work constitutes an important contribution given that it allows system analysts to consider NFRs in their models without requiring additional learning of new software. Part of our future plans include the automation of elements of the proposal in order to utilise its full usefulness in industrial settings. We accept that the full proposal is very difficult to automate, as there is a need to provide manual support by systems analysts at certain steps of the approach. However, this is the case with almost all methods proposed so far in this challenging area.

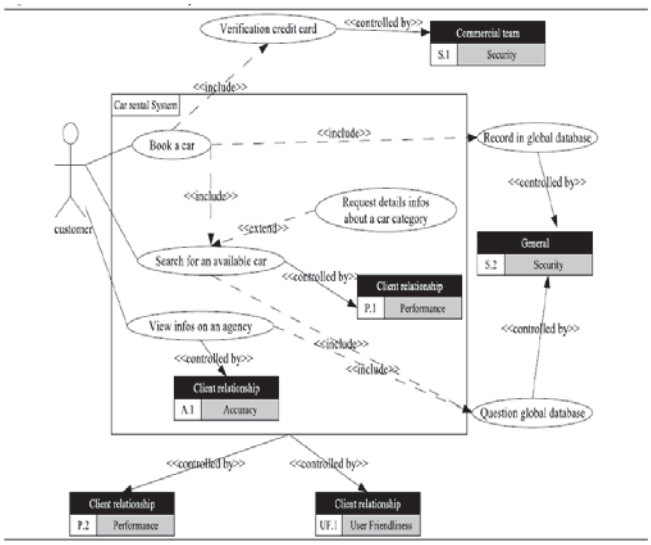


Figure 7: Use cases enriched with NFRs

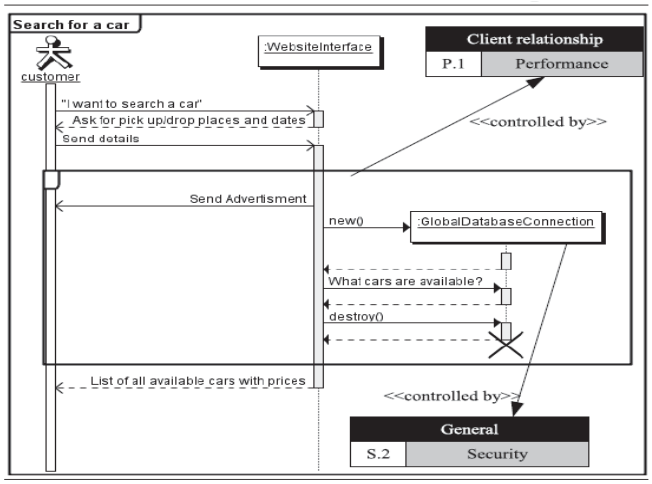


Figure 8: Sequence diagram of the searching process

References

[1] Chung, L., Nixon, B. A., Yu, E., Mylopoulos, J. *Non-Functional Requirements in Software Engineering*, Kluwer Academic Publishers, 2000

[2] OMG UML, *Unified Modeling Language (UML),* http://www.omg.org/spec/UML/ (accessed 10 May 2011)

[3] Cysneiros, L. M., do Prado Leite, J. C. S. & de Melo Sabat Neto, A framework for integrating non-functional requirements into conceptual models, *Requirements Engineering 6(2)*, pp. 97–115, 2001

[4] Zou, J., Pavlovski, C. J. Modeling architectural non functional requirements: From use case to control case. *Proceedings of IEEE International Conference on e-Business Engineering*, Shanghai, IEEE Computer Society, pp. 315–322, 2006

Use Case Driven Extension of ProjectIT-RSL to Support Behavioral Concerns

David de Almeida Ferreira, Alberto Rodrigues da Silva
INESC-ID / Instituto Superior Técnico
Rua Alves Redol 9, 1000-029 Lisboa, Portugal
david.ferreira@inesc-id.pt, alberto.silva@acm.org

***Abstract*—Requirements Engineering is a system engineering discipline of paramount importance. Its primary deliverable is the requirements specification, a document that entails the detailed description of business-specific needs to which the target software system must comply to. Despite the advances brought by modeling techniques, the specification of software systems still consists mostly in manually writing down requirements in natural language. Resorting to natural language to convey requirements has some advantages, such as its expressiveness and the stakeholders proficiency at using it for communication purposes, when compared with semi-formal or formal languages. However, ad-hoc natural language has intrinsic characteristics that make it error-prone, some of which may even hinder the effort of complying with good requirement criteria, regardless the usage of an automated requirements tools. To address this problem, within the scope of the ProjectIT initiative, a requirements specification language named ProjectIT-RSL was developed, based on common linguistic patterns. Despite covering several structural aspects of the software system under specification (e.g., actors, components, entities, properties, and relationships), ProjectIT-RSL still lacks the support for capturing behavioral concerns.**

This paper presents a Use Case Driven extension for ProjectIT-RSL , with the purpose of supporting the specification of functional requirements according to a controlled natural language approach. The proposed solution combines patterns from Use Case textual description best practices with Pseudocode specifications.

***Keywords*-Requirements Engineering; Specification Language; Controlled Natural Language; Use Cases; Pseudocode.**

I. Introduction

The size and complexity of modern large-scale software systems demand for high-levels of abstraction, conceptual reasoning, and validation upfront. Namely, a clear definition of the real business-specific needs is required. Therefore, regardless of the achievements in both industry and academia, the development of software systems is still a rather challenging process. From all system engineering disciplines, it is argued that Requirements Engineering (RE) is one of the most important regarding this issue, since it supports the work of several other disciplines throughout the entire software product life-cycle [1]: it provides useful scope and status information to Project Management upstream and provides a stable basis for development downstream, namely for design and testing.

Typically, the development of a software system begins with Requirements Engineering (RE) [2]. RE deals with the early activities related with the process of discovering the purpose, scope, stakeholders, boundaries, and actors of the system being specified [3]. These early activities can be classified as the *requirements development process*, whereas the activities of evolving accepted requirements (e.g., dealing with change requests, impact analysis, tracing, and status-tracking) can be regarded as the *requirements management process* [4]. In short, RE is concerned with real-world goals for software systems functionality, and also how it can be precisely specified and maintained throughout the software development process [5].

However, often the importance of RE is underestimated, resulting in a large amount of rework: any attempt to start technical work beforehand, without a deep understanding of the target software system's purpose, will certainly jeopardize the project outcome [6].

The main deliverable of RE is an artifact (usually called *requirements document*) that contains the detailed textual description of *what the target system should do*, or *constraints on its behavior* [7]. However, this form of specification (based on natural language) is ambiguous and, in many cases, unverifiable because of the lack of a standard machine-executable representation [7]. The main problem appears during the translation of natural language requirements into a formal or, at least, semi-formal computer model [8]. Despite these complementary approaches, natural language textual specifications are still the most suitable, fast, and preferred manner (by non-technical stakeholders) to contribute and validate requirements specifications [7].

Taking these facts into consideration, within the scope of the ProjectIT initiative [9], [10], a requirements specification language named ProjectIT-RSL [11] was developed. This initiative seeks to address some of the aforementioned problems, since it advocates that the software development process should be focused on higher-level activities (such as Project Management, Requirements Engineering, and Architectural Design), instead of repetitive and error-prone manual activities. Following this mindset, ProjectIT-RSL was designed to increase the rigor of requirements specifications to enable tool support, namely to provide user feedback

This work is supported by FCT (INESC-ID multi-annual funding and PhD Scholarship SFRH/BD/36972/2007) through the PIDDAC Program.

during the specification process and the automatic generation of initial design drafts upon validated requirements.

Despite the concrete results achieved, there are still several open issues, namely the lack of support for the definition of software system behavior. To cope with real-world problems, the controlled natural language approach provided by ProjectIT-RSL needs to cover not only static aspects of the target system (e.g., *domain model* concepts) but also dynamic aspects related with functional requirements.

The remainder of this paper is organized as follows. Section II presents a review of related work regarding the specification of software system behavior. Section III presents the Use Case Driven extension of ProjectIT-RSL and introduce an illustrative example. Finally, Section IV concludes this paper with final thoughts and discussion.

II. Related Work

Historically, the requirements specification process has consisted in creating a natural language description of *what the target system should do* or *constraints about its behavior* [4], [7]. As opposed to constructed, artificial, or any other kind of engineered languages (with specific purposes in mind), the term *natural language* refers to any language commonly used by humans in their daily communication. Its usage to support RE activities has already been studied for a long time [8]. However, this form of specification is both ambiguous and, in many cases, unverifiable because of the lack of a standard machine-executable representation [7].

To mitigate the undesirable effects of natural language, one can impose some restrictions to reduce its complexity while maintaining its naturalness. *Controlled Natural Languages* (CNL) are subsets of natural languages whose grammars and vocabularies have been engineered in order to reduce (or even eliminate) both ambiguity (to improve computational processing) and complexity (to improve readability). CNLs are typically designed for knowledge engineering, not addressing behavioral aspects. A thorough analysis on the state-of-the-art of CNLs is provided in [12]. Attempto Controlled English (ACE) [13] is a good representative of this category of textual languages, because it is a mature general-purpose CNL with a solid toolset and a wide literature body of knowledge[1]. Furthermore, ACE possesses some interesting (and useful) characteristics, namely every ACE sentence is unambiguous, even if people may perceive the sentence as ambiguous in full English [14].

To address functional aspects, namely to specify complex behavior (implementation-independent algorithms and other abstract computations), one can resort to *Pseudocode* [15]. The high-level algorithmic descriptions achieved with Pseudocode could be integrated with the previous CNL approaches to address their weaknesses in terms of behavioral expressivity. However, Pseudocode lacks a widely accepted and well-defined standard (only some guidelines are defined [16]). We considered that *PlusCal* [17], [18] is a suitable representative for this approach. Despite having formal roots, PlusCal was designed having in mind some of the Pseudocode's guidelines (e.g., simplicity), which in turn reflects on *PlusCal*'s design principles and syntax. PlusCal is an algorithm language that can be used to replace Pseudocode, for both sequential and concurrent algorithms [17]. Specifications in PlusCal can be automatically translated into another formal language enabling the verification of formal properties through model-checking.

Additionally, one can resort to semi-formal *graphical* approaches. This category of languages is well represented by *UML*, the general-purpose language endorsed by OMG and considered by many as the *de facto* standard for modeling and documenting object-oriented software. Usually, modelers use Activity and Sequence UML diagrams to specify software system behavior. However, the information conveyed by UML models is usually incomplete, informal, imprecise, and sometimes even inconsistent. These flaws are caused by the diagrams' limitations. These diagrams cannot fully convey the details of a thorough specification.

Also, there are hybrid approaches such as Use Cases: UML Use Case diagrams can be complemented with their textual descriptions counterpart. Use Cases provide a natural approach to organize and describe the overall system functionality through descriptions, in abstract terms, of how actors interact with the system to accomplish their goals [19]. Although they are not suitable to describe functional requirements of all kinds of systems, they are adequate to specify general information systems [19]. Also, since they describe the system functionality from an actor's perspective, they can be easier to read and understand by non-technical stakeholders than more formal or complex notations (formal methods or UML diagrams enhanced with OCL). However, although it might seem a trivial task, the relatively small number of concepts, allied with the lack of a rigid set of structuring rules (or compositional rules, when dealing with Use Case diagrams), makes people often struggle to write (or model) useful and adequate Use Cases.

Finally, there are some combined approaches. Natural Language Processing (NLP) approaches to automatically derive conceptual models from requirements texts are not new [8], and neither is the usage of formal methods to support consistency checking through mapping techniques [20]. Debnath et al. [21] propose a solution to integrate requirements written in natural language into the OMG's Model-Driven Architecture. They defined manual derivation strategies to obtain conceptual models and formal specifications from textual requirements. Their approach follows an OCL-based transformation process to define Computer Independent Model (CIM) models from natural language oriented models. Mala et al. present research regarding Use Cases combined with NLP techniques. They present an ap-

[1] http://attempto.ifi.uzh.ch/site/pubs/

proach for the automatic construction of object-oriented design models in UML from the natural language requirements specification. Unlike others, their approach combines domain modeling, Use Case descriptions (written in a restricted form of natural language), and non-functional taxonomy to derive non-functional requirements prioritization through trade-off analysis. Konrad et al. [24] also attempts to ease the integration of formal specification languages through the process of specifying properties of formal system models in terms of natural language and formally analyzing these properties using existing formal analysis tools. Still regarding natural language approaches, the work of Meziane et al. [8] follows a different approach to mitigate miscommunication problems by generating natural language specifications from UML class diagrams.

Despite this wide range of techniques and approaches, many problems concerning the pragmatic usage of these solutions are still unresolved [25].

III. ProjectIT-RSL Extension

ProjectIT-RSL is a controlled natural language aligned with a metamodel derived from common linguistic patterns [11]. This language was designed to convey information in such manner that a tool can extract requirements models from its textual specifications. ProjectIT-RSL metamodel already provides concepts (e.g., Operation, Activity, and Action) to convey information about software system behavior. However, until now, the concrete realization of the dynamic part of the metamodel in terms of language constructs has not been addressed [10]. Thus, ProjectIT-RSL lacks the means to convey behavioral concerns about the software system under consideration. The specification of complex interaction patterns between actors and the target system is crucial for gaining a thorough understanding of the software system's functionality, namely to perform a more extensive analysis (e.g., consistency checking and model-checking) on requirements specifications written in this controlled natural language. This kind of feedback is crucial to increase stakeholder engagement in RE activities related with verification and validation.

According to the object-oriented paradigm, so far the conceptualizations that ProjectIT-RSL conveys are typically referred to as a *domain model*. This kind of conceptualization describes the various entities, their properties and relations (an example is provided in [26]). The domain model provides a structural view of the domain of interest, which can be complemented by other views focusing dynamic aspects, such as the one presented in this paper.

Considering the original mindset and design principles of ProjectIT-RSL [11], namely the grammar and writing style, we follow the seminal results presented in [27]. Hence, our proposal extends the previous version of ProjectIT-RSL by combining Use Case textual descriptions with Pseudocode.

A. Reusing and Structuring with Use Cases

Besides the metamodel previously mentioned, ProjectIT--RSL specifications also follow another metamodel (depicted in Figure 1) with different concerns. The former is related with the specifications' content, whereas the latter addresses the best practices regarding requirements document's structure. Our proposal in this paper focuses on providing an extension for the *Functional Requirements* section.

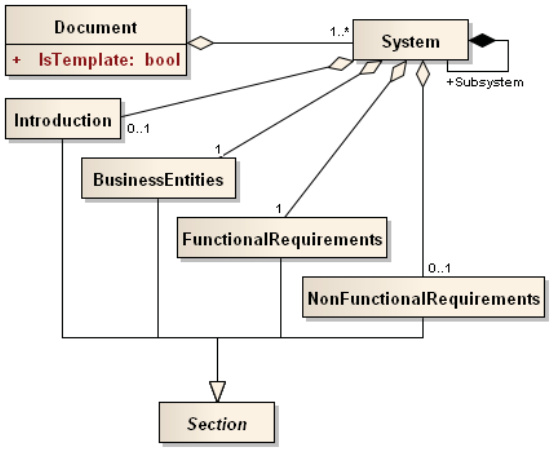


Figure 1. ProjectIT-RSL structural metamodel.

Given the match between our goals in terms of representation of functional requirements and the purpose of Use Cases specifications, we consider that, to avoid "reinventing the wheel", Use Cases are the best underpinning for developing the ProjectIT-RSL extension. Use Cases employ a small set of straightforward concepts that are easily understood by non-technical stakeholders. We also benefit from a well--defined body of knowledge [28]–[30], namely how these specifications can be structured and reused in an easy to understand and industry-standard manner. Finally, bearing in mind the current ProjectIT-RSL support for domain modeling, by adopting Use Cases for conveying functional requirements, we become aligned with the early stages of the well-known ICONIX process [30]. The major pitfall of using Use Cases as a requirements specification technique is the lack of well-defined rules. Nevertheless, this problem can be addressed by providing a Use Case context-free grammar that is aligned with the metamodel presented in Figure 2.

This extension enhances the *Functional Requirements* section by providing a subsection named *Use Cases*, which contains all Use Case specifications of the (sub)system being considered. An illustrative example is presented in Listing 1. This Use Case describes a simple and common interaction with a web application, in which a registered user tries to recover a password. The textual description already provides syntax highlighting according to a context-free grammar aligned with the metamodel presented in Figure 2.

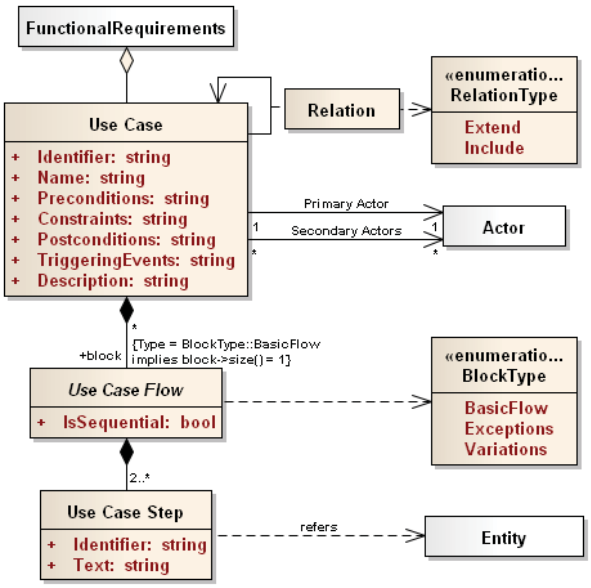


Figure 2. Use Case Driven metamodel.

B. Complex Behavior Specification with Pseudocode

To cope with the description of complex software system behavior, we advocate the usage of Pseudocode-like descriptions. According to our proposal, these Pseudocode--like [15], [16], [31] snippets act as adornments for Use Case steps. They only should be used when a fine-grain level of detail is required, namely to customize the intended action meaning or to entail the detailed specification of a complex behavior (not provided by built-in actions). The verbs used while describing detailed Use Case steps have linguistic properties (such as thematic relations[2]) that allow us to consider them as extension points to which functions in Pseudocode can be hooked up (if the function presents the same number of arguments and constraints as the verb theme). Gottesdiener [19] provides a comprehensive list of suggested verbs for both *informative* Use Cases that provide information to actors (e.g., *find*, *list*, *notify*, *select*, *view*) and *performative* Use Cases that allow actors to handle complex tasks (e.g., *approve*, *authorize*, *choose*, *send*, *submit*, *validate*). By resorting to this hook up mechanism we gain an additional level of flexibility regarding the possibility of providing custom-made functional specifications for the specific needs of the project at hand.

Pseudocode often appears in the literature as a simplified programming language to describe algorithms. Regarding the Pseudocode syntax, we adopt PlusCal's due to its benefits: we do not "reinvent the wheel" by designing a concrete syntax for Pseudocode, and we also automatically ensure compatibility with PlusCal's formal verification tools. PlusCal's documentation [17] already provides a context-free grammar that we reuse within the ProjectIT-RSL extension for specifying Pseudocode-like snippets.

```
1. System "MyWebApp"
  1.1. Section Business Entities (...) End Section
  1.2. Functional Requirements
    1.2.1. Section Actors Declaration (...) End Section
    1.2.2. Section Use Cases
      BeginUseCase
      Identifier: "UC22"
      Name: "Recover Password"
      PreConditions: Previous{"UC21"/"Authenticate User"} ;
                     State{"The User is not authenticated."}
      PostConditions: State("The system updates the User password.")
      PrimaryActor: "Anonymous User"
      SecondaryActors: "Mail Server"
      TriggeringEvent: "The user does not know the password."
      Description: "Verify if the e-mail is valid and if the e-mail is associated
                    with a registered user. Generate a new password and send the
                    new password to the user's email."
      BasicFlow: InTheFollowingOrder
          1. "The system asks the Anonymous User for the registration e-mail."
          2. "The Anonymous User inserts the registration e-mail."
          3. "The system verifies the registration e-mail."
          4. "The system generates a new password."
          5. "The system sends an e-mail with new password to the Anonymous User
              through the Mail Server."
          6. "The system displays a PasswordRecovered success message to the
              Anonymous User."
      Exceptions:
          3.   "The Anonymous User provides an invalid e-mail."
          3.1a "The system displays an InvalidEmail error message to the
                Anonymous User."
          3.1b "The system terminates the use case."
          3.   "The Anonymous User provides an inexistent e-mail."
          3.2a "The system displays an InexistentEmail error message to the
                Anonymous User."
          3.2b "The system terminates the use case."
          5.   "The Mail Server is unavailable."
          5.1a "The system displays a ServiceUnavailable error message to the
                Anonymous User."
          5.1b "The system terminates the use case."
      EndUseCase
    End Section
  End Section
End System
```

Listing 1. Use Case example: "Recover Password".

Following the example of Listing 1, we can illustrate the usage of Pseudocode to specify the behavior related with the verb *"send"* used within step 5 (*"The system sends an e-mail with new password to the Anonymous User through the Mail Server."*). According to VerbNet, the sentence's main verb (*"send"*) has three roles (arg0 the "sender", arg1 the "sent", and arg2 the "sent-to"). While mapping these roles we notice that step 5 contains additional information provided by a prepositional sentence ("through the Mail Server"). This piece of information will not be considered while hooking up the verb arguments with the built-in actions, unless we introduce a Pseudocode extension. We can specify a custom-made Pseudocode snippet (illustrated in Listing 2) to support this additional verb role (arg3 the "by-means").

```
--algorithm Send

variables maxAttempts = 3, secondsBetweenAttempts = 60

procedure SentToBy( variables sender, message, destination, channel )
  variables count = 0
  while ( count < maxAttempts )
    if ( call SendToChannel( message, destination, channel ) )
      return 'true'
    else
      count := count + 1
      call SleepSeconds( secondsBetweenAttempts )
    end if
  end while
  return 'false'
end procedure

end algorithm
```

Listing 2. "Send-To-By" Pseudocode specification example.

[2]Thematic relation is a linguistic term used to express a syntax–semantic correlation, namely the roles that a noun phrase plays with respect to the action or state described by a sentence's main verb.

C. Parsing of ProjectIT-RSL Extension

This extension of ProjectIT-RSL resorts to different parsing techniques, being the pipeline divided in two different stages. First, we make use of a structural template, derived from Use Case textual description best practices [28]–[30]. This derived template is processed with a context-free grammar similar to those used by programming languages. Next, after the structural information has been parsed, the process follows to the second stage of the pipeline. At this stage we make use of NLP techniques, namely shallow parsing[3] of the pieces of information previously captured according to its context within the structural template. The extracted information, namely from detailed Use Case steps, will populate models aligned with the ProjectIT-RSL metamodel. Pseudocode is used to specify complex behavior not covered by built-in actions associated with the verbs used while describing detailed Use Case steps.

We resort to shallow parsing techniques to address complexity of natural language, which hinders the possibility of building a general-purpose representations of meaning from ad-hoc natural language text [32]. To ease the burden of full parsing, we only look for very specific kinds of information in detailed Use Case steps, namely to populate models according to the *"Actor performs Operation on Entity"* core structure emphasized by the ProjectIT-RSL metamodel. Our shallow parsing approach can be considered as Chunking [32, Ch. 7, p. 261], more specifically a regular expression cascaded chunker. The possibility of adding new parsing rules that express linguistic patterns, namely to extract specific and contextualized pieces of information from detailed Use Case steps, makes the language extensible.

IV. Conclusion

Requirements Engineering is a mature discipline, and throughout the last decades several approaches to improve its processes and artifacts have been proposed. Yet, there are still open issues to address, namely problems regarding communication and requirements quality.

The ProjectIT initiative advocates that a greater attention should be given to the rigor and quality of requirements specifications. Namely, it emphasizes the need for tool support, because most RE techniques are manually applied, thus making them error-prone and less cost-effective when dealing with modern software systems, due to their size and complexity. Since requirements provide the foundation for most of subsequent technical work, these activities would certainly benefit from consistency checking of the requirements specifications. Also, unless suitable tool support is provided, to validate and generate an early design draft from requirements specifications, it will be difficult to seamlessly integrate RE with the Model-Driven Engineering paradigm.

[3] As opposed to full parsing, where one or more complete parsing tree are derived, shallow parsing techniques try to identify and extract information from relations between words.

Considering these facts, this paper presents an extension for the ProjectIT-RSL language, to address its current lack of support for the specification of behavioral aspects. To convey information about the software system behavior with ProjectIT-RSL, namely to specify functional requirements, we propose an integration of well-known techniques, more specifically textual Use Cases with Pseudocode specifications. The former is an industry-standard technique that provides a template for structuring functional requirements, thus reducing the complexity of natural language parsing with well-defined contexts. The latter provides a technology-agnostic approach to express complex behavior (the computations associated with the customized-actions, i.e., the verbs used within Use Case steps). This extension complements the current ProjectIT-RSL support for domain modeling and, although it is process-independent, it can be well supported by processes such as ICONIX or RUP.

As future work, in the short-term we plan to validate the ProjectIT-RSL language with real-world case studies. First, we are going to develop the required tool support by extending the current ProjectIT toolset, namely the parsing algorithms and text editor. In the long run, we plan to use the requirements models entailed within ProjectIT-RSL to infer or verify properties of the target system (e.g., model-checking or theorem proving) in order to ensure the overall quality of its specification.

References

[1] C. Hood, S. Wiedemann, S. Fichtinger, and U. Pautz, *Requirements Management: The Interface Between Requirements Development and All Other Systems Engineering Processes*, 1st ed. Springer, December 2007, ISBN: 978-3540476894.

[2] D. Firesmith, "Modern Requirements Specification," *Journal of Object Technology*, vol. 2, no. 1, pp. 53–64, March 2003.

[3] B. Nuseibeh and S. Easterbrook, "Requirements Engineering: a Roadmap," in *Proc. of the Conference on The Future of Software Engineering (ICSE'00)*. New York, NY, USA: ACM, 2000, pp. 35–46, ISBN: 978-1-58113-253-3.

[4] K. Wiegers, *Software Requirements*, 2nd ed. Microsoft Press, March 2003, ISBN: 978-0735618794.

[5] P. Zave, "Classification of research efforts in requirements engineering," *ACM Computing Surveys*, vol. 29, no. 4, pp. 315–321, December 1997, ISSN: 0360-0300.

[6] R. R. Young, *The Requirements Engineering Handbook*. Artech Print on Demand, November 2003, ISBN: 978-1580532662.

[7] H. Foster, A. Krolnik, and D. Lacey, *Assertion-based Design*. Springer, 2004, ch. 8 - Specifying Correct Behavior.

[8] F. Meziane, N. Athanasakis, and S. Ananiadou, "Generating Natural Language specifications from UML class diagrams," *Requirements Engineering*, vol. 13, no. 1, pp. 1–18, January 2008, DOI: 10.1007/s00766-007-0054-0.

[9] A. Silva, "O Programa de Investigação ProjectIT (whitepaper)," October 2004.

[10] A. Silva, J. Saraiva, D. Ferreira, R. Silva, and C. Videira, "Integration of RE and MDE Paradigms: The ProjectIT Approach and Tools," *IET Software Journal*, vol. 1, no. 6, pp. 217–314, December 2007.

[11] C. Videira, D. Ferreira, and A. Silva, "A Linguistic Patterns Approach for Requirements Specification," in *Pro. of the 32nd EUROMICRO Conf. on Soft. Eng. and Advanced Applications*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 302–309, ISBN: 0-7695-2594-6.

[12] T. Kuhn, "Controlled English for Knowledge Representation," Ph.D. dissertation, Faculty of Economics, Business Administration and Information Technology of the University of Zurich, 2010, Retrieved Friday 6th May, 2011 from http://attempto.ifi.uzh.ch/site/pubs/papers/doctoral_thesis_kuhn.pdf.

[13] N. E. Fuchs, U. Schwertel, and R. Schwitter, "Attempto Controlled English – Not Just Another Logic Specification Language," in *Logic-Based Program Synthesis and Transformation*, ser. Lecture Notes in Computer Science, P. Flener, Ed., no. 1559, Eighth International Workshop LOPSTR'98. Manchester, UK: Springer, June 1999.

[14] N. E. Fuchs, K. Kaljurand, and T. Kuhn, "Attempto Controlled English for Knowledge Representation," in *Reasoning Web, Fourth International Summer School 2008*, ser. Lecture Notes in Computer Science, no. 5224. Springer, 2008, pp. 104–124.

[15] S. McConnell, *Code Complete: A Practical Handbook of Software Construction*. Microsoft Press, June 2004, vol. 5565, ch. The Pseudocode Programming Process, ISBN: 978-0735619678.

[16] J. Dalbey, "Pseudocode Standard," Retrieved Friday 18th February, 2011 from http://users.csc.calpoly.edu/~jdalbey/SWE/pdl_std.html.

[17] L. Lamport, "The pluscal algorithm language," in *Proceedings of the 6th International Colloquium on Theoretical Aspects of Computing*, ser. ICTAC '09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 36–60, Retrieved Wednesday 15th December, 2010 from http://research.microsoft.com/en-us/um/people/lamport/pubs/pluscal.pdf. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-03466-4_2

[18] Microsoft Corporation, "The PlusCal Algorithm Language," January 2009, Retrieved Wednesday 15th December, 2010 from http://research.microsoft.com/en-us/um/people/lamport/tla/pluscal.html.

[19] E. Gottesdiener, *The Software Requirements Memory Jogger: A Desktop Guide to Help Software and Business Teams Develop and Manage Requirements*, 1st Spiral-Bound ed. Goal/QPC, October 2009, ISBN-13: 978-1576811146.

[20] W. E. McUmber and B. H. C.Cheng, "A General Framework for Formalizing UML with Formal Languages," in *Proceedings of the 23rd International Conference on Software Engineering (ICSE'01)*. Washington, DC, USA: IEEE Computer Society, 2001, pp. 433–442, ISBN: 0-7695-1050-7.

[21] N. Debnath, M. C. Leonardi, M. V. Mauco, G. Montejano, and D. Riesco, "Improving Model Driven Architecture with Requirements Models," *Information Technology: New Generations, Third International Conference on*, vol. 0, pp. 21–26, April 2008, ISBN: 978-0-7695-3099-4.

[22] G. S. A. Mala and G. V. Uma, *PRICAI 2006: Trends in Artificial Intelligence*, ser. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer-Verlag, July 2006, vol. 4099, ch. Automatic Construction of Object Oriented Design Models [UML Diagrams] from Natural Language Requirements Specification, pp. 1155–1159, ISBN: 978-3-540-36667-6.

[23] G. S. A. Mala and G. V. Uma, *Elicitation of Non-functional Requirement Preference for Actors of Usecase from Domain Model*, ser. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer-Verlag, December 2006, vol. 4303, pp. 238–243, ISBN: 978-3-540-68955-3.

[24] S. Konrad and B. H. C. Cheng, *Satellite Events at the MoDELS 2005 Conference*, ser. Lecture Notes in Computer Science. Springer-Verlag, January 2006, vol. 3844, ch. Automated Analysis of Natural Language Properties for UML Models, pp. 48–57.

[25] B. Cheng and J. Atlee, "Research Directions in Requirements Engineering," *ICSE FOSE*, 2007.

[26] D. Ferreira and A. Silva, "A Requirements Specification Case Study with ProjectIT-Studio/Requirements," in *Proceedings of the 2008 ACM symposium on Applied computing*. New York, NY, USA: ACM, 2008, pp. 656–657.

[27] D. Ferreira and A. Silva, "Survey on System Behavior Specification for Extending ProjectIT-RSL," in *Proc. of the 7th Int. Conf. on the Quality of Information and Communications Technology (QUATIC 2010)*. Los Alamitos, CA, USA: IEEE Computer Society, September 2010, pp. 210–215, ISBN: 978-0-7695-4241-6.

[28] A. Cockburn, *Writing Effective Use Cases*, 1st ed. Addison Wesley, October 2000, iSBN-13: 978-0201702255.

[29] S. Adolph, P. Bramble, A. Cockburn, and A. Pols, *Patterns for Effective Use Cases*, 1st ed. Addison Wesley, August 2002, iSBN-13: 978-0201721843.

[30] D. Rosenberg and M. Stephens, *Use Case Driven Object Modeling with UML: Theory and Practice*, 1st ed. Apress, January 2007, iSBN-13: 978-1590597743.

[31] C. Borysowich, "Guidelines for Pseudocode in Documenting Specifications," Retrieved Friday 18th February, 2011 from http://it.toolbox.com/blogs/enterprise-solutions/guidelines-for-pseudocode-in-documenting-specifications-16011.

[32] S. Bird, E. Klein, and E. Loper, *Natural Language Processing with Python*, 1st ed. O'Reilly Media, June 2009, ISBN-13: 978-0596516499.

Applying and Validating a UML Metamodel for the Requirements Analysis in Multi-Agent Systems: The AME-A Case Study

Gilleanes Thorwald Araujo Guedes, Rosa Maria Vicari
Instituto de Informática - Programa de Pós-Graduação em Computação (PPGC)
Universidade Federal do Rio Grande do Sul (UFRGS) - Porto Alegre - RS – Brasil
gtaguedes@inf.ufrgs.br, rosa@inf.ufrgs.br

***Abstract*—This paper presents the application and validation of a UML metamodel developed for the requirements modeling in multi-agent systems projects. In this paper we describe the metamodel and its application into the modeling of an intelligent tutor system called AME-A. We also demonstrate the way we intend to validate this metamodel.**

Keywords-Metamodels; Stereotypes; Agents; Agent Roles; AgentRole_Actors; Internal Use-Cases; AME-A

I. INTRODUCTION

AOSE (Agent-Oriented Software Engineering) is a new area that mixes concepts both from the software engineering and artificial intelligence areas. This new area arose from the multi-agents systems (MAS) development increase and from the new challenges relative to the analysis and project generated by this kind of software. The AOSE seeks to develop methodoologies, techniques, and languages that allow the modeling of the special features presented by the MAS.

UML (Unified Modeling Language) is a standard modeling language adopted by the software engineering area, thus, within the AOSE context, some attempts to create new languages after the UML as adapted to the Multi-agent Systems (MAS) project were made. However, none of the languages studied was concerned about extending or adding new metaclasses to the metamodel that serves as the basis for the use-cases diagram (UCD), for that is mainly employed in the requirements analysis (RA) an essential step for the achievement of a good software project.

Considering the importance of the requirements analysis phase, we developed a UML metamodel for the requirements modeling in MAS projects. In this metamodel we created new metaclasses and stereotypes, thus to allow the employment of the use-cases diagram for this purpose.

In this paper we shall discuss the UML-derived languages for the MAS project we have studied. Next we shall speak about the RA importance, passing on to the definition of metamodels, models, and profiles. After that, we shall describe the developed metamodel and present an application of it. Finally, we shall discuss the validation of the metamodel, when we shall compare the diagrams produced using our metamodel concepts with the diagrams produced by means of the standard UML, thus trying to demonstrate that the UML does not support the modeling of the concepts that our metamodel does.

II. DERIVED LANGUAGES FROM UML FOR THE MULTI-AGENT SYSTEMS PROJECT

Some languages were derived from UML to adapt it to the MAS Project, like AUML [1], AORML [2], AML [3], MAS-ML [4], and the [5] agency metamodel. However, there were found no mechanisms in these languages to model the requirements for the MAS project, nor any concern into making a RA as in the system life cycles described by [6] or in software development methodologies like the UP [7].

One of AUML [1] main contributions was the adaptation of the UML sequence diagram to the agent communication. However, there has been no adaptation of the UCD for it to be applied into the RA. We imagined that, as the AUML is based in UML, the UCD would be applied in its original way; however, were this the case, this diagram could not be used to model the internal requirements of a MAS.

The AORML concentrates on the matter of interaction modeling between agents and how these react to stimuli. With respect to the requirements modeling, [2] proposes, but does not show, the use of the activity diagram for the RA. In [8] the UCD is applied in its original format, identifying only external requirements; no adaptation of this diagram was attempted to model agent roles, goals, perceptions, actions or plans.

The AML represents agents and the roles played by them and it allows to dynamically model role changes. AML further represents the agents' perceptions and actions. [3] mentions it is possible to apply mental states so as to enrich the use case modeling by means of the requirements modeling based on goals; however we did not find examples of how that could be done. Besides, no metaclasses were created to model requirements in MAS as proposed in our work.

MAS-ML [4] represents agents, their environments, the resources the agents can manipulate, the organizations of the system and the roles that can be played by the agents, as well as their beliefs, goals, and plans. However, the language is not concerned about the matter of the requirements analysis and no attempt was made to adapt the UCD to this end.

The metamodel in [5] represents roles, organizations, environments, beliefs, goals, and plans, as well as the agents communication. However, this metamodel does not focus on the requirements modeling and no attempt was made to adapt the UCD for this function.

III. Requirements Analysis

According to [9], about half of the factors associated with successful projects is related to requirements and the project success is directly bound to its quality. Now [10] states that the incorrect requirements definition is one of the mainly factors contributing for the failure of a project. And, according to [11], the requirements elicitation, analysis and documentation in complex systems is a crucial and non-trivial task. Finally, according to [6], use-cases (UC) are of special interest in the requirements engineering, since they were demonstrated as valuable to elicit, document, and analyze requirements.

Within this context, several authors recommend the UML UCD for the requirements modeling, like [6], [9] and [11]. The objective of this diagram, as can be seen in [12] is that of identifying the actors that might interact with the system, the functionalities offered by the software and which actors will be allowed to use which functionalities. Being a standard language widely accepted and used, with great flexibility for adaptation to new dominions we believe that UML notation use can easen understanding by developers. Thus we developed a UML metamodel in which we created new metaclasses and stereotypes with the goal of allowing the employment of the UCD for the requirements modeling in the MAS project.

IV. UML, Metamodels, Models and Stereotypes

UML is a visual language for specifying, constructing, and documenting system artifacts. It is a general-purpose modeling language that can be applied to all application domains [13], The UML specification is defined by using a metamodeling approach. When metamodeling, a distinction is established between metamodels and models. A metamodel defines a semantics for the way of modeling elements within a model. A model captures a vision of a physical system, it is an abstraction of the software with a purpose that determines what must be included and what is irrelevant.

A stereotype is a limited type of metaclass that can only be used in conjunction with one of the metaclasses it is extending. A stereotype allows to define how a metaclass can be extended, assigning to it new characteristics and/or constraints. When a new metaclass is derived from a previous metaclass and uses the stereotype named, “stereotype”, this new metaclass allows to model instances of the original metaclass containing a stereotype with the new metaclass name, making possible to model stereotyped elements in a model with characteristics that differentiate them from the original elements.

V. The Proposed UML Metamodel

Considering that UML is a widely accepted language and that MAS has its own characteristics and that none of the studied languages that extended the UML to the MAS project focused on the RA issue, we created a metamodel containing new metaclasses and stereotypes prepared to identify the requirements of this kind of system.

Initially, we tried to create a UML profile to adapt the UCD, but we realized that only extending the metaclasses used by this diagram would not be enough. Previous versions of the original profile have been published in [14], [15] and [16]; nonetheless, we evolved it into a real metamodel, adding new metaclasses and stereotypes and adapting it to the work by [17]; besides, in the previous publications we did not speak about how we intend to validate this metamodel.

Initially, we tried to use the Actor metaclass to model agent roles, because, according to [12], this metaclass represents a kind of role played by an entity that interacts with the system, but is external to it. However, the agents are not external to the system, they are inserted within the system and, as they are independent, proactive, and able to interact with the software according to their goals, the roles of these agents must be represented as actors. Therefore, as states [18], it is necessary to adapt this concept, since agents are internal to the system, and if we intend to represent agent roles as actors, these actors must be represented internally, inside the system boundaries.

The representation of agents/roles as UML actors can also be seen in [19], where agents represent active objects and are modeled within the system as actors with square heads. However, [18] only suggests that the actor concept should be adapted, all the while [19] did not create a UML metamodel. In our work we explicitly created new metaclasses to allow for an adequate requirements modeling to be employed in the MAS.

All the same, according to [13], it is not possible to take away any constraints once applied to a metamodel in a profile. Thusly, instead of extending the Actor metaclass, we created a new metaclass from the same metaclass as was created the metaclass Actor, as can be seen in the Fig. 1, thus creating the metaclass AgentRole_Actor. In this new metaclass we copied all the characteristics of the metaclass Actor, suppressing only the constraint that an actor must be external to the system.

From this new metaclass we derived the Reactive and Cognitive_AgentRole metaclasses. We applied the stereotype named, “stereotype”, to both metaclasses, which means that these metaclasses will be applied as stereotypes upon AgentRole actors and will assign them special characteristics.

We specialized still further the Cognitive_AgentRole metaclass by deriving the PS, UAM and SMI_AgentRole metaclasses. They were created after the work of [17], who suggested the use of specialist agents Problem Solving (PS), that must own knowledge about the problem the application will help to solve; Users and Agents Modeling (UAM), that must own the knowledge about the users and agents modeling in order to make cognitive models; and Social Mediated Interactions (SMI), that must own knowledge about how to mediate social interactions between the user and the system.

According to [12], UC can be used to specify the system's external requirements. A UC is the specification of a set of actions performed by software, generating an observable result and producing some value for one or more actors. We intended to represent actions, perceptions, goals, and plans as UC, but the UseCase metaclass semantic states that UC represents external requirements, thus, with the objective of adapting the UC concept for the MAS modeling, we derived a new metaclass from the same metaclass as the UseCase metaclass had been derived, calling it InternalUseCase (IUC) and then we created relationships for this metaclass, equal to those existing between the Classifier metaclass and the UseCase metaclass.

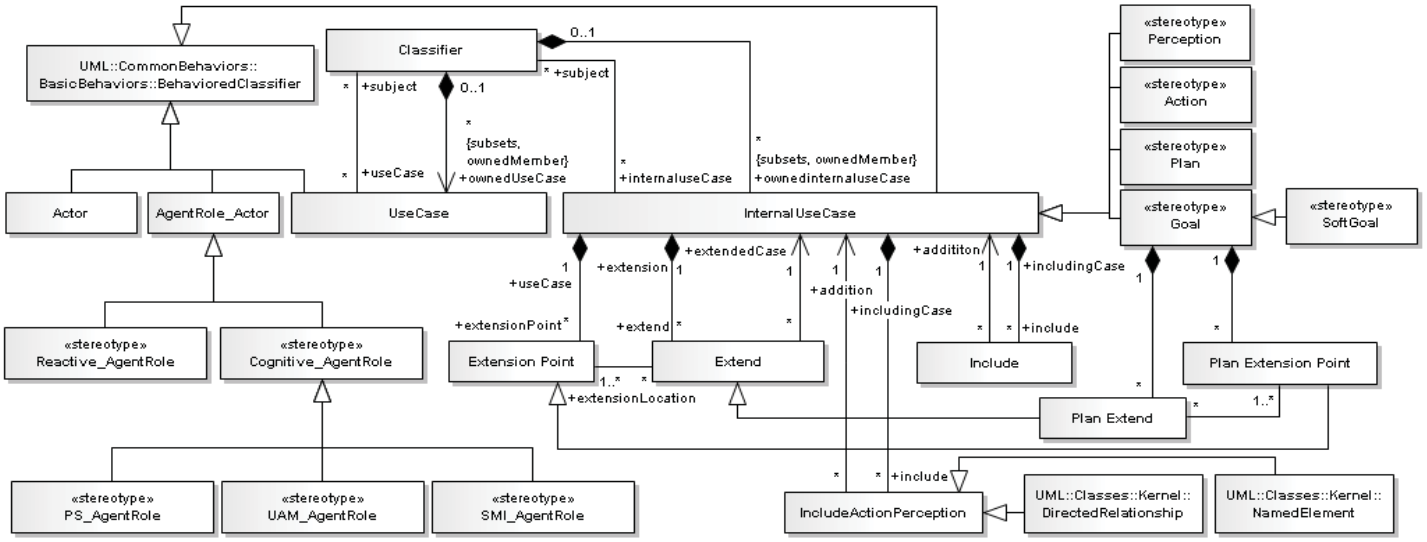


Figure 1. UML Metamodel for Requirements Modeling in the Multi-Agent Systems Project

All IUCs are internal to the system and cannot be seen or used by external entities. From the InternalUseCase metaclass, we extended some metaclasses to attribute special characteristics to the IUCs. These extended metaclasses will be employed as stereotypes. Thus, we created the Perception and Action metaclasses to model IUCs that contain the necessary steps for an agent to perceive an event or perform an action.

A third metaclass was derived from the InternalUseCase metaclass, to represent goals. An IUC employing the Goal sterotype will contain a description of an agent's desire and the possible conditions for that desire to become an intention. A somewhat similar proposal to represent goals as use cases can be seen in [19]. However, besides creating the IUC concept, we went even further, when we considered that, in the same way a goal represents a desire that will not necessarily became an intention, the steps for its execution should be detailed in one or more IUCs other than the IUC that represents the goal.

So, in a situation where an IUC employing the Goal stereotype was used, we would detail in this IUC only those perceptions and conditions necessary for that goal to become an intention. Considering that a goal can eventually own more than a plan and that these plans only will be executed under certain conditions, we decided to derive a fourth metaclass named Plan, to identify the plans associated to the goals.

According to [12], an extend is a relationship that specifies how and when a behavior defined in the extended use case can be inserted into the behavior of the extending use case. If the extension condition is true, the extended use case behavior will be also performed. Considering that a plan is only triggered after a condition is satisfied, we extended the Plan Extend metaclass from the Extend metaclass and associated it with the Goal metaclass. We proceeded this way to differentiate normal extension associations from plan extension associations.

We also derived the Plan Extension Point metaclass from the Extension Point metaclass. According to [12], an extension point identifies a point in the use case behavior that can be extended by the behavior of another use case. We derived this metaclass to establish a difference between plan extension points and normal extension points.

Finally, we derived the IncludeActionPerception metaclass from the same metaclasses that the Include metaclass was derived. We made it because the semantics of the Include metaclass says that it is intended to be used when there are common parts of the behavior of two or more use cases. Though agent roles can share perceptions and actions it is not a rule, so we choose to create a new metaclass in which we copied the same characteristics of the Include metaclass but we taked off these constraint.

VI. A CASE STUDY - THE AME-A PROJECT

The AME-A architecture [20] is composed by a hybrid society of agents that cooperate into aiding students' learning. The environment interacts with human agents that can be both the teacher or the students and has several reactive and cognitive agent roles. Previous versions of this case study were published in [15] and [16]; however, in these publications we used old versions of the current metamodel that did not support the types of agent roles proposed by [17]

The teacher can create a learning activity or evaluate the students with the aid of the agent who assumes the Teacher's Tools agent role. The student can choose between performing an unmonitored or a monitored learning session. In the first, he only interacts with the agent who takes the Unsupervised Learning agent role that only presents the content to be learned.

The monitored learning activity is the system main focus, in which it aims to maximize the student learning by means of the aid of five cognitive agent roles, to wit: Student Modeling (SM), Methodology and Teaching Plan (MTP), Learning Orientation (LO), Learning Analysis (LA) and Knowledge Application Orienting (KAO). The first models the student profile in a dynamic way, while the second chooses the methodology and teaching plan that are more adequated to the

student profile every time it changes or whenever the student performance is lower than the expected level; the LO agent role selects the contents to be taught and the way how these will be presented according to the methodology; the LA agent role checks on the student performance throughout the session and the KAO agent role applies a evaluation after the session ends.

Considering that teacher and student are both roles assumed by external human agents, we modeled them as normal actors outside the system boundaries and we associated to that actor who represents the teacher the functionalities "Create learning activity" and "Evaluate students", represented by normal use cases. In these functionalities there is also an interaction with the agent role, Teacher Tools, that being a reactive agent role, was modeled as an AgentRole_Actor with the sterotype "Reactive AgentRole" and put within the system boundaries, since it is inserted in the system.

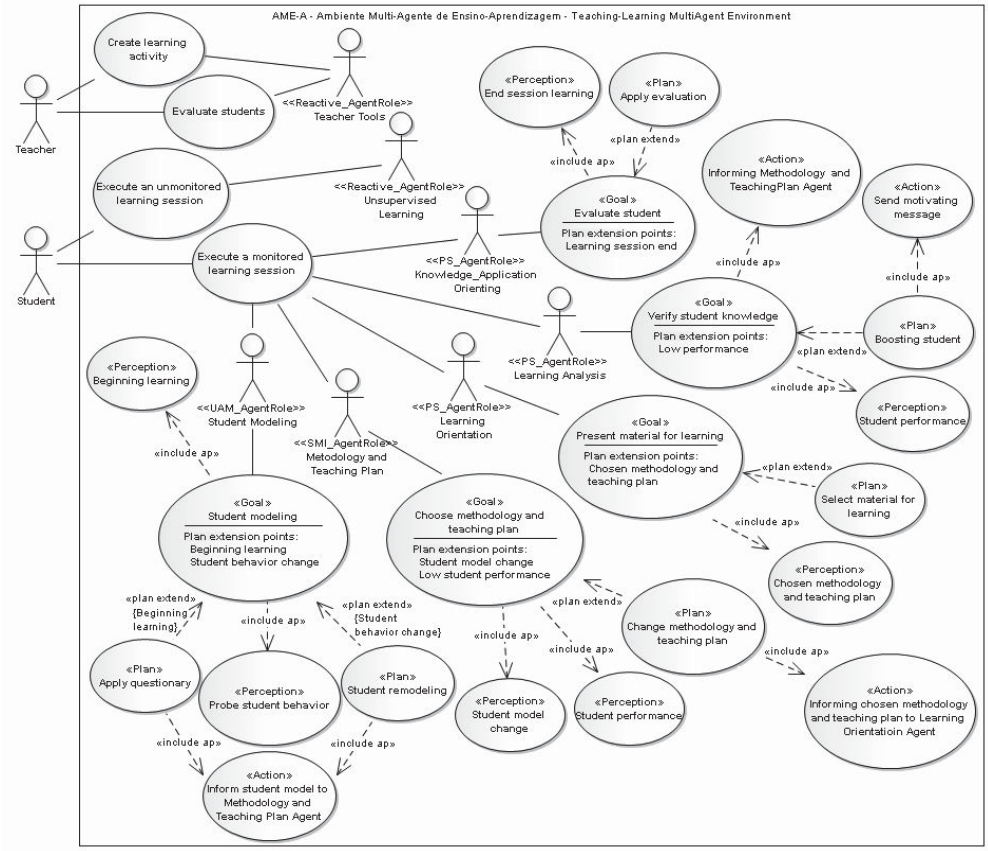


Figure 2. The AMEA Use-Cases Diagram

The actor student was associated to the functionalities, "Execute an unmonitored learning session" and "Execute a monitored learning session"', represented as normal use cases. In the first functionality there is also an interaction with the agent role, Unsupervised Learning, that was represented as an AgentRole_Actor with the stereotype "Reactive_AgentRole".

The functionality, "Execute monitored learning session" involves the five cognitive agent roles, that were classified as PS, UAM and SMI actors, as is demonstrated by their stereotypes in the figure 2. Thus, the SM actor received the UAM_AgentRole stereotype, since it is responsible for the modeling of the students that interact with the system. On its turn the MTP actor received the SMI_AgentRole stereotype, since it is responsible for establishing which methodology will be used with each student. All the other actors received the PS_AgentRole stereotype, because they own knowledge about the problem the application intends to solve.

The SM agent role has for its goal to model the student in a dynamic way. The agent who plays this role needs to perceive when the learning session is beginning and, in this case, trigger the plan, "Apply questionary", to determine the student profile. He also needs to perceive when the student behavior changes, thereby to trigger the plan to remodel the student profile.

We associated the SM agent role to an IUC with the Goal stereotype representing the goal to which the agent who plays this role has to model the student profile. Note that this goal has two inclusion actionperception (IAP) associations (<<include ap>>) with two IUCs that represent the perceptions the agent needs to own to determine whether it is necessary to trigger any of the plans associated with the goal. Thus, to achieve its goal, the SM agent role has to perform mandatorily these perceptions. So, we used an IUC with the Perception stereotype to represent the perception of the learning session beginning and other IUC with the same stereotype to represent the perception of the student behavior.

In the IUC that represents the goal, there are two Plan Extension Points that represent those points in the goal behavior where the plan associated to it can be extended and

besides establish also the conditions for those plans to be performed. So, in the moment the agent perceives the learning session is beginning, it will trigger the plan to apply a questionnaire to the student, represented by an IUC with the stereotype Plan; and, if the agent perceives a change in the student behavior, it will forthwith trigger the plan to remodel the student, equally represented by an IUC with the Plan stereotype. Both IUCs that represent the plans are associated with the IUC Goal by means of plan extend associations, i. e., these IUCs only will be performed when the conditions detailed by the Plan extension points are satisfied. Both plans own an action which is also represented by an IUC with the Action stereotype, and it represents the sending of the student model to the agent who plays the MTP agent role.

On its turn, the MTP agent role has for its goal to choose the methodology and the teaching plan most adequate to the student; to do this, it needs to perceive when the student model changes. This goal is associated to a plan, "Change Learning Methodology and Teaching Plan", that will be triggered whenever the student's model changes or every time the student's performance is proven to be poor. The execution of this plan includes the sending of a message to the agent who plays the LO agent role, to inform the latter that the methodology has been changed.

To model these requirements we associated an IUC with the Goal stereotype with the agent role in order to represent its goal. Next, we associated, by means of IAP associations, two IUCs with the Perception stereotype, to represent the student model change and the student performance perceptions. After that, we created a plan extend association to link the IUC with the Plan stereotype, that represents the plan to change the methodology and the teaching plan, with the IUC Goal. This plan will be triggered only when the student model changes or when the student performance is poor, as is shown by the Plan extension points of the IUC, Goal. Finally, if the plan is triggered, it will be necessary to communicate to the agent who plays the LO agent role the methodology change; as this is done by means of a communication between agents, we identified it as an action and we associated it to the plan by means of an IAP association.

The LO agent role has for its goal to present learning contents for the student and for its perception the choice of the methodology and teaching plan. When the methodology and the teaching plan are perceived, the agent who takes the LO agent role executes the plan "Select material for learning". To model these requirements we represented the goal, the perception, and the plan as IUCs containing respectively the stereotypes, Goal, Perception and Plan. As the plan will only be triggered when the choice of a methodology is perceived, there is an IAP association between the goal and the perception, forcing the agent to verify whether it occurs. There is also an extension association between the goal and the plan, since the plan only will be triggered if the condition is satisfied.

The LA agent role has for its goal to check the knowledge acquired by the student, represented by an IUC containing the Goal stereotype. To execute this goal, the agent who takes this role must perceive the student's performance; this perception is represented by an IUC containing the Perception stereotype. Besides, the agent who assumes the LA agent role must inform this performance to the agent who takes the MTP agent role, which is represented as an IUC containing the Action stereotype. If the student performance is considerated low, then the plan, "Boosting student", is triggered, equally represented as an IUC with the Plan stereotype. This plan has for action to send motivation messages to the student; we identified this as an IUC containing the Action stereotype and we connected this to the plan by means of an IAP association.

Finally, the KAO agent role, has for its goal to evaluate the student after the session ends. Thus, it needs to perceive that the learning session has ended so as to know when to trigger the plan, "Apply evaluation".

VII. VALIDATING THE PROPOSED METAMODEL

With relation to the validation, we compared the models created by means of this metamodel with the models created from the UML original UCD, trying to demonstrate that the proposed metamodel allows the modeling of concepts not supported by the standard UCD. Following this idea, we produced a UCD for the AME-A system using only the standard UML and comparing it with the diagram produced by means of our metamodel, as can be seen in the next figure.

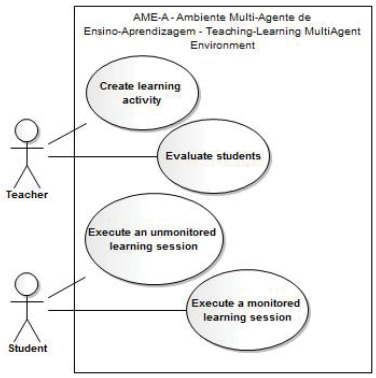


Figure 3. Use-Cases Diagram Using the Standard UML

We did not consider possible to model the goals, plans, perceptions, or actions of these roles, because they are internal functionalities to which the external users do not have access. Besides, the standard UML simply does not have mechanisms to represent goals, plans, perceptions, or actions in a UCD.

Alternatively, using the standard UML, we could try to make the use case "Execute monitored learning session", to encompass the steps of the use cases, "Student modeling", "Choose methodology and teaching plan", "Present material for learning", "Verify student knowledge", and "Evaluate student". What could be done in this specific situation, would be to associate the use cases that represent the plans to the use case, "Execute a monitored learning session", by means of extend associations, establishing the conditions for those use cases to be performed, as demonstrated in the figure 4.

By this approach we tried to use the standard UML in order to achieve the same objective reached by the use of the proposed metamodel. As can be perceived we kept suppressing the representation of agents, as these are internal to the system

and an actor represents entities external to the system.

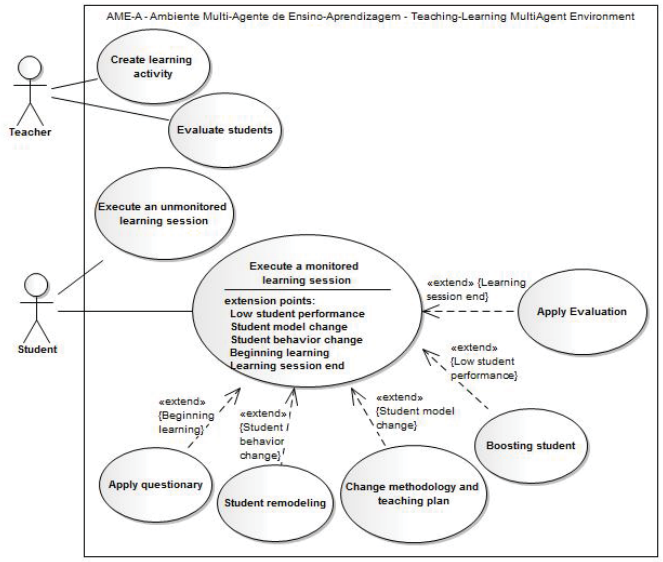


Figure 4. An alternate for the AME-A modeling using extend associations

We tried to associate to the use case, "Execute a monitored learning session", the use cases originally used to represent the plans that should be performed by the agents. Thus, we attributed the plan extension points of the Goal internal use cases to the use case, "Execute a monitored learning session", thus trying to demonstrate under which conditions each "plan" could be performed.

However, these functionalities were not conceived as services to be performed in the "traditional" way by the software but to be divided among cognitive agents, i. e., they are functions separated from the "Execute a monitored learning session" functionality, that represents a service offered to the student, where he will choose, for instance, the learning session theme, but thereafter there will be an interaction with the agents, when those will assume a series of tasks to aid the student and to do it they will need to perceive events, take decisions, and perform actions with respect to which attitudes should be performed. It is not possible to render explicit these kinds of functionalities by using the standard UML.

It is important to note that in reality the use cases associated by means of extensions are not functionalities that the student actor has access to; these functionalities should be performed by agents according to their goals. We further highlight that it is not possible to represent the goals of an agent nor the plans associated to each goal using the standard UML. Likewise, the UML does not support the perception and action representation nor it is able to represent internal agents as actors.

VIII. CONCLUSIONS

We presented a UML metamodel developed for the requirements analysis for the multiagent systems project. This metamodel allows to represent agent roles, besides modeling the perceptions and actions the roles should own, as well as the goals of these roles, together with the plans to achieve them and the conditions for these plans to be performed. We demonstrated the applicability of this metamodel by means of the AME-A intelligent tutor system modeling.

The metamodel in question fills a gap not focused by other approaches that extended the UML language for the MAS project, since none of the studied languages was concerned with addressing the requirements analysis question, a phase of extreme importance for the success of a software project.

With respect to the metamodel validation, we compared the models created by means of this profile with models created from the UML original use-cases diagram; we thus tried to demonstrate that the new proposed metamodel allows to model the concepts not supported by the standard use-cases diagram.

REFERENCES

[1] M. Huget, J. Odell, "Representing Agent Interaction Protocols with Agent UML", The Third International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2004)}, New York, 2004.

[2] G. Wagner, "A UML Profile for External AORModels", Third International Workshop on Agent-Oriented Software Engineering, Bologna, Italy, LNCS, Vol. 2585. Berlin: Springer-Verlag, 2003.

[3] I. Trencansky, "Agent Modeling Language (AML): A Comprehensive Approach to Modeling MAS", Informatica, v 29, n 4, 2005.

[4] V. Silva, R. Choren, C. Lucena, "MAS-ML: A Multi-Agent System Modeling Language", IJAOSE, Interscience Publishers, v.2, no.4, 2008.

[5] C. Silva, "Separating Crosscutting Concerns in Agent Oriented Detailed Design: The Social Patterns Case", Doctoral Thesis, UFPE, 2007.

[6] I. Sommerville, "Software Engineering, 6th Ed", Addison-Wesley, 2000.

[7] J. Arlow, I. Neustadt, "UML 2 and the Unified Process: Practical Object-Oriented Analysis and Design, 2nd Ed", Addison-Wesley, 2005.

[8] K. Taveter, G. Wagner, "Towards Radical Agent-Oriented Software Engineering Processes Based on AOR Modelling", Idea Group, Agent-Oriented Methodologies, ISBN 1-59140-581-5, USA, 2005.

[9] B. Berenbach, Software and Systems Requirements Engineering In Practice, McGraw-Hill, New York, 2009.

[10] E. Hull, K. Jackson, J. Dick, Requirements Engineering - Second Edition, Springer, London Berlin Heidelberg, 2005.

[11] B. Regnell, Requirements Engineering with Use Cases - A Basis for Software Development, Lund University, Sweden, 1999.

[12] OMG - Object Management Group, Unified Modeling Language: Superstructure Specification, v 2.1.1, OMG, 2007. http://www.omg.org.

[13] OMG - Object Management Group, Unified Modeling Language: Infrastructure Specification, v 2.1.1, OMG, 2007, http://www.omg.org.

[14] G. Guedes, R. Vicari, "A UML Profile Oriented to the Requirements Collecting and Analyzing for the Multi-Agent Systems Project", 22nd International Conference on Software Engineering and Knowledge Engineering – SEKE 2010, Redwood, California, USA, 2010.

[15] G. Guedes, "A UML Profile Oriented to the Requirements Modeling in Intelligent Tutoring Systems Projects", The Third IFIP International Conference on Artificial Intelligence, IFIP AI 2010, Brisbane, 2010.

[16] G. Guedes, R. Vicari, "Applying a UML Profile in the Requirements Modeling to Multi-Agents Systems", Quatic 2010, Porto, Portugal, 2010.

[17] R. Vicari, J. Gluz, "An Intelligent Tutoring System (ITS) View on AOSE", IJAOSE, v. 1, n 3-4, 2007.

[18] B. Bauer, J. Odell, "UML 2.0 and Agents: How to Build Agent-based Systems with the new UML Standard", IJEAAI, v 2, i 2, 2005.

[19] S. Flake, C. Geiger, J. Kuster, "Towards UML-based Analysis and Design of Multi-Agent Systems". ENAIS'2001, Dubai, 2001.

[20] C. D'Amico, R. Viccari, "Adapting Teaching Strategies in a Learning Environment" on WWW, WebNet, 1998.

A Panorama of Software Architectures in Game Development

Leonardo Bitencourt Morelli
Elisa Yumi Nakagawa
Dept. of Computer Systems
Institure of Mathematics and Computational Science - ICMC
University of São Paulo - USP
morelli,elisa@icmc.usp.br

***Abstract*—As video games evolve into richer and more sophisticated products, the software driving those games become more complex. One of the research areas offered by Software Engineering to cope with this complexity, while reducing risks and improving software quality, is Software Architecture. The purpose of this paper is to present an overview of possibly all work having investigated, established and used software architectures for the development of video games. For this, a Systematic Mapping was conducted. The achieved results show an increasing, however still mild, interest in the exploration of software architectures for the development of video games, and lays out lines of research that can be explored.**

Keywords- Software Architecture; Game Development; Video Game; Computer Game; Systematic Mapping

I. Introduction

Developing video games has evolved from a one-man job into highly complex projects, executed by large groups of highly specialized professionals [4] over the period of years. This reflects the fact that the video games industry is the fastest growing one in the entertainment market [17], with titles (i.e., individual games) selling millions of copies in the week immediately following each release date [5]. Both the growing complexity and the diversity of target platforms require efforts from the development team to improve software quality and reduce project risks, as well as minimize the cost of a product and make sure it meets today's market standards.

One of the Software Engineering disciplines that can address these efforts is Software Architecture [18], commonly associated with software quality and maintainability. From the first work of Kruchten on iterative software development with a focus on software architecture [10], a number of works has recognized the value of considering software architectures explicitly in system development processes [2], [19]. As already stated by Shaw and Clements [15], software architectures have attained the status of truly successful disciplines. Software architectures play a major role in determining system quality — performance, maintainability, and reliability, for instance — since they form the backbone of any successful software-intensive system [18].

According to Garlan [8] and the SEI [16], a Software Architecture consists of a component structure, the relations between components, and the rules governing the design and evolution of software systems. A well-defined software architecture can improve system reliability, enable the integration of third-party libraries and COTS(Commercial, Off-The-Shelf) components, and promote component reuse, reducing producion costs between projects. Because of this, numerous application domains have been the subject of software architecture studies, and so should video games. It is, therefore, interesting to search for previous works associating software architectures with video game development.

Evidence-Based Software Engineering (EBSE), which is inspired in the medical area and has contributed considerably to software engineering practices, gives us the Systematic Mapping technique. It offers a systematic method for mapping a given research topic in order to obtain a comprehensive overview of the area[13].

This paper's main objective is, through the conduction of a Systematic Mapping Study, to identify possibly all research works investigating the use of software architecture in video game development. While few such publications currently exist, results point out that interest in the area is growing and reveal interesting specific topics that haven't been explored.

This paper is organized as follows. In Section II we present a brief overview on the Systematic Mapping technique. In Section III we present the conducted Systematic Mapping. In Section IV we discuss about achieved results. Finally, in Section V we summarize our contributions and discuss perspectives for further work.

II. A Brief Overview of Systematic Mapping

When a research field reaches its maturity, there is almost always a noticeable increase in the number of reports and results which are made available. At some point in the study of new areas, researchers usually conduct bibliographical reviews (almost always informal ones) to identify publications corresponding to their specific subjects. These reviews are not, however, done systematically, nor is any support given to prevent bias from occurring during the selection of publications set to be analyzed. It is important to have mechanisms to summarize and provide overviews of an area or topic of interest [13].

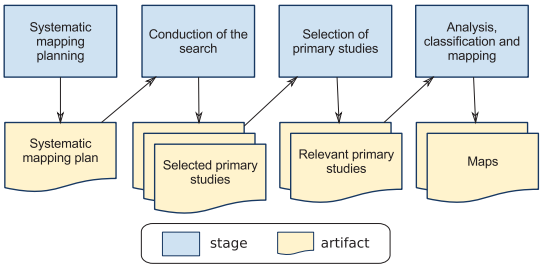


Fig. 1. Systematic mapping process (Adapted from [13])

For this, EBSE has proposed the use of the Systematic Mapping technique [3], [13]. In this context, any individual evidence (e.g., a case study or an experimental study divulged in a publication or paper) which contributes to a systematic mapping is called a *primary study*, while the result of a systematic mapping is a *secondary study*. Systematic mapping aims at providing an overview of a research field, assessing the quantity and types of primary studies existing in the area of interest [13]. In short, systematic mapping is carried out by planning, conduction of search and screening of primary studies by use of inclusion and exclusion criteria [3]. A systematic mapping also includes data extraction and analysis through the identification of categories and classification of primary studies within these categories. As a result, this technique provides maps (i.e., tables and charts) containing condensed information on the area of interest.

III. Conducted Systematic Mapping

This systematic mapping aims at identifying studies describing the use of software architectures in the development of video games, and it was conducted between May and June, 2010, using the process shown in Fig. 1. It is, in short, composed by four stages: (i) systematic mapping planning; (ii) conduction of the search; (iii) selection of the primary studies; and (iv) analysis, classification and mapping. These steps are explained in further detail in the next sections, which present the conducted systematic mapping and explain its execution.

A. Systematic Mapping Planning

This plan consists of formulating the research questions, selecting the sources of primary studies and establishing the inclusion and exclusion criteria.

1) **Research Questions (RQ)**: These questions are structured according to the objective of the systematic mapping. In this case, their purpose is the identification of a panorama featuring both software architectures and video game development. The research questions for our systematic mapping are:
   - RQ1: Which software architecture topics have been studied for the development of video games?
   - RQ2: Who are authors researching the use of software architectures for the development of video games?
   - RQ3: Which software architecture topics for video game development have been given more attention?
   - RQ4: Which video game subsystems have been targeted in software architecture research?
2) **Selection of Sources**: As the sources of primary studies, the databases indexing the most important publications in software engineering were selected: IEEE Xplore[1], ACM Digital Library[2] Springer Link[3], Scopus[4], Science Direct[5] and ISI Web of Knowledge [6]. Only papers written in English were considered in the systematic mapping, since English is the most widely adopted language in the publication of scientific papers.
3) **Establishment of Selection Criteria**: The definition of Inclusion Criteria (IC) and Exclusion Criteria (EC) is another important element in the planning of systematic mapping. These criteria make it possible to include primary studies that are relevant for the research questions and to exclude studies that do not answer them. The inclusion criteria in our systematic mapping are:
   - IC1: The study describes the investigation of a software architecture for video games;
   - IC2: The study describes the investigation of a reference architecture for video games;
   - IC3: The study describes the investigation of a framework for video games;

   The exclusion criteria established are:
   - EC1: The study is not in English;
   - EC2: The study does not have an abstract;
   - EC3: The study is in abstract-only format.
   - EC4: The study does not describe the investigation of a software architecture.
   - EC5: The study does not describe an investigation about video game development.
   - EC6: The study is a copy or older version of another considered study.

B. Conduction of the Search

At this stage, the search for primary studies is conducted according to the previously established plan, i.e., by searching for all primary studies matching a search string in the databases selected as sources. This can be done automatically if the sources provide an efficient search engine. For this, useful keywords are selected and search strings are created.

1) **Keywords**: To keep studies within the systematic mapping's scope, the keywords must be simple and well-chosen. They must be sufficiently simple to produce an adequate number of results, but also rigorous enough to avoid the inclusion of undesired studies. The keywords

[1]http://ieeexplore.ieee.org/
[2]http://portal.acm.org/
[3]http://www.springerlink.com/
[4]http://www.scopus.com/
[5]http://www.sciencedirect.com/
[6]http://www.isiknowledge.com/

chosen for this systematic mapping, separated by area, are shown in Table I.

TABLE I
KEYWORDS SEPARATED BY AREA

Area	Keyword
Software Architecture	"software architecture"
Video Games	"video game", "computer game"

2) **Search Strings**: Using the keywords, the search string is built with AND/OR operators in a way that it represents the correct search parameters. The search string used in our systematic mapping was:
("software architecture" OR "software architectures") AND
(("video game" OR "video games") OR ("computer game"
OR "computer games"))
This search string was accepted by all selected databases, so no custom strings were necessary. With the Springer Link search engine, application of filters after the search was required. The following two filters were used: (i) in "*Subject*" the filter "*Computer Science*"; and in "*Subject*" the filter "*Software Engineering*". With the ISI Web of Knowledge database, the string was used in "topic or title".

This stage resulted in a total of 276 primary studies: 70 from ACM digital library, 12 from IEEE Xplore, 77 from Springer Link, 95 from Science Direct, 6 from Scopus and 16 from ISI Web of Knowledge.

C. Selection of Primary Studies

At this stage, the inclusion and exclusion criteria were employed to select the relevant primary studies.

Table II summarizes the number of primary studies obtained after applying the inclusion and exclusion criteria. To give the databases a fair comparison, this table does not take **EC6** (which excludes duplicated studies) into account. From a total of 276 primary studies previously identified, 33 studies (i.e. 12 %) were considered relevant and thus selected. ISI Web of Knowledge, Scopus and IEEE Xplore were very useful sources for this study: They had high inclusion rates, even though the number of returned studies was low. Ten of the studies returned by Springer Link were included and, despite the low inclusion rate, it should also be considered relevant. Most of the excluded studies didn't cover software architectures or video games at all.

TABLE II
PARTIAL AND TOTAL AMOUNTS OF PRIMARY STUDIES INCLUDED AND EXCLUDED, EXCEPT BY **EC6**

	incl	excl	total	% incl
IEEE Xplore	4	8	12	33.3%
ACM Digital Library	3	67	70	4.3%
Springer Link	10	67	77	13%
Science Direct	4	91	95	4.2%
Scopus	3	3	6	50%
ISI Web of Knowledge	9	7	16	56.3%
total	33	243	276	12%

For the remainder of the Systematic Mapping, **EC6** is also applied, excluding duplicated results from the relevant primary studies. If there were two revisions of the same primary study, the oldest was discarded. The updated relevant primary studies are shown in Table III.

TABLE III
PARTIAL AND TOTAL AMOUNTS OF PRIMARY STUDIES INCLUDED AND EXCLUDED, INCLUDING BY **EC6**

	included	excluded	total	% included
total	31	245	276	11.2%

D. Analysis, Classification and Mapping

While the previous section presented a quantitative analysis of the systematic mapping, this section categorizes the primary studies. For this, two tasks were conducted: (i) search by keywords; and (ii) grouping of the primary studies into categories. To start with, primary studies had their titles, keywords and abstracts carefully read. Following this procedure, terms that seemed to be more relevant were selected as keywords, and these keywords are divided into categories. For this systematic mapping, there was an interest in identifying relationships between software architecture and the development of video games. Thus, four categories, each with a group of keywords, were created:

Category 1: Video game subsystems: This category shows which video game subsystems software achitectures have been proposed for. Fig. 2 illustrates these subsystems and their relationships in a common game system. The subsystems considered here are the result of empirical observation, both by the authors and by the game development community [12]. The keywords for this category are:

- *graphics*: Studies that describe the subsystem in charge of graphics rendering and display;
- *audio*: Studies that describe the subsystem in charge of audio, e.g., music and sound effects;
- *input*: Studies that describe the subsystem in charge of capturing user input from devices, e.g., controllers, mouse and camera;
- *network*: Studies that describe the subsystem in charge of network communications;
- *physics*: Studies that describe the subsystem in charge of physics simulations. This may include the use of a dedicated physics processor or GPU (Graphics Processing Unit) processing;
- *AI*: Studies that describe the subsystem in charge of Artificial Intelligence processing. This may include scripting, strategic/behavioural (i.e., how entities decide what actions to execute) and tactical (i.e., how entities decide how to execute actions) logic, some of which may be title-specific;
- *control*: Studies that describe the subsystem in charge of orchestrating the execution of the other subsystems. It includes the game loop and other title-specific logic; and
- *hardware*: Studies that describe an abstraction layer between the other subsystems and hardware.

Category 2: Software architecture topics: This category shows which topics of the software architecture domain have been more applied in the development of video games. Keywords in this category are:

- *product line*: Studies that present Software Product Lines (SPLs) for the development of video games. SPLs were born as an approach to develop a diversity of software products and software-intensive systems at lower costs, in shorter time, and with higher quality [14];
- *reference architecture*: Studies that propose reference architectures for the development of video game software. Reference architectures have emerged as an element that aggregates knowledge of a specific domain by means of activities and their relations. They can promote reuse of design expertise by achieving solid, well-recognized understanding of a specific domain[1];
- *framework*: Studies that describe software frameworks, which are, in short, abstracted application layers containing common code that provide generic functionality for a domain [6];
- *design pattern*: Studies that describe the use of Software Design Patterns in the development of video games. Design patterns represent reusable solutions to recurring problems in software design [7];
- *aspect*: Studies that describe the use of Aspect-Oriented Programming (AOP) for the development of video game software. AOP has arisen as a new technology to support a better SoC (Separation of Concerns) and more adequately reflects the way developers think about the system [9]. Essentially, AOP introduces a unit of modular implementation — the aspect — which has been typically used to encapsulate crosscutting concerns in software systems (i.e., concerns that are spread across or tangled with other concerns). Modularity, maintainability, and ease to write software can be achieved with AOP [11];
- *COTS*: Studies that describe the use of COTS components that can be used as libraries, tools or building blocks for the development of video games; and
- *quality*: Studies that explore the use of software architectures aiming at improving the quality of video game software.

Category 3: Research environment: This category shows whether the studies come from the industry or the academia. Keywords in this category are:

- *academia*: Studies that were conducted and published by the academia; and
- *industry*: Studies that were conducted and published by the game industry.

Category 4: Evaluation and use level: This category shows how mature the presented software architecture is and how, if at all, it was used.

- *in use*: Studies presenting the use of software architectures that were or are in use in a video game product.
- *prototype*: Studies presenting the use of software architectures that were validated through the implementation of a prototype.
- *study only*: Studies that propose software architectures for the video game domain and that presented the use of neither a prototype nor a product.

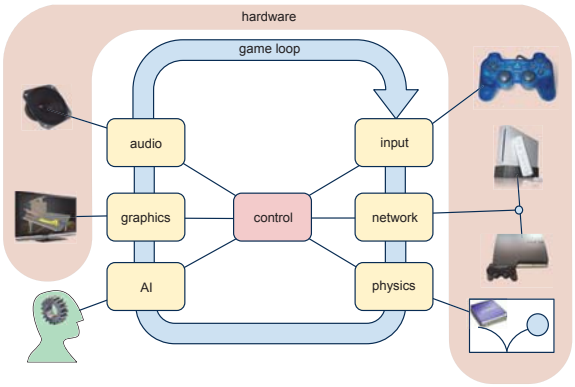


Fig. 2. Game loop and subsystems in a common game system (adapted from [12])

Following the creation of those four categories, the primary studies were classified according to topics within them. It is important to notice that a study may fit into more than one topic per category, or not be classifiable based on the reading of its abstract, title and study keywords. Additional categories for target platform and game genre were considered but ultimately omitted for brevity and because only a few studies could be classified by those categories.

After the conclusion of these groupings, several maps were built, partitioning the relevant primary studies by categories and by year, and the most relevant of those maps are presented and discussed.

Fig. 3 shows the result for the category "Video game subsystems", organized by year of publication. The map shows an increasing interest in researching software architectures for video games, with the substantial rise in 2009 hinting at a larger number of future studies. A constant amount of AI research over time can also be identified, as well as an increasing interest in the control, graphics, input and network subsystems. The augmented interest in the control subsystem can be linked to the newer multi-processed target systems. The appearance of the input and network subsystems can also be explained by the emergence of new technologies, such as non-conventional input systems (e.g., motion and biometric sensors) for the first, and ubiquitous networks for the latter. It is observable, therefore, that there is a tendency towards the use of software architectures to improve each of the subsystems that make up a video game. Considering how software architectures can affect the quality of software products, however, more research work can greatly contribute towards architecturally organizing the video game subsystems and improving the maturity of future video game software.

Figure 4 shows the category "Software architecture topics", by year of publication. This map shows an increase in research in the last few years, and gives us a recurring software archi-

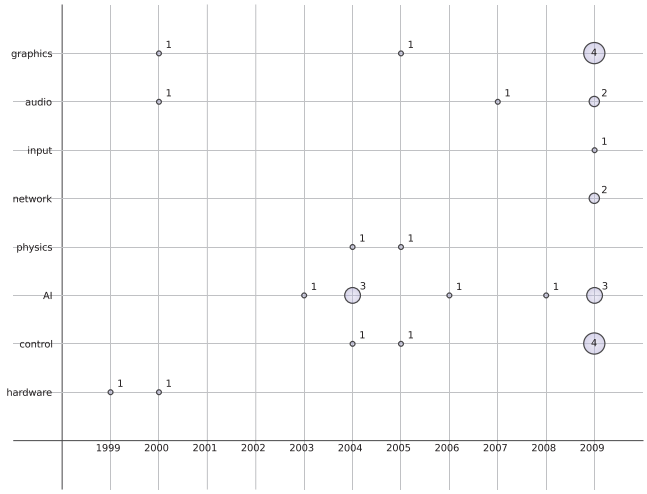


Fig. 3. Map for the category “Video game subsystems”, by year

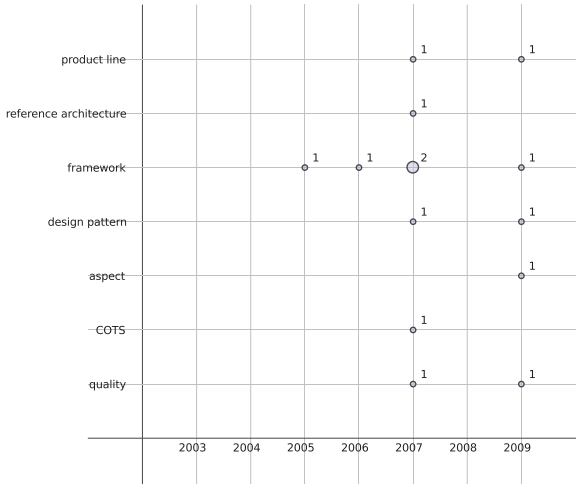


Fig. 4. Map for the category “Software architecture topics”, by year

tecture topic: Frameworks, which are straightforward enough to define and develop, and help streamline the development of video games by keeping all the common problems between different titles treated and solved in a single software package. All topics do, however, remain fairly unexplored in the context of video game development, especially aspect-oriented architectures, COTS usage and reference architectures. The study of validated architectural solutions to problems specific to a game platform or genre, for example, could give birth to reference architectures for that particular platform or genre, which could drastically reduce development time and improve product quality. The study of aspect-oriented architectures for video game development, on the other hand, could help developers model cross-cutting concerns. There is a need for discussions evaluating the use of aspects in game development, be it positively, e.g., identifying important cross-cutting concerns, or negatively, e.g., in respect to program performance or the lack of justifiable reason for the use of aspects.

Fig. 5 condenses two maps. The first crosses two categories, “Research environment” and “Video game subsystems”, and shows that the great majority of studies comes from the

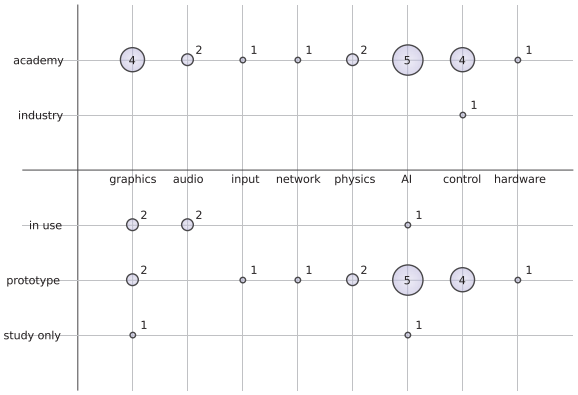


Fig. 5. Map for the categories “Research environment” and “Evaluation and use level”, by “Video game subsystems”

academia. These academic studies focus mainly on the AI, control and graphics subsystems. We have found the lack of publications originated in the industry disturbing, since game developers and software engineers alike could greatly benefit from the knowledge of how real-world problems are solved in production environments. A deeper, more detailed look into the knowledge created by the industry could greatly improve our perception of how video games evolve and how they influence computing and engineering in general, as well as contribute new ideas on how to solve complex architectural problems in software.

The second map in Fig. 5 crosses the categories “Evaluation and use level” and “Video game subsystems”, showing that most investigations involve the development of prototypes. The studies resulting in “in-use” products are about the AI, audio and graphics subsystems, and studies that result neither in prototypes nor in products are related to the AI and graphics subsystems. Although the evaluation by prototype is better than no evaluation at all, the small number of “in-use” studies can be a symptom that either the solutions proposed are not practical, or that the studies are not being taken seriously enough.

In regards to **RQ2**, which asks who are the authors researching the use of software architectures in video game development, no author appeared in more than one of the relevant primary studies. Therefore, it is observed that there is a lack of research groups investigating and publishing work about software architectures in the context of video game development.

IV. Discussion

The use of systematic mapping to elicit previous and current research on the use of software architectures for video game development has uncovered evidence of the growing interest in this subject. This section includes the discussion of a few issues, before a presentation of limitations and lessons learned.

During Section III, all research questions established for our systematic mapping were successfully answered. This suggests that the general knowledge of software architectures in video game development has been mapped. It is also believed that

the results presented in this work are representative of the whole video game development domain, since the systematic mapping technique provides mechanisms and results that allow us to make such statement.

Considering the information gathered throughout this work, it is possible to identify interesting and important research lines that can be investigated in future work.

Current software engineering trends point towards the use of design patterns, COTS components, product line engineering, reference and aspect-oriented architectures to further reduce production costs, project risks and help build products with better quality. These topics do, however, deserve more attention from the game development community if a consensus on their value for developing video games is to be reached.

From the video game perspective, there is room for research in software architectures for all subsystems, in special the control, graphics, input and network subsystems, which have been increasing in popularity.

Also, the game development community can strongly benefit from a detailed description of industry practices that are tested and already known to be successful in real-world applications.

Regarding the limitations of this work, other categories could be established and, consequently, different related maps could be drawn. Also, new research questions could be asked, targeting specific research topics.

V. Conclusions

This work's main contribution is the presentation of an encompassing view of the application of software architectures in the development of video games. To establish this view, a set of steps provided by the Systematic Mapping technique was systematically applied. It is believed that this overview can contribute to the video game development area, shedding light on the existing domain and presenting new oportunities for investigation.

It is widely accepted that software architectures play a major role in determining system quality. More work bridging software architecture and video game development can reduce the gap between these two areas and both improve the quality of video games and bring new architectural styles and solutions to the software architecture domain.

Motivated by the promising results, we intend — as future work — to conduct Systematic Mapping studies involving more specific topics. Thus, our research group strives towards the achievement of an even better understanding of the intersection between software architectures and video game development.

Acknowledgements

Thanks to Alexandre R. Inforzato for linguistics support. Also to Daniel Feitosa and Lucas Bueno Ruas de Oliveira for guidance with the Systematic Mapping technique.

References

[1] Samuil Angelov, Jos J. Trienekens, and Paul Grefen. Towards a method for the evaluation of reference architectures: Experiences from a case. In *Proceedings of the 2nd European Conference on Software Architecture (ECSA'08)*, pages 225–240, Berlin, Heidelberg, 2008. Springer-Verlag.

[2] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*. The SEI Series in Software Engineering. Addison-Wesley, 2 edition, 2003.

[3] D. Budgen, M. Turner, P. Brereton, and B. Kitchenham. Using mapping studies in software engineering. In *Proceedings of the 20th Annual Meeting of the Psychology of Programming Interest Group (PPIG) 2008*, pages 195–204, Lancaster, United Kingdom, September 2008. Lancaster University.

[4] D. Callele, E. Neufeld, and K. Schneider. Requirements engineering and the creative process in the video game industry. *Requirements Engineering, 2005. Proceedings. 13th IEEE International Conference on*, pages 240 – 250, aug.-2 sept. 2005.

[5] Brian Crecente. Take-two confirms gta's half a billion week. On-line, 2008. http://kotaku.com/5008102/take+two-confirms-gtas-half-a-billion-week, 20/05/2010.

[6] M. F. Fayad and D. C. Schmidt. Object-oriented application frameworks. *Communications of the ACM*, 40(10):32–38, 1997.

[7] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns – Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[8] D. Garlan. Software architecture: a roadmap. In *ICSE '00: Proc. of the Conference on The Future of Software Engineering*, pages 91–101, New York, NY, USA, 2000. ACM Press.

[9] Gregor Kiczales, John Irwin, John Lamping, Jean Marc Loingtier, Cristina Videira Lopes, Chris Maeda, and Anurag Maendhekar. Aspect-Oriented Programming. In Mehmet Akşit and Satoshi Matsuoka, editors, *ECOOP '97: Proceedings of the 11th European Conference on Object-Oriented Programming*, pages 220–242, Jyväskylä, Finland, 1997. Springer-Verlag.

[10] P. Kruchten. An iterative software development process centered on architecture. In *Proc. 4ème Congrès de Génie Logiciel*, pages 369–378, 1991.

[11] R. Laddad. Aspect-oriented programming will improve quality. *IEEE Software*, 20(6):90–91, Nov.-Dec. 2003.

[12] Loki Software, Inc. and John Hall. *Programming Linux Games: Learn to Write the Games Linux People Play*. Linux Journal Press, San Francisco, CA, USA, 2001.

[13] Kai Petersen, Robert Feldt, Shahid Mujtaba, and Michael Mattsson. Systematic mapping studies in software engineering. In *EASE '08: Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering*, pages 1–10, Bari, Italy, 2008. Blekinge Institute of Technology.

[14] Klaus Pohl, Günter Böckle, and Frank van der Linden. *Software Product Line Engineering: Foundations, Principles, and Techniques*. Springer-Verlag, 2005.

[15] Mary Shaw and Paul Clements. The golden age of software architecture. *IEEE Software*, 23(2):31–39, Mar/Apr 2006.

[16] Software Engineering Institute. How do you define software architecture? On-line, 2009. http://www.sei.cmu.edu/architecture/definitions.html, 20/05/2010.

[17] Sizhu Tan and Mingzhi Li. The market structure of the video game industry: A platform perspective. In *2008 International Conference on Service Systems and Service Management*, pages 1 –4, june 2008.

[18] A. I. Wasserman. Towards a discipline of software engineering. *IEEE Software*, 13(6):23–31, November 1996.

[19] Zhang You-Sheng and He Yu-Yun. Architecture-based software process model. *ACM SIGSOFT Software Engineering Notes*, 28(2):1–5, March 2003.

Detecting Architecture Erosion by Design Decision of Architectural Pattern

Lei Zhang, Yanchun Sun*, Hui Song, Franck Chauvel, Hong Mei
Institute of Software, School of Electronics Engineering and Computer Science, Peking University
Key Laboratory of High Confidence Software Technologies, Ministry of Education, Beijing 100871, China
Email: {zhanglei07, sunyc, songhui06, franck.chauvel}@sei.pku.edu.cn, meih@pku.edu.cn

***Abstract*—In the software architecture design, architects usually adopt many classic architectural patterns. However, such important design decisions often fail to be effectively captured in practice. The problem called "architecture erosion" may happen easily. During the design iterations, the latest design may accidentally violate the constraints implied by existing architectural patterns. During the evolution of system, the improper modification may conflict with the original design.**

In this paper, we present an approach for detecting architecture erosion, by capturing the most important design decisions about the adopted architectural patterns. Architects can use a collection of predefined and frequently-used architectural patterns in their design. Our supporting tool can capture what and how the architectural patterns are used, and then generate a set of corresponding OCL code automatically. With these specifications, we can both verify the architecture in design phase and validate the run-time architecture to detect architecture erosion. Finally, we use an illustrative example to demonstrate the feasibility of this approach.

keywords: architectural pattern, architecture erosion, design decision

I. Introduction

Nowadays as the size and complexity of software systems increase, software architecture plays an important role in software development [1]. It is important not just in the development phase, but in every aspect of software development, from the initial conceptualization of the system through requirements to the system's deployment. [2]–[4]. Nevertheless, in current architecture centric development process, the problem called "*architecture erosion"*(or "*decay"*) [5] frequently happens in many projects. On one hand, it is common to see the architecture design of a system and its implementation diverged to a certain extend. On the other hand, when architecture undergoes several evolutions for changed or new added requirements, architecture erosions may also happen.

The key cause of architecture erosion is the loss of the important "*design decisions*" [5], which are taken during the architectural design and embedded in the resulting software architecture implicitly. To address this problem, design decision became an emerging research trend in recent years and it drew increasingly attention in architecture community [6]. Many useful approaches and methods have been proposed. Such as, Archium [7], AREL [8], ArchDesigner [9], AQUA [10], and our own ABC/DD [11]. Most of them provide different methods to capture kinds of design decisions. However, as pointed by van Vliet et al., there are too many design decisions but not all of them are that important [2], and it is expensive to document design decisions. Therefore, we hope to only capture the minimal set of the really important ones.

Adopting architectural patterns[1] is one kind of the most important decision. Architectural pattern expresses a fundamental structural organization or schema for software architecture design [14], and in fact it is incorporated in most software architectures. Since 1994 when Shaw and Garlan classified architectural patterns in their seminal book [15], a number of patterns have been applied in many modern systems. Many are used frequently. For instance, pipe-filter, black-board, multi-layers, model-view-controller, client-server, and so on.

However, architectural pattern does not become the focus in most design decision approaches. In practice, adopted architectural patterns are usually hidden in architecture design without an explicit notation. When these design decisions are lost, it will take time to identify those used patterns. Thus, they might be distorted unintentionally by other patterns or improper modifications Even worse, once those patterns have been programmed, they disappear in the code. It is more difficult to ensure whether they are implemented faithfully.

To address this problem, we present an approach to detect architecture erosion, by capturing the most important design decisions about the adopted architectural patterns. Architects can use a collection of predefined and frequently-used architectural patterns in their design. Our supporting tool can capture what and how the architectural patterns are used, and then generate a set of corresponding OCL code automatically. With these specifications, we can both verify the architecture in design phase and validate the run-time architecture to detect architecture erosion.

The rest of the paper is organized as follows: Section II describes the motivative example; Section III presents our proposed approach of capturing architectural pattern, as well as the process of our approach; then Section IV provides an illustrated example using this approach; and related work are discussed in Section V; in the final Section VI, we discuss some limitations of our approach and future work.

* Corresponding Author

[1] we use the term "architectural pattern" as synonym for "architecture style" as suggested by Hofmeister et al. [12] and Bass et al. [13]

II. Motivative Example

Multi-layer architecture (also referred as *n-layers architecture*) is one of the most common used architectural pattern. When adopting multi-layer pattern, the architect needs to break up a system into several layers which logically grouping components that offers a cohesive set of services. Each layer provides a set of services to the layer above and uses the services of the layer below. The most widespread use of multi-layer pattern is *three-layer architecture* as shown in Fig.1(a). It separates presentation, application processing, and the data management components into three independent layers.

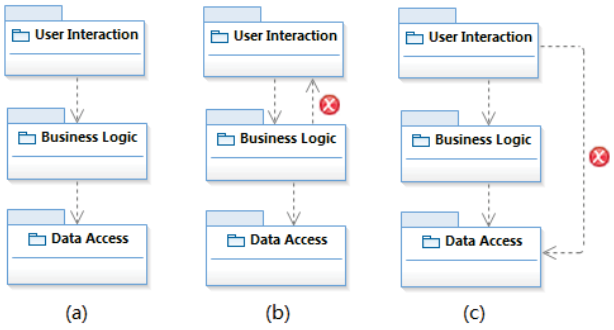


Fig. 1. Multi-layer architectural pattern

Two topological constraints are imposed when adopted multi-layer pattern. Firstly, components in one layer cannot use services provided by the upper layer. Secondly, components cannot use services across the layer. The dashed line marked with error in Fig.1 (b) and (c) illustrate the two violations respectively.

Nevertheless in practice, throughout the entire software life cycle there are some situations in which architecture erosion may occur.

(a) Patterns in Architecture Design

Within one architecture, often several architectural patterns can coexist in the same design. Previously used patterns may conflict with the subsequently used patterns during architecture design iterations.

For example, multi-layer pattern in Fig.1 (a) was adopted already, then *Model-View-Controller (MVC)* pattern was adopted later. The architect wants to separate the components of *model, view*, and *controller* into different layers. Then the interaction between these components maybe conflict with multi-layer pattern, if it is not designed properly.

(b) Patterns in Implementation

The best architecture is worthless if the code does not follow it [1]. However, it is hard to guarantee each design introduced by architects can be faithfully transformed to program source code.

For example, the layered architecture design in Fig.1 can be violated accidentally. A report of a real project said that the key developer of the GUI directly accessed the database without modifying architecture design just like Fig.1 (c) [16]. Even the popular open-source IDE – Eclipse, which adopts multi-layer pattern, has similar layers violations, as discussed in [17].

Similar violations also can happen during the evolution phase of systems. Without an explicit representation of architectural pattern, people often modify the system without a fully understanding of the system. If it evolves in a way that conflicts with the original design, the architectural pattern may be broken.

III. Proposed Approach

A. Software Architecture Meta Model

Based on our previous underlying meta-model for architecture design [11], we extend it to support architectural pattern, as shown in Fig.2.

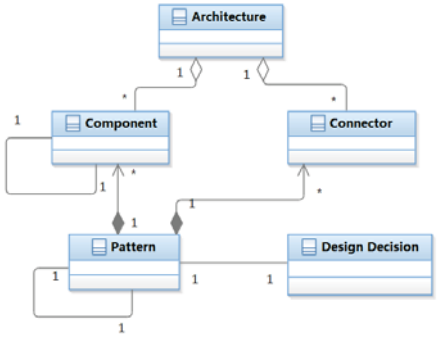


Fig. 2. The Core Meta-Model of the Approach

B. Predefined Architectural Pattern

In order to provide a handily cataloged architectural pattern repository of ready-to-use, we must clearly define what constitute a pattern in advance. According to existing literature, mainly [3], [14], [15], [18], we use the minimal set of constituent elements to characterize an architectural pattern:

(1) Structure

Each architectural pattern has a structure which is constituted by several subsystems. The name of each subsystem provides a vocabulary of design solution. For example, pipe, filter, blackboard, client, server. We use a package diagram to illustrate the structure of a pattern.

(2) Structural Constraints

The constraints define how elements can be integrated to form this pattern and determine allowed interaction between subsystems, e.g., as discussed in the motivative example, components in multi-layers architectural pattern, should not access the components across the layer, as Fig.1 (c). We use natural language to describe these constrains, and use a set of templates of OCL code to precisely describe them.

(3) Known Quality Attributes Impact

Most architectural pattern will affect the quality attributes. Applying a given architectural pattern may make it easier or harder to implement certain quality attributes. Based on existing literature, we collect known impact on quality attributes. Table.1 lists some of the most representative ones.

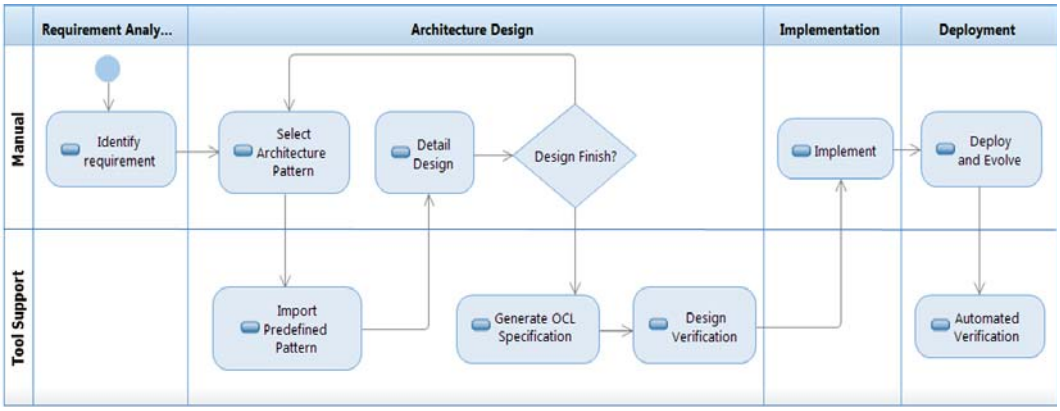


Fig. 3. Process diagram of the approach

C. Process of Proposed Approach

The process of our approach is depicted in Fig.3. There are in all four segments horizontally representing four phases of the software life cycle, including *requirement analysis, architecture design, implementation, deployment/evolution*. In the vertical direction, the diagram is divided into two compartments: the upper one contains manual activities, while the lower one contains the activities finished by our tool.

(a) Requirement Analysis Phase

In the first phase, architects must organize all the architectural significant requirements (ASR) of software system, including both functional and non-functional requirements.

(b) Architecture Design Phase

(b.1) Select Architectural Pattern

By examining the predefined architectural patterns, architects should select a suitable one as a starting point according to identified non-functional requirement and the known quality attributes impact.

(b.2) Import Predefined Architectural Pattern

With the help of our modeling tool, adopting a specific architectural pattern is simple. All that the architect needs to do is to click on the selection from the palette of patterns, then drag and drop to the editing area. By doing this, our tool will automatically place a skeleton of this pattern. For example, Fig.4 shows the case that MVC pattern is adopted. The rounded rectangles labeled with the name of each subsystem are explicit notations of the adopted architectural patterns.

The action of adopting a pattern is an important design decision and it will be captured by our tool automatically in chronological order.

(b.3) Detailed Design

After importing the architectural pattern, architect then can

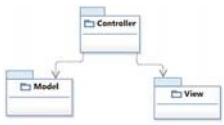
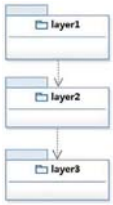
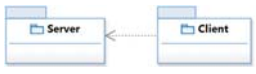
Pattern Name	Pattern Structure	Pattern Element	Structural Constraints	Quality Attributes
Pipe and Filter		A set of Pipes	Pipe is connected with a filter End with a pipe not a filter	+ modifiability + portability + scalability – performance
Model-View-Controller		Model, View, Controller	Components in View and Model should not directly access All the connections should be managed by Controller	+ scalability + extensibility + reusability + evolvability – performance
Layers		Layer	If component resides in layer i , then it can only access the components reside in layer i and i-1.	+ portability + extensibility + reusability – performance – evolvability – reconfigurability
Client-Server		Client, Server	Client should only access the server Client can not access other clients	+ scalability + evolvability – reusability

Table.1. Predefined Architectural Patterns

continue the detailed design within the generated skeleton of pattern. Concrete components and connectors can be filled into the skeleton. The explicit skeleton makes the boundary of architectural pattern more clearly and prevents an ad-hoc architecture design.

(b.4) Generate Architectural Pattern Specification

When the architecture design is finished, the supporting tool will generate a set of OCL code to specify every constraints imposed by existing architectural patterns.

In our predefined architectural pattern repository, pattern constraints are denoted as several templates of OCL code. When generating the executive OCL code, parameters in those templates will be substituted by concrete components and connectors in pattern.

For example, in the *Model-View-Controller* architectural pattern, one constraint is that components of *Model* cannot directly access components of *View*, and all the connections should be managed by components of *Controller*. We denote this constraint as a template of code as shown in Fig.5. The parameters denoted as c_1 to c_3 and r_1 to r_3 represent components and connector respectively, and they will be substituted by concrete components, such as "OnlineShop", "Compare" components in Fig.4.

The reason why we use OCL is because it is a precise language that provides constraint expressions on any MOF model or meta-model [19]. The meta-model that we used is conformed to MOF, therefore we chose OCL to specify pattern constraints.

context MVC_Pattern:
invariant
$c_1.name$ = "controller" ***and*** $c_2.name$
= "view" **and** $c_3.name$ = "model"
and component :: $AllInstances() \rightarrow includes(c_1)$ **and** $r_1.source$
$= c_1$ **and** $r_1.target = c_2$
and connector :: $AllInstances() \rightarrow exists(r_3 | r_3.source =$
c_2 **and** $r_3.target = c_3)$

Fig. 5. Template of OCL Code for MVC Pattern Constraints

(c) Detect Architecture Erosion

In our approach, the generated OCL code can be used to detect architecture erosion in two situations, as discussed in section II. Firstly, we can take a static verification to detect whether there are some conflicts among existing architectural patterns. When a conflict is detected, the tool will warn the architect and highlight the components and connector that are improperly designed in the architectural model.

When system has been implemented and deployed, we can leverage the ability of reflecting the runtime system's architecture [20] to reflect the real architecture. By executing the OCL codes against this architecture, we can judge whether the system still conforms to its original design. When architecture erosion is detected, ABC tool also will prompt a warning and highlight the problematic components and connectors.

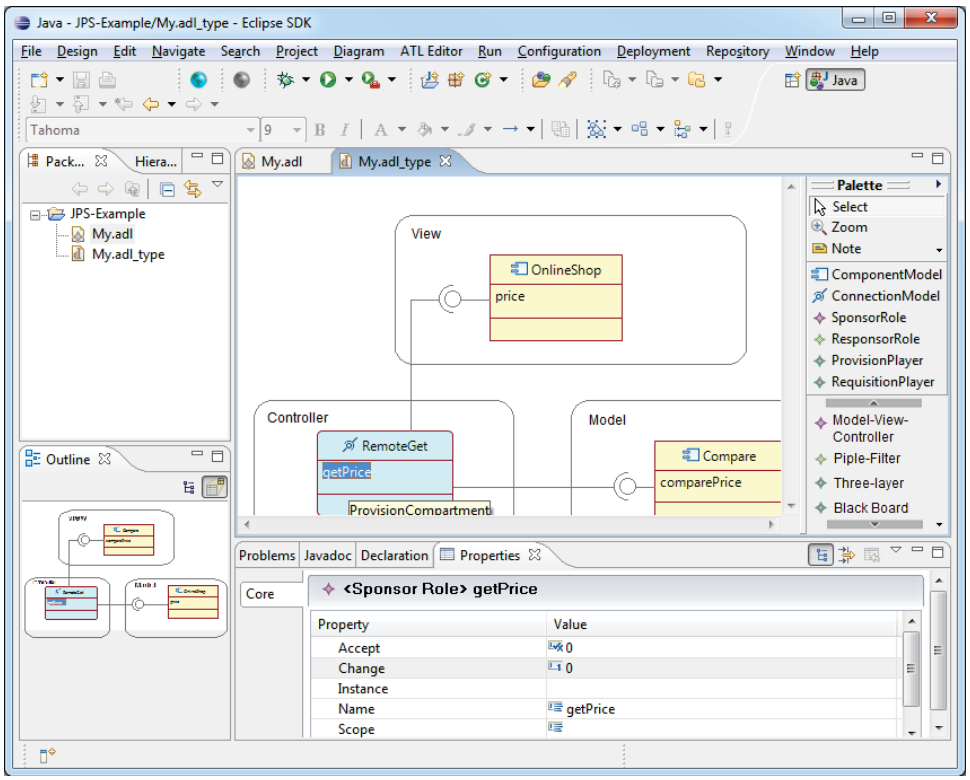


Fig. 4. Screenshot of Supporting Tool

D. Tool Support

Based on our previous architecture modeling tool, ABC (Architecture-Based Component Composition) Tool [21], which we developed as an new plug-in to support the proposed approach. This tool implements an integrated environment that provides support for designing architecture, adopting predefined architectural pattern, capturing design decision and detecting architecture erosion.

ABC Tool is developed as a suite of Eclipse Plug-ins. We mainly use three open source frameworks: EMF (Eclipse Modeling Framework) to model our architecture model, architectural pattern, design decision; and use GMF (Graphical Modeling Framework) to provide visual design ability; and use Eclipse-OCL to parse and execute generated constraints.

IV. Illustrative Example

A. A Simple On-line Voting System

In this section, we use a simple case to illustrate our approach. This is a project selected from a software architecture course for graduate students. Requirements of this on-line voting system includes: (1) provide a user interface for each kind of user; (2) when login, user can view each candidate's information, (3) user can vote for a selected candidate; (4) each user can only vote at most three times; and (5) this system should have high portability.

B. Using the Proposed Approach

One group of students' architecture is presented in Fig.6. In this architecture, there are three adopted architectural patterns. The first one is three-layer pattern; and within the middle layer, there is the second pattern: pipe-and-filter; and the third one is model-view-controller which is scattered in three layers. In total, there are eight components and seven connectors.

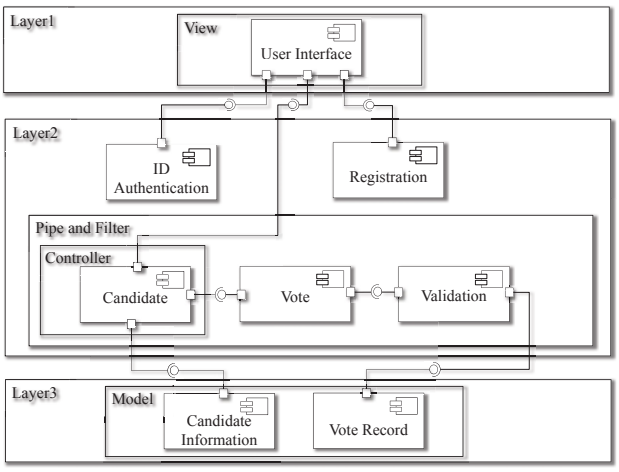


Fig. 6. Architecture of On-line Voting System

In the help of ABC Tool, this group of students used our proposed approach and finished this project. By reviewing this project, we find that ABC Tool detected 2 structural problems in their architecture design and 2 implementation problems during the runtime stage. That is to say, there are 4 potential architecture erosion. We list them in Table.2.

	Architecture Design	Run-time Architecture
Three-Layer	Connect "User Interface" to "Vote Record"	1) "User Interface" directly invokes "Candidate Information" 2) "User Interface" directly invokes "Vote Record"
Pipe-Filter	null	null
Model-View-Controller	Connect "Candidate Information" to "User Interface"	null

Table.2. Detected Architecture Erosion

For example, during the architecture design stage, architect connected "User Interface" to "Vote Record" which broke the constraint of multi-layer pattern; the tool also detected an improper design that connected "User Interface" to "Candidate Information", which violated the constraint of model-view-controller pattern.

V. Related Work

A. Software Architectural Pattern

There are three major approaches used for modeling architectural patterns, including ADL(Architecture Description Language), UML and formal Method.

Most of ADLs are introduced by academia and aim at representing software architectures in general [22]. Some of them directly support architectural pattern description and also leverage formal or semi-formal approaches for the formalization of pattern specifications. For example, ACME [23] uses first-order logic language, Armani, to formally specify architectural pattern. Likewise, Wright ADL uses CSP and ArchWare uses (π-calculus), etc.

UML aims at providing a generic modeling language. There are also some researchers tried to use UML to describe software architectures [24] and architectural patterns [25].

Some researchers proposed to use pure formal methods to describe architectural pattern. For example, [26] uses graph grammars to formal specify pattern. Similarly, [27] also uses some formal methods to analysis architectural pattern. Another comprehensive work [18] presents a semi-formal way using architectural primitives to model architectural patterns. Their work provided a solid foundation for our work.

B. Software Architecture Design Decision

In recent years, there are lots of valuable work in design decision area. Such as Archium, AREL, AQUA, ArchDesigner, ABC/DD, and so on. The most related one is SEURAT [28] that supports the software architects in selecting architectural patterns, design patterns, and idioms for use in architecture. It also captures the rationale for each pattern and the rationale and store them in the pattern library. Compared to our work, although SEURAT records the decisions of adopted pattern and corresponding rationale, it did not provide a visual modeling environment and did not support verification.

VI. Conclusion and Future work

In this paper, we have proposed an architectural pattern-based approach to capture important architecture design decisions. By predefining a collection of architectural patterns, architects can directly select and apply them in their architecture design. Meanwhile our ABC tool can capture these important design decisions, i.e., adopted architectural patterns, and generate a set of OCL code to specify the constraints imposed by those patterns. Then, we can detect architecture erosion in both conceptual design and runtime by executing the generated OCL code.

This paper presented our preliminary work. So far, we have already identified a number of areas that call for further improvements and some of them have become our ongoing work.

The first limitation is that we do not support variants of architectural pattern. Some patterns have several variants to adapt to a specific problem. For example, in the multi-layer pattern, sometimes components in a high layer do need to use components in a very low layer. Such intentional violation of layers constraints is called "*layer bridging*". Similarly, there are many patterns have variants, so we need to improve this problems.

The second limitation is that we do not provide an extension mechanism for architects to add some new architectural patterns. All the predefined patterns in our tool are hard-coded. This problem limits the extendibility of our approach. Since our tool support is based on the Eclipse platform, we plan to take advantages of the meta-modeling capabilities of EMF to support this.

The third limitation is our approach is unavailable to support specifying behavioral aspect of architectural pattern. We plan to extend this work to detect critical behavioral design decisions and their trade-offs.

As part of our future work, we also plan to evaluate the feasibility of our approach with quantitative experiment in a more complex system in real world.

Acknowledgment

This effort is sponsored by the National Basic Research Program of China (973) under Grant No. 2009CB320703, the Science Fund for Creative Research Groups of China under Grant No. 60821003, and the National Natural Science Foundation of China under Grant No. 61073020. The authors want to express their gratitude to the anonymous reviewers for their valuable comments that helped to improve this work.

References

[1] P. Clements and M. Shaw, ""the golden age of software architecture" revisited," *Software, IEEE*, vol. 26, no. 4, pp. 70 –72, 2009.

[2] H. Van Vliet, *Software engineering: principles and practice (3rd Edition)*. John Wiley & Sons, 2008.

[3] P. Clements, F. Bachmann, L. Bass, J. Garlan, G. Ivers, R. Little, P. Merson, R. Nord, and S. J., *Documenting software architectures: views and beyond (2nd Edition)*. Addison-Wesley Professional, 2010.

[4] R. N. Taylor., N. Medvidovic., and E. M. Dashofy, *Software Architecture: Foundations, Theory, and Practice*. John Wiley and Sons, Inc., 2010.

[5] J. Bosch, "Software architecture: The next step," *Software Architecture*, vol. In EWSA, 3047 of LNCS, pp. 194–199, 2004.

[6] M. A. Babar and P. Lago, "Design decisions and design rationale in software architecture," *Journal of Systems and Software*, vol. 82, no. 8, pp. 1195 – 1197, 2009, sI: Architectural Decisions and Rationale.

[7] A. Jansen and J. Bosch, "Software architecture as a set of architectural design decisions," in *Software Architecture, 2005. WICSA 2005. 5th Working IEEE/IFIP Conference on*, 2005.

[8] A. Tang, Y. Jin, and J. Han, "A rationale-based architecture model for design traceability and reasoning," *Journal of Systems and Software*, vol. 80, no. 6, pp. 918–934, 2007.

[9] T. Al-Naeem, I. Gorton, M. Babar, F. Rabhi, and B. Benatallah, "A quality-driven systematic approach for architecting distributed software applications," in *Proceedings of the 27th international conference on Software engineering*. ACM, 2005, pp. 244–253.

[10] H. Choi, Y. Choi, and K. Yeom, "An integrated approach to quality achievement with architectural design decisions," *Journal of Software*, vol. 1, no. 3, p. 40, 2009.

[11] X. Cui, Y. Sun, and H. Mei, "Towards automated solution synthesis and rationale capture in decision-centric architecture design," in *Proceedings of the Seventh Working IEEE/IFIP Conference on Software Architecture (WICSA 2008)*. IEEE Computer Society, 2008, pp. 221–230.

[12] C. Hofmeister, P. Kruchten, R. L. Nord, H. Obbink, A. Ran, and P. America, "A general model of software architecture design derived from five industrial approaches," *Journal of Systems and Software*, vol. 80, no. 1, pp. 106 – 126, 2007.

[13] L. Bass, P. Clements, and R. Kazman, *Software architecture in practice*. Addison-Wesley Professional, 2003.

[14] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *A system of patterns: Pattern-oriented software architecture*. Wiley, 1996.

[15] M. Shaw and D. Garlan, *Software architecture: perspectives on an emerging discipline*. Prentice Hall Englewood Cliffs, NJ, 1996, vol. 123.

[16] M. Stal, "Over the Fence," *http://stal.blogspot.com/2010/03/ over-fence.html*, 2010.

[17] B. Merkle, "Stop the software architecture erosion: building better software systems," in *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*. ACM, 2010, pp. 129–138.

[18] U. Zdun and P. Avgeriou, "A catalog of architectural primitives for modeling architectural patterns," *Information and Software Technology*, vol. 50, no. 9-10, pp. 1003–1034, 2008.

[19] M. Richters and M. Gogolla, "OCL: Syntax, semantics, and tools," *Object Modeling with the OCL*, pp. 447–450, 2002.

[20] H. Song, G. Huang, F. Chauvel, Y. Sun, and H. Mei, "SM@ RT: representing run-time system data as MOF-compliant models," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 2*. ACM, 2010, pp. 303–304.

[21] H. Mei and G. Huang, "ABCTool: a tool for architecture centric engineering of component based systems," in *Companion of the 30th international conference on Software engineering*. ACM, 2008, pp. 957–958.

[22] N. Medvidovic and R. Taylor, "A classification and comparison framework for software architecture description languages," *Software Engineering, IEEE Transactions on*, vol. 26, no. 1, pp. 70–93, 2000.

[23] D. Garlan, R. Monroe, and D. Wile, "Acme: Architectural description of component-based systems," *Foundations of component-based systems*, pp. 47–68, 2000.

[24] M. Bjerkander and C. Kobryn, "Architecting systems with UML 2.0," *Software, IEEE*, vol. 20, no. 4, pp. 57–61, 2003.

[25] N. Medvidovic, D. Rosenblum, D. Redmiles, and J. Robbins, "Modeling software architectures in the Unified Modeling Language," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 11, no. 1, pp. 2–57, 2002.

[26] D. Le Métayer, "Describing software architecture styles using graph grammars," *Software Engineering, IEEE Transactions on*, vol. 24, no. 7, pp. 521–533, 2002.

[27] J. Kim and D. Garlan, "Analyzing architectural styles," *Journal of Systems and Software*, 2010.

[28] W. Wang and J. Burge, "Using rationale to support pattern-based architectural design," in *Proceedings of the 2010 ICSE Workshop on Sharing and Reusing Architectural Knowledge*. ACM, 2010, pp. 1–8.

A Flexible Event-Driven Architecture for Peer-to-Peer Based Applications

Leone Parise Vieira da Silva, Rajiv Geeverghese, Edward de Oliveira Ribeiro,
Genaína Nunes Rodrigues and Célia Ghedini Ralha
Computer Science Department, University of Brasília
POBox 4466, Brasília/DF, ZIP 70.904-970, Brazil
{leone.parise,rajiv.unb,edward.ribeiro}@gmail.com, {genaina,ghedini}@cic.unb.br,

Abstract

Over the last decade, we have seen an increasing interest in the event-driven architecture (EDA) approach. EDA allows the transmission of events among loosely coupled and highly-distributed software components, which is totally adequate to peer-to-peer based applications mainly considering features of scalability and flexibility. In this paper, we present the initial steps towards the design and implementation of a peer-to-peer based EDA, which is intended to reduce the coupling between application layers, and achieve a better distribution of responsibilities between the architecture elements and the environment. Our flexible EDA framework, proposed in this work, inherits important features such as scalability, decentralization and fault-tolerance from the peer-to-peer domain. For a preliminary evaluation of our approach, two experiments using the EDA framework were conducted in different domain applications.

1. Introduction

Event-driven architecture (EDA) is a style of software architecture based on real time flows promoting the production, detection, consumption of, and reaction to events [8]. This architectural pattern may be applied by the design and implementation of applications and systems, which transmit events among loosely coupled and highly-distributed software components and services.

In EDA, events processing may follow three different styles: simple, stream and complex [8]. In simple processing, when a meaningful event happens it initiates downstream actions. Stream processing deals with multiple streams of event data aiming at identifying the meaningful events within those streams. Complex event processing evaluates a confluence of events and then takes action. The events, which may be meaningful or ordinary, may cross event types and occur over a long period of time.

Peer-to-peer (P2P) networks offer an appropriate paradigm for developing efficient distributed systems and applications. In a P2P network, all processes involved in a work or activity have similar roles, interacting cooperatively as peers with no distinction between client and server computers or the infrastructure where they perform [13]. P2P applications are composed by a great number of processes executing on a distributed peer way. The communication pattern among processes is a key point which depends on the application necessities.

Important aspects of P2P applications must be fulfilled through software development frameworks encompassing loose coupling, scalability, decentralization and fault-tolerance skills. The convergence of EDA and P2P is a natural outcome of the recent evolution of distributed systems, since many of the challenging standards issues are closely related. This convergence creates best practices that enable interoperability between networking systems for P2P applications. Therefore, a flexible EDA is essential to improve communication, allowing the exchange of objects between processes in a natural way.

In this work, we propose a flexible EDA for peer-to-peer based applications, keeping the principles of modularity, loose-coupling, well-distribution and P2P technology independence. We implemented a basic version of the proposed EDA, which accommodates the simple event processing style. We evaluate our proposal using two different domain applications, so as to explore distribution through parallelization of the applications tasks. Our results show the easy integration with P2P technologies, while seamlessly preserving EDA principles.

The rest of the paper is organized as follows. The proposed EDA is presented in Section 2, focusing on the architecture (Section 2.1), with the implementation aspects (Section 2.2) and the developed interfaces (Section 2.2.1). The description and the results of the experiments carried out are described in Section 3. Related work is presented in Section 4. Finally, Section 5 concludes the paper and presents our future research directions as well.

2. Our Proposed EDA

The main principle adopted during the definition of the proposed EDA is to divide the development of applications in different layers, providing communication among them through the use of an event bus module and their respective interfaces. This principle allows flexibility, loosely coupling between application layers, better distribution of responsibilities between the architecture elements and the environment, maintaining functionality even when using mobile devices through a TCP network and personal computers in a P2P approach. Together with the P2P applications, other important skills are maintained e.g. scalability, decentralization and fault-tolerance.

The rest of this section presents the proposed EDA architecture, including implementation aspects with the defined interfaces and P2P technologies.

2.1. The Architecture

Our proposed EDA presents four layers: Application, Environment, Network Adapter and Event Bus, as presented in Figure 1. The Application layer is the upper platform layer, which allows the development of different P2P applications, using our EDA framework as a middleware of development.

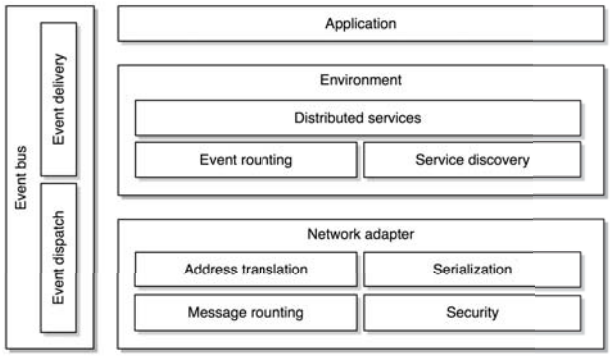


Figure 1. The four-layered architecture.

The Environment layer is composed by the Distributed Services, the Event Routing and the Service Discovery modules. The Environment layer abstracts the modules, allowing the communication and localization of services to be done in a transparent way. In other words, the application issues events in the Environment, which is responsible to notify listeners, without direct communication and connection among different services. In this way, the applications do not know if they are in the same or in different machines. This abstraction feature would help the development of different applications, including software agents, allowing their interaction without the communication bottleneck and independent of the network topology being used.

Apart from offering the described abstraction feature, the Environment layer has an Event Routing and a Service Discovery modules to publish and locate services in a distributed way. Services like distributed database, agent transport, integration of mobile devices can be developed and coupled to the environment transforming it in a more integrated framework. Lots of applications using for example multiagent approach with heavy data, information and knowledge requirements such as bioinformatics, forensics, automated reasoners on the Web may benefit from the features above mentioned.

The Network Adapter is responsible for integrating the Environment to the underlying network, acting as an interface between the P2P middleware and the other EDA framework layers. This layer is composed by the Address Translation, Serialization, Message Routing and Security modules. The integration among the chosen P2P middleware and our EDA framework is done by assigning a unique identifier of each addressable component of the platform, which is represented by the UUID (Universally Unique IDentifier) object of Java API. The P2P middleware implementation uses a table in order to translate these identifiers into other address type, e.g. IP address or P2P network node. In this way, another P2P implementation, such as Chord [17] or Tapestry [10], can be couped to our proposed EDA without modifying other modules.

We should note that, if the chosen P2P middleware has the necessary features of address translation, message routing, serialization and security, then it is not necessary to implement the Network Adapter layer. But, in case it is necessary, it receives the events from the upper platform layers, serializes as a message in the network and sends to the application. Whenever the adapter receives a message, it unserializes as an event, which is sent to the event bus module so that the upper platform layers receive the communication.

Finally, the Event Bus layer is responsible for the Event Dispatch and Event Delivery modules. An event flow starts with the event being generated and culminates with the execution of any downstream activity. The processing of this flow can be described in logical layers. The four logical layers are: event generators, event channel, event processing, downstream event-driven activity [8]. The event processing flow used in our work includes the event generator, event channel, event processing and downstream event-driven activities. The flexibility at each logical layer is the power of event, since the way events are created, how they are transmitted, received and processed is transparent to the P2P applications developed over our proposed EDA.

2.2. Implementation Aspects

The methodology used to define the EDA was based in design patterns with the objective to support its implementation. In order to keep the principles of modularity and loose coupling of event-driven architecture we applied six design patterns, which were the most suitable to the principles of the proposed EDA: $AbstractFactory$, $Singleton$, $Observer$, $Decorator$, $Command$ and $Mediator$ [5]. To validate the proposed EDA, we implemented a prototype in Java at the Computer Science Department of University of Brasília. The prototype has 67 classes and 3,600 lines of documented code.

All the communication in the proposed EDA is done by events, which reduces the object coupling, allowing flexibility and scalability to the architecture. In this sense, the communication has two important elements: the event and the listener, which always come in pairs, since the event carries the object message content and the listener defines which object has to listen specific event.

The Event Bus layer is responsible for the events dispatch and delivery in the platform, which is implemented through three design patterns: $Singleton$, $Observer$ and $Mediator$. $Singleton$ guarantees only one Event Bus instance, while $Observer$ records all listeners and notify them of sent events. The Event Bus layer is also the object mediator in the communication of the platform according to the $Mediator$ pattern. The Event Bus interface is described in Section 2.2.1.

The possibility to execute commands between objects through the Event Bus layer allows the method call between different distributed components in the P2P network. Using $Command$ design pattern we have implemented two special events: $CommandEvent$ and $ResponseEvent$, which inherit from the $Event$ abstract class. Though these events work just like the others, their expected behavior is different. All object that receives a $CommandEvent$ may execute the command and send the results by the $ResponseEvent$. All objects that send a command may wait for an answer during a limited time, and in case the answer does not come, a timeout event is send to the object to notify the expiration time. A method $getId()$ is used in the $Event$ class to identify events.

2.2.1 Interfaces

In order to allow the flexibility and the loosely coupled features of our architecture, considering the application, environment, network and event processing, along with the EDA layers intercommunication, we have developed some interfaces. We present the major interfaces implemented: (i) $Platform$ – interface where all layers and modules are integrated; (ii) $EventManager$ – the event interface that handles the event processing; (iii) $Environment$ – the environment interface to deal with services discovery and distribution; (iv) $NetworkAdapter$ – network interface to deal with the P2P processes.

The $Platform$ interface is the core of our EDA framework. It defines the management of the platform configurations and disaggregates all components (layers and modules) into implementation interfaces, allowing uncoupling and flexibility. This interface follows the $AbstractFactory$ design pattern. Therefore, every component can be reimplemented and replaced without affecting the operation of the other components. The $Platform$ component is also a $Singleton$, which helps and unifies the access of objects among all $Platform$ components, helping the unitary and integration tests.

The $EventManager$ is one of the most important developed interfaces that implements the Event Bus layer, presented in Figure 2. Note that the $EventManager$ interface has two methods: $addlListener()$ – responsible to register listeners and $fireEvent()$ – responsible to communicate an event to a group of listeners. The $EventManager$ interface keeps the reference of all the registered listeners. Whenever the event bus receives an event, it extracts all the objects that inherit the class obtained from the method $getAssociatedListener()$ and notify them in a separate thread. The $EventManager$ implements the $Mediator$ design pattern. Suppose ObjectA wants to communicate to ObjectB, then ObjectA creates a $ConcreteEvent(string)$ and send it to $EventManager$ using the $fireEvent()$ method. The $EventManager$ that has ObjectB registered in the listeners list, communicates the event to ObjectB using $onEvent()$ method.

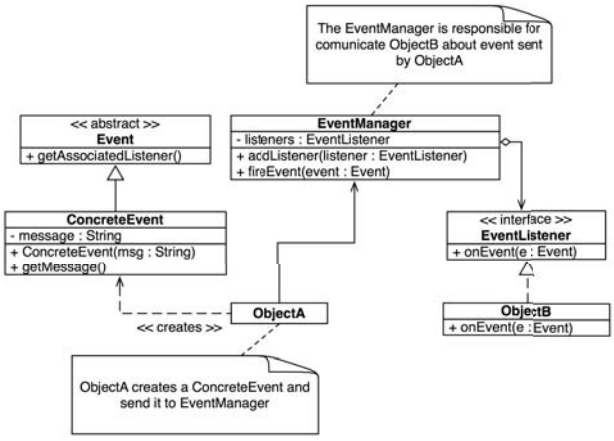


Figure 2. Implementation of the $EventManager$ interface.

The $Environment$ interface defines the environ-

ment in the EDA framework, which has two methods: $requestService()$ and $destroyService()$. The $requestService()$ is responsible to publish and get an instance of the published service.

$NetworkAdapter$ interface deals with the P2P processes. The $NetworkAdapter$ contains two send methods, one to send simple events and the second to send commands and the respective temporary listener to the network. Related to the P2P approach some middlewares were studied to integrate our framework. Important aspects of scalability, fault-tolerance and self-organization were considered, including Chord [17], Can [9], Tapestry [10] and Pastry [12].

We could have used any of the above mentioned P2P middlewares. However, we chose Pastry together with Scribe [2] considering their comprehensiveness. Pastry uses routing mechanisms to achieve great scalability, while Scribe depends on Pastry to route messages to the destinations. Scribe [2] is a typical pub-sub system built on top of Pastry, which leverages its scalability, routing efficiency and self-organization capabilities [15]. The event processing, including dispatch, delivery and routing of events, and the service discovery in our EDA is executed over the Pastry and Scribe mechanisms, making use of the benefits provided by such combination. In this sense, the $NetworkAdapter$ interface is responsible for encapsulating an event through the $PastryEvent$ and send to Pastry.

3. Experiments and Evaluation

During this research project two experiments using the EDA framework prototype were conducted: the turtle and the bioinformatics projects. The objective to use two experiments is to evaluate our framework.

3.1. The Turtle Project

The turtle project is the simple sum of harmonic series, which apparently converge to a number, but is proved mathematically that it varies very slowly. Equation 1 presents the sum of first n elements of the series.

$$H_n = \sum_{k=1}^{n} \frac{1}{k} \quad (1)$$

Table 1 presents the relationship between the lowest number of terms required for the sum of the series is equal to or greater than a specific number. Note that to reach a total value of 1,000 this amounts to 10^{434} terms which is a computational challenge to calculate the sum of the harmonic series term by term, in an iterative fashion.

Our solution was to build a distributed application using the "divide and conquer" and software agents approaches, using the facilities that the EDA framework offers. We

Table 1. Number of n terms sum.

k	least $n \mid H_n \geq k$
1	1
2	4
3	11
5	83
10	12,367
20	272,400,600
100	$\approx 1.51 * 10^{43}$
1,000	$\approx 1.10 * 10^{434}$

used blocks, with the start and size information, and distribute the blocks throughout the network to perform a partial calculation. To implement the distributed solution, two distinct roles become evident: (i) a central figure, to control the distribution of blocks in the network and the receipt of partial calculations; and (ii) a task force, specialized in the processing of the blocks. Each role is implemented by one particular agent, the $BossAgent$ and the $TraineeAgent$ (respectively) both agents have a parent in common, $AbstractAgent$, and inherit their behavior. The $AbstractAgent$ seeks the event services and discovery of services and adds to the platform using the $environment$ interface (Section 2.2.1).

The $BossAgent$ has an internal data structure to track which blocks are being calculated at the time and a variable that stores the next block, starting with the value 1. The block size is fixed, but can be changed before starting the agent. When the $BossAgent$ receives a block request from the $TraineeAgent$, it sends the next block to be processed and waits for the partial sum. Every 5 seconds, the $BossAgent$ uses a log tool to print the partial sum and the number of blocks processed in a file. Once created the agents, the application is started automatically via the exchange of events.

The computational platform used in this test bed (experiment) had eight computers with the same configuration: Pentium 4, 2.66 GHz, 1GB of RAM, and Ethernet 10/100 MB/s. The number of blocks processed with one computer is 209, with two computers is 418, with four computers is 827 and with eight computers is 1,612, considering the total execution time of 00:04:58. Note that the growth in the number of processed blocks is linear as presented in Figure 3.

3.2. The Bioinformatics Projects

P2P model is being used for applications that requires great amount of computer resources, since it offers some direct solution to modularity and scaling properties in large scale distributed systems. In this context, P2P approach is very important to bioinformatics. Thus, we have conducted

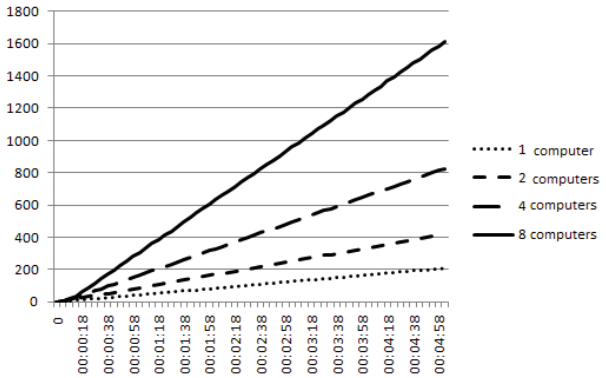


Figure 3. The number of blocks processed over time.

this experiment to illustrate the use of our EDA prototype in this area.

In this project we executed a particular bioinformatics application, BLAST, which is a family of algorithms for searching similarities between two biological sequences, widely used in genome projects [1]. We used real data of $Escherichiacoli$ or $E.coli$ bacteria, which is very known to bioinformatics community. The objective of the experiment is to apply BLAST algorithm to $E.coli$ sequences in order to validate EDA aspects of distribution, parallelism, and fault-tolerance. Although the $E.coli$ sequences is not expressive enough to justify high-throughput processing, our intention is to evaluate the abstraction of EDA proposal from the application perspective in different domains.

To implement the distributed solution, we used the same approach of the turtle project, presented in Section 3.1, according to the simple event processing style. The computational platform used in this test bed (experiment) had eight machines with the same configuration: Intel(R) Core(TM)2 Duo CPU E7500, 2.93GHz, 2GB of RAM and Ethernet 10/100 MB/s. Table 2 presents the execution time to process distributed BLAST over 6,400 $E.coli$ sequences. Note that the growth in time is linear, just like the turtle project presented in Figure 3.

3.3. Experimental Evaluation

Considering the fact that both experiments were carried out in different domains, extra effort was not necessary in the EDA layers. The modularity of our EDA framework was guaranteed by the defined layers and the modules. The provided interfaces (e.g., $Environment$, $EventManager$) assured the flexibility of our EDA framework, since it could encapsulate the application in both experiments, without generating extra implementation efforts to the application development. Other computer intensive P2P applications may use our EDA in a similar way.

Table 2. Time execution to process 6,400 sequences.

Number of Computers	Execution Time
1	00:24:44
2	00:12:25
4	00:06:16
8	00:03:07

In addition, our architecture promotes a high level of abstraction of the underlying P2P technology used, preserving flexibility. The EDA framework could benefit from the scalability and decentralization guaranteed by the P2P middleware. In this work, we have analyzed basic fault-tolerance features of our framework through the use of Scribe, which implements the services of the Network Adapter layers. The analysis consisted in repeatedly disabling the root node during our test beds. The *rendez-vous* point was reassumed without harming the processes computation.

As a preliminary evaluation of our approach the conducted test beds were satisfactory, but further experiments are necessary for a comprehensive evaluation, specially considering scalability and fault-tolerance features. The modularity and the extensibility of our framework may allow the use of other implementations for the security and reliability of the P2P processes at runtime, such as the work proposed by Spanoudakis [16] in the context of mobile P2P systems.

4. Related Work

Our proposed EDA functionally keeps the same principles presented in the EDA architecture, but the elements are organized in a different way, since our focus lies on P2P applications [8]. Pastry was chosen in this work due to its advantages over previous P2P frameworks as JXTA [19]. JXTA modular design and building blocks makes it relatively quick and simple to prototype P2P systems like document sharing and instant messaging in a timely fashion. Nevertheless, its design choices had severe impact on its scalability and reliability. JXTA is defined by a six layer XML based protocol stack that makes its implementations complex, heavyweight and slow [6].

Quite a few research work have focused on event-driven systems [3, 4, 7]. The EDA pattern has been usually associated to SOA, since they are complimentary, as SOA focuses on the decomposition of business functions and EDA focuses on business events [14]. Our approach to EDA is com-

prehensive since the development of P2P applications can be related to services through the decomposition of business functions.

Much research has been done to propose systems over structured P2P networks [18]. There are many applications developed using a P2P approach in various domains, particularly in bioinformatics [11]. However, as far as we are concerned, we could not find research work that integrates EDA approach to P2P networks with focus on a flexible architecture suitable for a multitude of various application domains.

5. Conclusion and Future Work

In conclusion, this work proposed a flexible EDA for P2P based applications. We implemented a basic version of the proposed EDA, which accommodates simple event processing style. We evaluate our proposal using two different domain applications exploring distribution of the P2P network. Our results an easy integration with P2P technologies, while exploring the features provided by the underlying P2P middleware used. Our preliminary results show that our EDA framework could be used seamlessly in two different domains.

As future work, we intend to expand the basic version of our current EDA implementation in order to deal with complex event processing style. Particularly, extend the implementation to deal with various and distinct event types simultaneously. For example, applications in web domain with events of different source types, video, text and image. We also plan to evaluate our framework for its flexibility using different P2P middlewares for validation purposes.

Another important future work is concerned to the extension of the current EDA core in order to transform the application module to encapsulate agents approach. In this direction, computer-intensive applications that may take advantage of multi-agent approach, through the use of rationality and distributed resources can take advantage of our EDA approach.

References

[1] S. Altschul, W. Gish, W. Miller, E. Myers, and D. Lipman. Basic Local Alignment Search Tool. *Journal of Molecular Biology*, 215:403–410, 1990.

[2] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron. Scribe: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communications (JSAC)*, 20:2002, 2002.

[3] S. Chakravarthy and R. Adaikkalavan. Events and streams: Harnessing and unleashing their synergy! In *Proc. of the 2nd DEBS, Italy*, pages 1–12. ACM, 2008.

[4] H. Dempo. Qos evaluations of distributed event orchestration system. In *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems*, DEBS '09, pages 22:1–22:5, New York, NY, USA, 2009. ACM.

[5] E. Gamma, R. Helm, R. Johnson, and J. M. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1995.

[6] E. Halepovic and R. Deters. The costs of using jxta. In *Proceedings of the 3rd International Conference on Peer-to-Peer Computing*, P2P '03, pages 160–, Washington, DC, USA, 2003. IEEE Computer Society.

[7] A. Hinze, Y. Michel, and L. Eschner. Event-based communication for location-based service collaboration. In *Proc. of the 20th ADC - Volume 92*, pages 125–134, Australia, 2009. Australian Computer Society, Inc.

[8] B. M. Michelson. Event-driven architecture overview. Technical report, Patricia Seybold Group, OMG, 2006. http://www.omg.org/soa/UploadedDocs/EDA/bda2-2-06cc.pdf.

[9] S. Ratnasamy, P. Francis, S. Shenker, and M. Handley. A Scalable Content-Addressable Network. In *In Proceedings of ACM SIGCOMM*, pages 161–172, 2001.

[10] S. Rhea, C. Wells, P. Eaton, D. Geels, B. Zhao, H. Weatherspoon, and J. Kubiatowicz. Maintenance-Free Global Data Storage. *IEEE Internet Computing*, 5(5):40–49, 2001.

[11] E. O. Ribeiro, M. Walter, M. M. Costa, R. Togawa, and G. Pappas. p2pBIOFOCO: Proposing a Peer-to-Peer System for Distributed BLAST Execution. In *10th IEEE HPCC, China*, pages 594–601, 2008.

[12] A. Rowstron and P. Druschel. *Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems*, volume 2218, pages 329–350. Springer, 2001.

[13] R. Schollmeier. A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications. In *Proceedings of the First International Conference on Peer-to-Peer Computing*, P2P '01, pages 101–, Washington, DC, USA, 2001. IEEE Computer Society.

[14] H. Sharma and J. Odell. Event-Driven Architecture (EDA) And Its Relationship with SOA & BPM. Technical report, OMG, September 2006.

[15] D. Shi, J. Yin, Z. Wu, and J. Dong. A Peer-to-Peer Approach to Large-Scale Content-Based Publish-Subscribe. In *Proc. of the 2006 IEEE/WIC/ACM WI-IATW*, pages 172–175, USA, 2006. IEEE Computer Society.

[16] G. Spanoudakis and K. Androutsopoulos. Monitoring security and dependability in mobile p2p systems. In *Proc. of The 3rd International Conference for Internet Technology and Secured Transactions*, ICITST '08. IEEE Computer Society, 2008.

[17] I. Stoica, R. Morris, D. Karger, F. M. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. of ACM SIGCOMM*, volume 31, pages 149–160, New York, NY, USA, October 2001. ACM.

[18] L. Stout, M. A. Murphy, and S. Goasguen. Kestrel: an XMPP-based Framework for Many Task Computing Applications. In *Proc. of the 2nd Workshop on Many-Task Computing on Grids and Supercomputers*, MTAGS '09, pages 11:1–11:6, New York, NY, USA, 2009. ACM.

[19] W. Yeager and J. Williams. Secure peer-to-peer networking: The jxta example. *IT Professional*, 4:53–57, 2002.

A Formal Approach for Incorporating Architectural Tactics into the Software Architecture

Hamid Bagheri
University of Virginia,
151 Engineer's Way,
Charlottesville, VA 22903 USA
hb2j@virginia.edu

Kevin Sullivan
University of Virginia,
151 Engineer's Way,
Charlottesville, VA 22903 USA
sullivan@virginia.edu

***Abstract*—Software architects principally leverage successful architectural design practices systematized in terms of architectural styles and tactics. While architectural tactics focus on particular non-functional properties, styles are typical solutions that generally coverage several aspects. The problem is that we do not yet have a formal account of how styles and tactics interact in a way that is sufficient to enable automated synthesis of architectures from application models and the combination of architectural styles and tactics specifications. The contribution of this paper is an extension of our previous work on formal *architectural maps* that makes this fundamental relationship clear, and a demonstration that it enables automated formal derivation of architectures.**

***Keywords*-Software Architecture; Architectural Styles; Architectural Tactics; Architectural Maps; Alloy Language.**

I. Introduction

Software engineering researchers and practitioners have recognized software architecture as a promising means of managing the complexity of software systems [20], [22]. Other studies have further shown its significant role in achieving systems' non-functional properties [7], [13]. Developing a sound and appropriate architecture, however, remains a significant and intellectually challenging activity.

Notwithstanding the growing body of research and significant progress in architecture-based software development, the success of architecture construction still mainly depends on the experience of software architects [14]. As software systems unrelentingly grow in complexity and size, designing such systems manually in this way becomes more costly and labor-intensive, and, once completed, hard to change. Moreover, failure to meet the requirements, which might be later discovered, causes the need to backtrack, which is expensive especially in the case of architectural decisions.

To address this issue, we proposed a model-based approach to automated synthesis of architectural descriptions from formal, abstract application models and separate choices of architectural style specifications [3]. While our previous work deals with issues of architectural structure, it does not address support for non-functional properties.

Architectural tactics [5] have emerged as architectural decisions that codify and record best practice knowledge for achieving a desired quality attribute. However, despite years of research and practice, we have little formal account of how styles and tactics interact. Furthermore, the reusability of architectural tactics is hampered by a number of factors, among which is the absence of formalization, as they are typically represented as a set of documents. In fact, unlike architectural styles, characteristics of an architectural tactic are not understood in such a formal manner, making it difficult for architects to make informed choices.

In this paper, we present a novel approach based on our formal notion of *architectural maps* that makes this fundamental relationship clear. In particular, we show the feasibility of incorporating the notion of tactics with that of styles to enhance the specification of architectural maps, which provides the capability of formally applying refinement tactics to improve quality attributes.

This paper makes three principal contributions in this area. First, we formalize and automate architectural tactics by means of constraint predicates parameterized by choices of *application type* and *architectural style*. Second, we show that our ideas for formal and reusable representation of tactics parameterized in this way can be realized in practice. More specifically, starting with an application model and using the formal definition of the architectural maps as well as the architectural tactics provided in this work, it is possible to arrive to the set of compliant architectural descriptions. Third, we have developed the technique as an extension of our tool for model-based development of software architecture, *Monarch* [1], which leverages Alloy as the basis for the specification of mappings' semantics and the automatic analyzer [15].

The rest of the paper is organized as follows. Section II explains the background context of our work. While Section III presents our approach for incorporation of architectural tactics into the formal synthesis of software architecture, Section IV demonstrates that it can be realized in practice. In Section V, we report on experiences applying our approach and tool set and provide a discussion. Section VI briefly surveys related work. Section VII finally concludes.

II. Background: Architectural Maps

In this section, we provide an overview of the notion of *architectural maps*, introduced and formalized in our previous work [3] based on the idea that choices of architectural style are made separately from choices of essential application properties, and that architects in essence implement a mapping that takes an application model and an architectural style to an architectural model. Knowledge of this mapping is crucial to expertise in software design. Given an application description of some type, the experienced designer knows both what architectural style to pick, and how to map an application description of the given type to an architectural model in the chosen style. In some sense an architectural map embodies knowledge of how to realize different types of applications in different architectural styles.

Figure 1, which is an extension of our previous work, represents the fundamental elements of our model on the basis of the architectural maps and their relationships: s, an *architectural style* specification; $\{a_i\}$, a set of architectural models; a binary relation, *conforms,* on the cross product, that encodes the conformance of the set of architectural models, $\{a_i\}$, to an architectural style, s; t, an *application type* specification; m, an application model; an analogous binary relation, *conforms* encoding the conformance of a given application model, m, to a given application type, t; and a relation, *refines,* encoding the notion that a set of application-specific architectural models, $\{a_i\}$, refines, or implements, the application model, m. Finally, we have $map_{(t,s)}$, which takes an application model, m of type t, and an architectural style, s, to the set of architectural models, $\{a_i\}$, such that $refines(a_i, m)$ and $conforms(a_i, s)$.

More recently, we have shown that the proposed separation of concerns supports a model-based development and tools approach to architectural-style-independent application modeling, and architecture synthesis with style as a separate design variable [4].

III. Approach

Above and beyond providing a proof of concept of the feasibility of the proposed formal *architectural maps* in an automated way, previous works revealed deficiencies in current specifications of architectural styles. That is, applying architectural maps leads not to a single plausible architecture, but to a set of architectures, where not any instances in that set necessarily satisfies all required non-functional properties. In fact, as can be seen in the diagram, one more optimizing search step is required within that space to find the subset of plausible instances especially with respect to quality attributes. This is mainly because style specifications to which application models were being mapped are still underspecified, and in turn their corresponding architectural maps leave overly large architectural spaces.

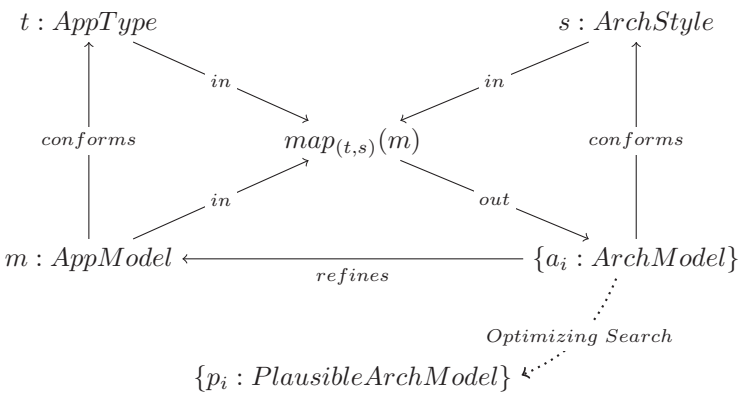


Figure 1. Key entities and relations involved in the notion of *architectural maps.*

As such, the systematic process of getting from an application model to an architecture involves two steps: (1) constraining the formal architectural space; (2) performing optimizing search over the derived space for properties of interest. The more we can do in the mapping, the less we have to do in the search. Therefore, to achieve a practical automated design tool for software architecture, there is a pressing need to formally specify relationships between architectural design decisions and quality attributes [7].

Architectural tactics are architectural decisions concentrating on non-functional properties that shape the system's architecture but in a smaller extent compared to the style decisions [5]. Several architectural tactics have been proposed for various quality attributes such as reliability, performance and modifiability. As architectural styles have extensive use in the architectural modeling, relating styles to tactics provides a basis for making rational design decisions [5]. That is, tactics can systematically be leveraged to adapt styles for the purpose of improving their properties with respect to particular quality attributes. As such, the structure of the tactic must fit within the rules implied by the style. To effectively apply architectural tactics along with the styles the architect needs to realize their side effects and how they relate each other [14].

This paper develops the notion that architectural tactics are not independent of, but rather are parameterized by, choices of application type and architectural style, and it is possible to represent tactics parameterized in this way in a formal and reusable form. Architectural styles and tactics then can be considered as complementary techniques, subsequently allowing for the automated reuse of architectural best practices. Indeed, the effect of a tactic refines and extends the architectural map.

$$map_{(AppType,ArchStyle)} \xleftarrow{extends} tactic_{(AppType,ArchStyle)}$$

In our approach, one expresses an application model as an instance of an *application type,* then selects an *architectural style* in which to synthesize an architecture for the application. The combination of application type and architectural style selects a mapping from applications of that type into architectural instances in that style. The decision to use a particular tactic in mapping an application model to an architecture then involves the explicit inclusion of such a reusable tactic specification along with a more generic architectural map. Applying this compound map to the application model yields a family of correct-by-construction architectural descriptions for the given application in the given style supporting the given tactics. In the next section, we show that our ideas can be reduced to practice.

IV. Automation

To demonstrate the viability of implementing tools that support automated formal synthesis of software architectures complying with the rules implied by choice architectural styles and tactics, we present a proof of concept leveraging the Alloy language for formalization and the Alloy Analyzer for automation.

Alloy is a lightweight specification language based on the first-order relational logic [15]. *Signature* is an essential construct of the Alloy language that represents the basic type of elements and the relationships between them. *Facts* can be used to define constraints over model instances. Alloy further provides *Predicates* to be used in defining parameterized reusable constraints always evaluated to be either true or false. A *Function* similar to a predicate can be invoked by instantiating its parameters, but what it returns is either a true/false or a relational value instead.

Its SAT-based analyzer makes automatic analysis of Alloy models possible. The Alloy Analyzer can be used either to find satisfying solutions with respect to the constraints of a given model, or to find any possible counterexample violating constraints in a given model. The former is used in our approach to compute architectural models.

Five pieces of Alloy specifications are conjoined in the process of synthesizing compliant architectural models: (1) an *application type* represented in an Alloy module; (2) an *architectural style* specification module; (3) an *application model,* comprising an instance of an application type, again represented in an Alloy module; (4) an *architectural map* specifying the relationships required to hold between an application of the given type and an architecture in the given style, represented in an Alloy predicate; and (5) a *tactic predicate* specified for a given pair of an application type and an architectural style to incorporate a given tactic into the software architecture. The Alloy Analyzer computes satisfying solutions to the compound specification, leading to the synthesized architectures.

V. Evaluation

Our claim we make in this paper is that it is feasible to represent architectural tactics parameterized by choices of application type and architectural style in a formal and reusable form. In this section, we report and interpret data from experimental testing of our approach and hypothesis. More specifically, we first show that our approach supports formal specifications of architectural tactics with respect to the *architectural maps* developed for each pair of an application type and an architectural style. We then show that it also supports automatic synthesis of architectural instances which conform to the rules implied by both the application model and the architectural style as well as supporting the given architectural tactic.

In support of our approach, we have extended our tool suite, Monarch, which is available for download and inspection [1], and applied it to several case studies. This paper cannot accommodate detailed presentations of all of our experiments. Rather, we report one case in more detail, which is about the employment of a fault detection tactic in the process of formal synthesis of Lunar Lander architecture [22] in the *implicit-invocation* (II) style from its abstract application description in the *sense-compute-control* (SCC) application type. This section is followed by presenting discussion over the experiments.

A. Application Model: $Lunar\ Lander_{SCC}$

Our application case study is inspired by Taylor et al. [22]. In their new textbook on software architecture, they illustrate the structuring of an embedded control system called *Lunar Lander* in different architectural styles. The *FlightControl* analyzes the information provided by various sensors, i.e. *Altimeter*, *Gyroscope*, *Fuel level indicator* and *Engine control switch*, to maintain the state of a spacecraft. Then, it provides updated data to the various actuators: *Descent engine controller*, *Attitude control thruster* and *Display*.

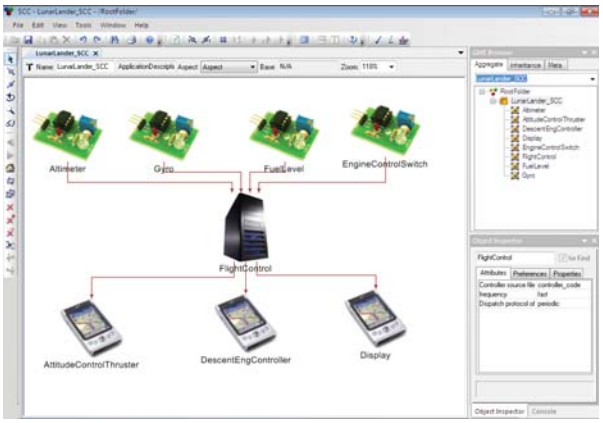


Figure 2. Lunar Lander application modeled within GME using the generated modeling environment for our SCC metamodel.

Figure 2 illustrates Lunar Lander application modeled as an instance of our SCC metamodel within Generic Modeling Environment (GME). We customize GME such that application types are realized concretely as GME metamodels, providing *architecture-independent modeling languages (AIML)*. We specify a formal model of the SCC type to model applications in which sensors and actuators are connected to controllers that cycle through the steps of fetching sensor values, computing function values, and sending outputs to actuators.

```
module LunarLander
open SCC

one sig Altimeter extends Sensor{}
one sig FuelLevel extends Sensor{}
one sig EngineControlSwitch extends Sensor{}
one sig Gyro extends Sensor{}
one sig AttitudeControlThruster extends Actuator{}
one sig DescentEngController extends Actuator{}
one sig Display extends Actuator{}
one sig FlightControl extends Controller{}{
 sensors= FuelLevel+ EngineControlSwitch+ Gyro+ Altimeter
 actuators= DescentEngController+ Display+
      AttitudeControlThruster
 controller_dispatch_state = periodic
 frequency_state = fast
 program = controller_code
}
...
```

Listing 1. Lunar Lander application description represented in Alloy

Listing 1 partially outlines the Alloy representation of the Lunar Lander application description, automatically synthesized from its concrete representation. The application model (an Alloy module) imports the *SCC* module, and defines four sensors and three actuators using signature extension to subtype the *Sensor* and *Actuator* types. The only controller of the application is then synthesized, while its properties are specified as Alloy facts.

B. Tactic: $PingEcho_{(SCC,II)}$

Ping/Echo and *Heartbeat* are two architectural tactics proposed for *Fault Detection*, which is one of the four design concerns for reliability [5]. In this case study we employ the Ping/Echo tactic, although the structures of these two tactics are very close. For detecting a fault using the aforementioned tactic, a monitor component regularly sends ping messages to receivers. The receiver is then supposed to respond with an echo message within a certain time period, otherwise it is considered to have failed.

Listing 2 partially presents the Alloy representation of the Ping/Echo tactic. It starts by importing the Alloy specification modules of the pair of an application type and an architectural style, for which an architectural map, enhanced by the architectural tactic under consideration, is developed. The parameterized predicate which gets as input a set of pinged components then, in line 6-7, dedicates an IIObject, an implicit-invocation object which extends Object [3], to the pinger. It has a port of type *Procedure* that can be called by other components. Lines 9–17 state that for each receiver, its *Procedure* port is connected to the *call* port of the pinger's IIObject via a *PrecedureCall* connector connected to it. On the other hand, the *call* ports of receivers are connected to the *Procedure* port of the pinger's IIObject.

```
1  module PingEcho_SCC_II
2  open SCC
3  open II
4
5  pred pingEcho_SCC_II(s:set needHandle){
6    one o:IIObject| o.handle = Pinger &&
7    no handled:needHandle-Pinger| handled in o.handle &&
8    one p: Procedure| p in o.ports &&
9    all n:needHandle| n in s =>{
10     one c1: procedureCall |
11     c1 in n.~handle.procedure[attachments].ran.connector&&
12     c1 in o.call[attachments].ran.connector
13
14     one c2: procedureCall |
15     c2 in n.~handle.call[attachments].ran.connector &&
16     c2 in o.procedure[attachments].ran.connector
17   }
18 ...
19 }
```

Listing 2. The Alloy predicate for $Ping/Echo$ tactic developed for the $architectural\ map$ of the pair of SCC application type and II architectural style

C. Incorporation of Tactic into Architectural Map

After modeling the application and specifying the mappings and tactics predicates, we need to define a compound module incorporating tactics' specifications into the process of synthesizing satisfying architectural models. Here, we have leveraged the architectural map for the SCC application type and the implicit-invocation architectural style specified in earlier work [4].

```
module LunarLander_SCC_II

open LunarLander as AppDesc
open SCC_II as map_SCC_II
open PingEcho_SCC_II

pred execute{
  map_SCC_II[]
  PingEcho_SCC_II[FlightControl]
}
```

Listing 3. The Alloy module for incorporation of $Ping/Echo$ tactic

Listing 3 represents the Alloy module for synthesizing Lunar Lander architectural models in implicit invocation style supporting the $Ping/Echo$ tactic. It starts by importing the modules of the Lunar Lander application description, the (SCC, II) architectural map and the $Ping/Echo$ tactic. The *execute* predicate is then specified, which calls both mapping and pingEcho predicates. The *Flightcontrol* element is then passed as an input to the parameterized tactic predicate to address the potential issue of crashing or otherwise failure of this component.

D. Satisfying Architectural Models

Given all the specifications and mapping constraints, Monarch using the Alloy Analyzer computes a set of architectural models that refine the application description in conformance with the fully formal definitions of the implicit invocation architectural style and the Ping/Echo tactic,

represented as Alloy solutions. To make those low-level, XML formatted outputs human-readable, it then using our *Alloy2ADL* transformer [4] converts them to an architecture description language (ADL).

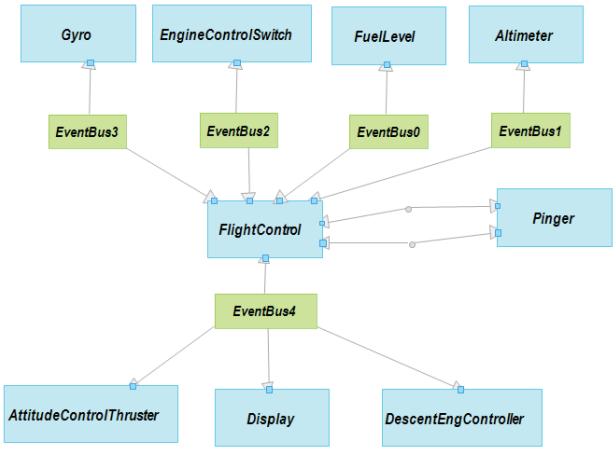


Figure 3. One of the computed architectural models represented in Acme.

A considerable number of ADLs have been proposed during the past several years to model specific application domains or as general purpose architectural modeling tools. We use Acme [11] in our tool, which is a mature general purpose ADL, with a particular support for architectural styles. It is also designed to work as an interchange between wide varieties of architecture description languages. Figure 3 shows an instance of the computed architectural models in Acme. In general, there is more than one satisfying solution, and the result is a set of formally derived architectural models for the given application with respect to the enforced architectural constraints.

E. Discussion

This work shows that architectural tactics can be formalized and implemented as executable specifications. By incorporating both the architectural styles and tactics during the synthesis, our approach formally derives narrower architectural spaces, which reduces the difficulty of design space exploration. We note that automated search could occur not only within one architectural style but across styles incorporating a set of tactics – leading to a tool able to automatically find appropriate architectural models for a given application model.

It is also worth mentioning that the incorporation of a tactic within a style might result in either the alteration of the style or even addition of a style to the architecture [14]. As a case in point, by application of the Ping/Echo tactic to an architecture developed upon the Pipe and Filter style, the derived architecture contains an added component in charge of performing monitoring process and having connections with the pinged components (filters), which consequently imposes Broker style into the architecture. The current version of Monarch avoids such mismatches by deriving an empty set of architectural instances, as there is no such architectural model that thoroughly complies with rules implied by the Pipe and Filter style and embodies an instance of the Ping/Echo tactic simultaneously. As a future work, we plan to investigate ways of handling such cases.

Another interesting avenue for future work is to provide more rigorous representation of application models especially with respect to non-functional requirements (NFRs). The more complete the specifications of application models, and the more tightly constrained the mapping specifications, the narrower the outcome architectural spaces, and the slighter the required postprocessing optimizing search. Work by Rosa et al. [19] and more recently by Jackson et al. [16], proposing that many forms of NFRs can be considered as constraints over the space of models, suggest interesting possibilities in this regard, which we intend to explore.

Finally, the upshot is that those who are doing research in software architecture, patterns, and tactics should recognize the need to specialize tactics to the particular settings induced by a choice of both application type and architectural style, and could consider the function of tactics as extending and refining architectural maps.

VI. Related Work

We can identify in the literature three categories of works that are related to our research. The first one concerns works that deal with formal modeling of architecture using Alloy. The second encompasses researches on leveraging architectural best practices to satisfy quality attributes. Finally, the last area of related work focuses on formal approaches to the evolution of architectures under the guidance of architectural styles

Focusing on the first category, numerous researchers have applied Alloy to formal specification of software architecture. Among others, Kim et al. [17] proposed an approach to translate architectural styles described formally in an architecture description language to Alloy language, and in turn, to verify properties implied by architectural styles. Georgiadis et al. [12] similarly specified structural architectural styles as a set of constraints in order to control runtime reconfiguration by means of constraints evaluation. While these works use formal modeling and verification for detecting architectural mismatch [9], we focus on preventing such mismatches with a formal synthesis approach. That is, we use Alloy not only to specify the application model or architectural constraints, but also to model the spaces of mappings consistent with both given application models as well as target architectural styles and tactics, and to automate the mapping process.

Bucchiarone and Galeotti [6] also proposed an approach based on graph grammars and *DynAlloy* [8] to verify programmed dynamic software architectures in which all

possible architectural changes are defined before run-time. Although this work, like other work we have studied, lacks an explicit notion of separating application description from other design decisions, it appears to have the potential to help us extend application models to include richer semantics.

Focusing on the second category, a number of approaches explored relating architectural styles and tactics stimulated by them regarding various quality attributes (e.g. modifiability [2], reliability [14]). Extending the same line of work, Kumar et al. [18] recently proposed a more general pattern-oriented knowledge model composed of four dimensions, including the pattern to tactic relationship. Our work is different from these works in several ways. First, our work is geared towards the systematic specification of mapping architecture-independent application models into the architectural styles and the associated tactics. Second, our approach has a formal basis which is missing in theirs.

Regarding the last area of related work, Tamzalit and Mens [21] recently proposed an approach for the evolution of an architecture description under the guidance of architectural style. To this extent, they rely on a formalism based on graph transformation. The other similar work is the recent work of Garlan et al. [10] on the evolution of programs with respect to architectural style. The premise of this work is that it is sometimes necessary to change a program written in one architectural style into a related program in another style. These works share with ours an emphasis on formal application of architectural best practices, but our work focuses on formal mappings of architecture-independent application models to a diversity of realized architectural models in practical architectural styles and tactics.

VII. Conclusion

While a wealth of research has been done on software architecture, architectural styles and architectural tactics, very little has been done on automated support for the derivation of software architecture with respect to styles and tactics. We presented an approach based on the notion of *architectural maps*, that formally supports automatic synthesis of architectural models from abstract application models in conformance with the architectural styles. In this paper, we showed that architectural maps can also incorporate architectural tactics in a formal and reusable form.

References

[1] Monarch tool suite. http://monarch.cs.virginia.edu/.

[2] F. Bachmann, L. Bass, and R. Nord. Modifiability tactics. Technical report SEI-2007-TR-002, 2007.

[3] H. Bagheri, Y. Song, and K. Sullivan. Architectural style as an independent variable. In *Proceedings of the 25th IEEE/ACM International Conference on Automated Software Engineering*, 2010.

[4] H. Bagheri and K. Sullivan. Monarch: Model-based development of software architectures. In *Proceedings of the 13th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, pages 376–390, 2010.

[5] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. Addison-Wesley Professional, 2nd edition, 2003.

[6] A. Bucchiarone and J. P. Galeotti. Dynamic software architectures verification using DynAlloy. In *Proceedings the 7th International Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT)*, 2008.

[7] P. Clements and M. Shaw. "The golden age of software architecture" revisited. *IEEE Software*, 26(4):70–72, 2009.

[8] M. F. Frias, J. P. Galeotti, C. G. L. Pombo, and N. M. Aguirre. DynAlloy: upgrading alloy with actions. In *ICSE*, 2005.

[9] D. Garlan, R. Allen, and J. Ockerbloom. Architectural mismatch: Why reuse is still so hard. *IEEE Software*, 26(4):66–69, 2009.

[10] D. Garlan, J. M. Barnes, B. Schmerl, and O. Celiku. Evolution styles: Foundations and tool support for software architecture evolution. In *WICSA*, 2009.

[11] D. Garlan, R. T. Monroe, and D. Wile. Acme: architectural description of component-based systems. In *Foundations of component-based systems*, pages 47–67. 2000.

[12] I. Georgiadis, J. Magee, and J. Kramer. Self-organising software architectures for distributed systems. In *Proceedings of the first workshop on Self-healing systems*, 2002.

[13] N. B. Harrison and P. Avgeriou. Leveraging architecture patterns to satisfy quality attributes. In *ECSA*, 2007.

[14] N. B. Harrison and P. Avgeriou. How do architecture patterns and tactics interact? a model and annotation. *Journal of Systems and Software*, 83:1735–1758, 2010.

[15] D. Jackson. Alloy: a lightweight object modelling notation. *TOSEM*, 11(2):256–290, 2002.

[16] E. K. Jackson, D. Seifert, M. Dahlweid, T. Santen, N. Bjorner, and W. Schulte. Specifying and composing non-functional requirements in Model-Based development. In *SC*, 2009.

[17] J. S. Kim and D. Garlan. Analyzing architectural styles. *Journal of Systems and Software*, 83(7):1216–1235, 2010.

[18] K. Kumar and P. T.V. Pattern-oriented knowledge model for architecture design. In *PLoP*, 2010.

[19] N. S. Rosa, G. R. R. Justo, and P. R. F. Cunha. Incorporating non-functional requirements into software architecture. In *Proceedings of the 15 IPDPS 2000 Workshops on Parallel and Distributed Processing*, pages 1009–1018, 2000.

[20] M. Shaw and D. Garlan. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, 1996.

[21] D. Tamzalit and T. Mens. Guiding architectural restructuring through architectural styles. In *ECBS*, 2010.

[22] R. N. Taylor, N. Medvidovic, and E. Dashofy. *Software Architecture: Foundations, Theory, and Practice*. Wiley, 2009.

Towards Quality Based Solution Recommendation in Decision-Centric Architecture Design

Lei Zhang, Yanchun Sun*, Yuehui Peng, Xiaofeng Cui, Hong Mei
Institute of Software, School of Electronics Engineering and Computer Science, Peking University
Key Laboratory of High Confidence Software Technologies, Ministry of Education, Beijing 100871, China
Email: {zhanglei, meih}@pku.edu.cn, {sunyc, pengyh08, cuixf04}@sei.pku.edu.cn

***Abstract*—Designing software architecture is an important and complex activity. To design a high quality architecture, architects need to make decisions on a number of inter-dependent design issues, each of which usually has a set of viable alternative solutions. Moreover, each viable solution often has diverse impact on different quality attributes which often conflict with each other. Existing methods for architecture design still face the challenge of bridging the gap between software requirements and high quality architecture. To alleviate the complexity of architectural design, we propose a *decision*-centric approach: ABC/DD. We attempt to provide pragmatic assistance for practitioners to narrow this gap, and also ensure the quality of target architecture. This approach transits from requirements to architectures through a five-step process including *requirement refining, solution exploiting, relation identifying, solutions and quality attributes synthesizing*, and *architecture deciding*. Finally we illustrate the applicability of this approach using a case study.**

keywords: architecture design; design decision; quality attribute

I. Introduction

Designing software architecture is one of the most important activities for a software system. Because architecture not only provides the key framework for the earliest *design decisions*, which are taken to achieve both functional and non-functional requirements [1], but also plays a crucial role in the whole lifecycle of software system [2].

On one hand, designing software architecture is not a trivial activity. Architects need to elaborate system requirements and then transform them into a viable software architecture with respect to reusable subsystems, existing technology and quality attributes, etc. Because of the inherent gap between problem domain and solution domain, the transition from software requirements to architecture design remains a challenging problem. There is still a lack of fundamental process models and guidelines. In practice, architects usually design based on their own experience and intuition [3].

On the other hand, designing a high quality architecture is also very difficult, because architects usually need to make *design decisions* on a number of inter-dependent design issues, each of which has a set of viable alternative solutions [4]. Moreover, different alternatives usually have diverse impacts on different quality attributes. Even worse, such impacts sometimes conflict with each other. Consequently, architects have to deliberately tradeoff among these entangled problems.

* Corresponding author

In recent years, a new research trend has emerged in architecture community to facilitate a better software architecture design process. It is suggested that *design decisions* should be modeled and considered as a first class entities [5]. The seminal paper [6] advocates this perspective, in which software architecture can be seen as a composition of architectural *design decisions*. Many design methods have been proposed in such a paradigm shift [7], including Archium [8], AREL [9], ArchDesigner [4], AQUA [10], and Bayesian-Belief Network based method (BBNbASM) [11], etc. These design methods focus on describing architecture design process using *design decision* and they greatly promote the progress of research. However, most of these design methods do not aim at directly guiding architects to derive architecture from requirement. Besides, only a few of these decision-centric design methods explicitly support quality analysis. More details about these methods is analyzed in Section IV.

To address this problem, we propose a decision-centric architecture design method: **ABC/DD (Architecture Based Component composition by Design Decision)** in order to narrow the gap between requirement and architecture. We use "*Divide and Conquer*" as the principle idea of our approach. We firstly divide system requirement as a set of design issues, then solve these design issues and determine the quality attributes independently, and then synthesize solutions for each design issue to generate a set of candidate architectures, finally recommend the optimal candidate based on quality attributes. Specifically, our approach transits from requirements to architectures through a five-step process including: (1) *requirement refining*, (2) *solution exploiting*, (3) *relation identifying*, (4) *solutions and quality attributes synthesizing*, and (5) *architecture deciding*.

The remainder of this paper is structured as follows. Section II presents our approach and tool support; Section III demonstrates the applicability of our approach using a case study; then Section IV gives an overview of related work including decision-centric architecture design methods, and quality attributes related research in architecture; finally Section V concludes the paper.

II. Proposed Approach

A. Design Decision-Centric Meta-Model

Based on the Core model [12], we propose the meta-model of our approach in Fig. 1.

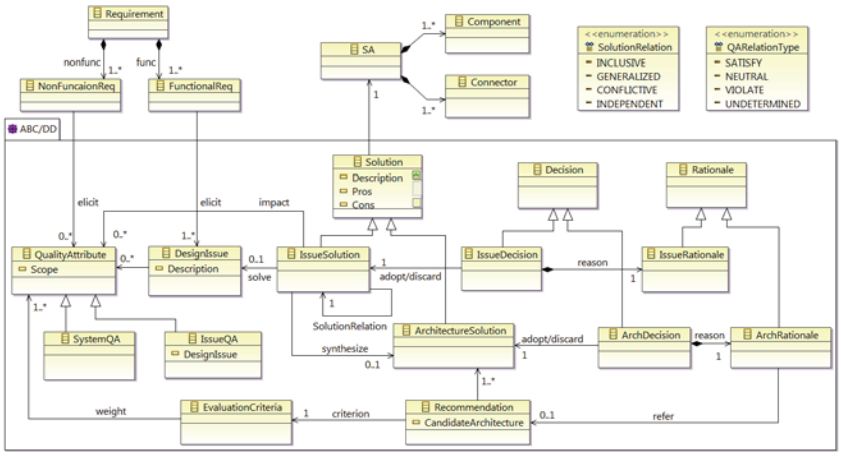


Fig. 1. The design decision-centric meta-model

In this approach, a *design issue* means an architecturally significant requirement. It acts as a linkage between the architecture design and requirements. *Design issues* can be either project-specific or reusable. For those appear in many architecture design projects, we named them "*reusable design issue*", and we can retrieve them from repository and then reuse them. And for those only appear in a specific project, they are "*project-specific design issue*", and need to be solved independently manually.

An *issue solution* provides a possible way of solving a *design issue*. And an *architecture solution* is a candidate architecture, which is synthesized from all *issue solutions* and solves every *design issue*. In our approach, each solution is instantiated by a set of components, connectors and their configuration(*C&C view*).

Quality attribute (QA), such as maintainability, portability, scalability, etc., are identified from non-functional requirement. The scope of quality attribute can be either a design issue (*IssueQA*) or the entire system *(SystemQA*). Usually, a design issue is related with multiple quality attributes. While several different design issues also can be related to the same one quality attribute.

Quality attributes do not share the same equal significance. We use an *evaluation criterion* to characterize different priorities of quality attributes in order to reflect system's quality goal. Based on an *evaluation criterion*, a *recommendation* will be presented among a set of candidate architectures. According to this *recommendation*, architects can make a *decision*, i.e., to accept or to reject one candidate architecture solution.

In addition, another important element enclosed in design process is *rationale*, which is the reason and justification behind a decision. Solution's impact on quality attributes is a kind of important rationale, and will be captured by ABC/DD automatically. Besides, architects can also denote some *pros*, *cons* and any comments to each decision. These *rationales* will be valuable when reviewing or evolving the architecture.

B. Decision-Centric Design Process: ABC/DD

Based on the above meta-model, we present an iterative process to implement our design decision-centric design method in Fig.2. We use an activity diagram to illustrate this process in five steps. Main inputs to this process include both functional and non-functional requirement. And outputs of this process include a final architecture, along with a set of unselected candidate architecture and captured rationale. We describe each step of this approach in turn.

Step (1) : Refine System Requirement

In the first step, architects need to clarify the requirements. Functional requirements describe supports that user needs, while non-functional requirements impose constraints on how the system should accomplish the systems functionality [13]. Architects deliberate and determine *design issues* from functional requirement, and identify *quality attributes* from non-functional requirement.

When architect refining the requirement, ABC/DD will search repository in order to match existing *design issues*. For example, "application integration style" is a common *design issue* when designing a distributed system, and possible alternative solutions include "Remote procedure call", "Message Bus", "Shared Database" and "File Transfer" [14]. These solutions are relatively 'stable', and less likely need to redesign in a new project. Thus, we can directly retrieve this set of alternative solutions and present them to architects. Reusing *design issue* not only reduces the burden, but also improves the quality of design.

Step (2) : Exploit Solutions

In this step, architects need to derive alternative solutions to each design issue. They can create or reuse a concrete design fragment. We adopt common used *Component & Connector View* (*C&C View*) to describe the solution.

For *reusable design issues*, architects can refer to existing issue solution and determine whether reuse one of them or not. For *project-specific design issues*, architects need to solve them on their own by designing a set of components and connectors.

Fig.3 and 4 are two examples of issue solutions.

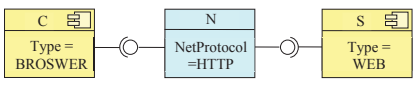


Fig. 3. Solution instance of "B/S structure"

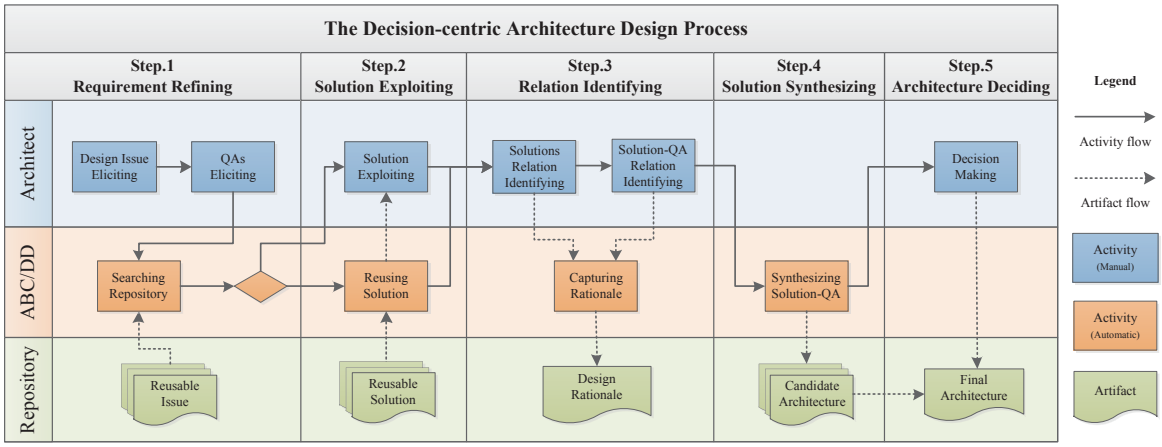


Fig. 2. The decision-centric architecture design process

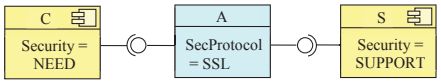


Fig. 4. Solution instance of "Security communication"

Step (3) : Identify Relations

(a) Relations between issue solutions

In this step, architects need to explore the *C&C* models for each issue solution, and identify relations between solutions of any two different issues. In [15], Kruchten et.al, present an ontology which describes a group of relations. We select four relations from them, which can be used in our approach, including:

- $INCLUSIVE$, means that one solution IS_a includes another solution IS_b, or IS_a is part of IS_b.
- $CONFLICTIVE$, indicates that two solutions are conflictive to each other and cannot be selected together to synthesize architecture solution.
- $GENERALIZED$, means that one solution is a generalization or a specialization of another solution.
- $INDEPENDENT$, means the above three relations do not exist.

Each pair of issue solutions should be identified as only one kind of these relations. Relations determined in this step are foundations for solutions synthesis in the following step. According to these identified relations, different issue solutions, which are not $CONFLICTIVE$, can be merged together. For example, issue solutions in Fig.3 and Fig.4 are $INDEPENDENT$, and they can be merged as one solution as shown in Fig.5.

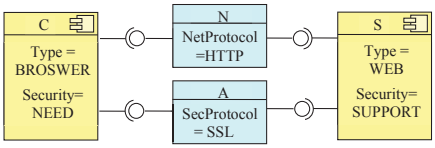


Fig. 5. The merged solution of (a) and (b) in Fig.3 & 4

(b) Relations between issue solutions and quality attributes

Each issue solution impacts on several quality attributes. And these impacts should be determined by architects after they design or reuse an issue solution. We use four kinds of relations to qualitatively describe such relations.

- $SATISFY(solution, QA)$
- $NEUTRAL(solution, QA)$
- $VIOLATE(solution, QA)$
- $UNDETERMINED(solution, QA)$ $[default]$

Step (4) : QA-Based Solutions Synthesize

According to the identified relations between the issue solutions, as well as the quality attributes for each issue solution, our recursive synthesis algorithm can build a combination tree, in which all feasible combinations of issue solutions can be explored. And it merges every feasible combination of issue solutions automatically. The outputs of this synthesis are the *C&C* models of candidate architecture solutions, and the established synthesis of quality attributes. We denote the quality of one candidate architecture as a vector:

$$CA._{QA} =< QA_1, QA_2, \ldots, QA_n > .$$

and then we assign a value to each relation as: $SATISFY = 1$, $VIOLATE = -1$, and $NEUTRAL = 0$. Then each quality attribute of candidate architecture can be calculated:

$$CA._{QA_i} = \sum_{i=1}^{n} value(IS_{x,y}, QA_i),$$

$$\forall x, y(IS_{x,y} \in CA._{QA_i} \wedge value(IS_{x,y}, QA_i) \neq null)$$

Step (5) : Architecture Deciding

(a) Evaluation Criterion

In most cases, different quality attributes have different importance. In order to express the desired quality goals, architects can give a partial order or specify concrete weights to all quality attributes.

For example:

$\{Real\text{-}time\} > \{Maintainability, Cost\}$, or

$Real\text{-}time._{weight} = 0.3$; $Maintainability._{weight} = 0.1$, etc.

These values will form a quality weight vector.

(b) Recommendation and Architecture Decision

In this final step, we already have quality value vectors for different candidates and quality weight vector. The dot product of these two vectors reflects the quality of each candidate architecture, and can be used for comparison.

To design a high quality architecture, we should satisfy QAs as more as possible. Therefore, after ranking all candidates,

the one which received the highest quality value will become the recommendation. Then architects can inspect this recommended architecture holistically and decide whether accept or reject it as the final architecture.

C. Tool Support

We have developed a visual architecture modeling tool named ABC Tool to support this design decision-centric design method. This tool implements an integrated environment that provides support for architecture design, automated architecture synthesis, quality-attribute analysis and candidate recommendation, etc. ABC Tool is developed as a suite of Eclipse Plug-ins. We mainly used two Eclipse open source frameworks: EMF (Eclipse Modeling Framework) to model our meta-model supported design method and GMF (Graphical Modeling Framework) to provide visual design ability.

III. Case Study

A. Case Description

This is a case study based on a real-world Commanding Display Systems (CDS) that has served in a Space Flight Center for years. And this system is currently in need of refactoring to deal with several emerging problems. Fig.6 shows the simplified architecture of the legacy system. In this architecture, there are a data source (DS) that reads various kinds of live data, and a set of monitors (Ms) that display the processed data. There is also a dedicated display server (Svr) that receives the data transmitted from DS, pre-processes the data in some specific ways, and pushes the data to Ms.

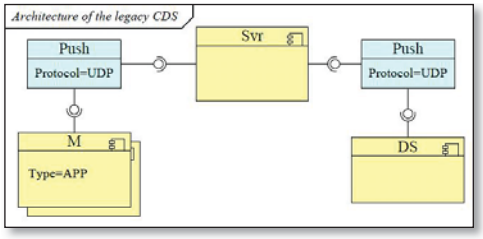


Fig. 6. Simplified Architecture of Legacy System

Several main problems of this legacy system include:

(a) Due to increasing amount of data, current design exposes the system to performance problem. When a burst of data occurs, it will cause seconds of delay. However, as a commending system, quality of real time is crucial.

(b) The history data is stored in Svr without database (DB) supporting. The management and usage of the history data are inefficient and inconvenient.

(c) Current CDS is designed as a stand-alone application. Because of a great deal of changing and upgrading requirements and implementations, the maintenance of this system is a hard work.

We applied our approach in this case study.

B. Applying the Method

Step 1: Identify System Requirement

Table I
Elicited Design Issues and Quality Attributes

Design Issue	*Description*	*Quality Attributes*	
		Issue	**System**
DI1	Live data acquiring: how to transfer the live data from DS to Ms.	Real time, Non-overload	Cost, Maintainability
DI2	History data acquiring: how to transfer the history data from DB to Ms.	Non-overload	
DI3	Persistence of the live data: how to store the lastest collected data into DB.	Recoverability	
DI4	Visually display both live and history data friendly in Ms.	N/A	

Firstly, we elicit design issues and quality attributes in Table I. In all, there are four design issues, three issue-QAs and two system-QAs.

Step 2: Exploit Solutions

Then we discover and instance a group of issue solutions for each elicited design issue. For example, there are two issue solutions for design issue 1, as shown in Fig.7 and 8:

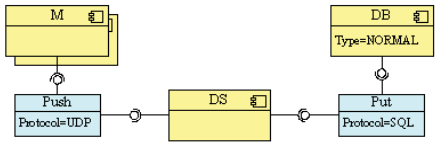


Fig. 7. Issue solution 1.1

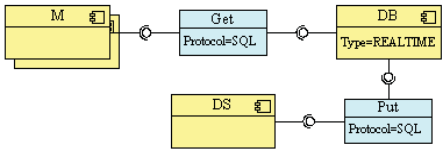


Fig. 8. Issue solution 1.2

Step.3: Identify Relations

(a) Solutions Relation

Firstly, relations between the issue solutions are identified, as shown in Table II.

Table II
Identified Relations between Issue Solution

Relation Type	*Issue Solution*
INCLUSIVE	(IS1.2, IS2.3)
GENERALIZED	(IS2.1, IS3.1), (IS2.2, IS3.2)
CONFLICTIVE	(IS1.1, IS2.3), (IS1.2, IS2.1), (IS1.2, IS2.2), (IS3.1, IS4.2)

For example, IS2.3 is part of IS1.2, so INC(IS1.2, IS2.3) is true; the DB in IS2.1 is a specialization of the DB in IS3.1, so GEN(IS2.1, IS3.1) is true; the DB in IS1.1 has different attribute value from the DB in IS2.3, so CON(IS1.1, IS2.3) is true.

(b) Solution-QA Relation

Secondly, relations between the issue solutions and quality attributes are identified. We implemented a *Solution Quality*

View, in which architects can evaluate each issue solution and determine related QA relation. Fig.9 illustrates the issue solution 1.1 and its corresponding QAs, including two issue QAs and two system QAs.

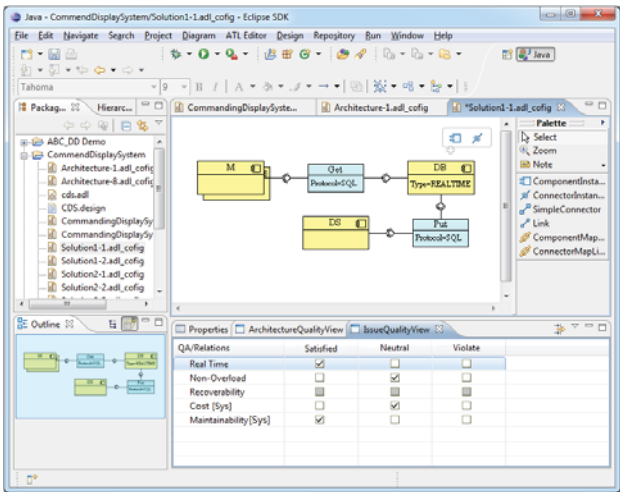


Fig. 9. Quality-View of Issue Solution 1.1

Step.4: QA-Based Solutions Synthesize

In this step, issue solutions and solution-QA relations will be synthesized to generate a set of feasible candidate combinations.

In this case study, the results are eight candidate architecture solutions, named $CA_1 \sim CA_8$. Limited by space, we only illustrate CA_7 in Fig.10.

Step.5: Recommend Candidate Architecture

(a) Specify Evaluation Criterion

We ordered these QAs and quantitatively set the weight to them.

$Real\text{-}Time = 0.3,$
$Non\text{-}overload = 0.2$
$Recoverability = 0.1$
$Maintainability = 0.2$
$Cost = 0.1$

Then, the quality weight vector can be denoted as:

$$QA = <0.3, 0.2, 0.1, 0.2, 0.1>$$

(b) Recommendation

Using this quality weight vector as a evaluation criterion, we can calculate the dot product of candidate architectures quality and weight vector as shown in Table IV.

Table IV
CANDIDATE ARCHITECTURE AND QUALITY VALUE

CA1	CA2	CA3	CA4	CA5	CA6	CA7	CA8
0.1	-0.2	0.9	0.5	0.8	0.7	**1.8**	1.3

We implemented another view to present this evaluation result. Fig.10 shows this view, in which architects can examine each feasible architecture and its various quality attributes. And the highlighted candidate architecture is the recommendation. In the case, CA_7 is highlighted.

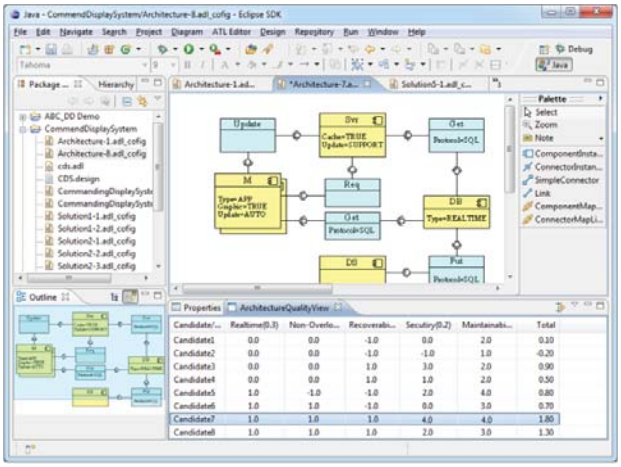


Fig. 10. Architecture Quality View of CA7

C. Discussion

This case study illustrates a simple but complete usage of our approach. From this experience, both effectiveness and limitations can be learned. We discuss about some of our finding in this section.

This approach provides the support in several activities to alleviate the difficulties. Our approach can relieve the difficulty of directly designing a complete architecture solution. When facing a number of requirements, architects need to explore every combination of the options and then present all of the resulting architectures for comparison. It will be a hard and error-prone work to accomplish. By applying the principle of "divide and conquer", we separate requirements into a set of *design issues*, and let architects concentrate on designing solutions for relatively simple problem. Then, with the help of solution synthesize algorithm, architects are able to rely on automated processes and ABC Tool that help him intelligently explore the possibility solution space, and provide a small set of humanly-intelligible recommendations for him to chose from at the end.

However, the limitation is that only the problem whose design issues are relatively independent is suitable for our approach. For the problems whose design issues are overlapping and tightly interrelated, are not quite suitable.

IV. RELATED WORK

The software architecture community has developed different methods to support reasoning about various quality attributes during architecture design and also support evaluation of final architecture's quality.

Bruin and Vliet introduce an approach named Quality-Driven Composition (QDC) [16]. It firstly derives the architecture that only addresses functional requirements and contains a number of variability points. The solution fragments addressing quality requirements are used to iteratively compose the architecture.

The ADD method [17] is proposed by CMU SEI. This method defines software architecture as a design process based on the quality attribute requirements. ADD follows a

recursive process that decomposes a system or system element by applying architectural tactics and patterns that satisfy its driving quality attribute requirements. ADD can be used with traditional and with agile software engineering methods.

Jansen et al. propose an approach named Archium [8]. They give a meta-model of software architecture decisions and they develop a tool to model design decision and architecture fragment separately. They use composition model as a glue to combine decision and design fragment.

Choi et al. introduce an integrated design-decision based architecture design approach, AQUA [10]. It has a three-step process includes finding, evaluating and changing decisions. They use a decision constraints graph as the most significant technique to support the entire process and guaranteed the quality.

Al-Naeem presents ArchDesigner [4] as a systematic approach for facilitating the architectural design. Using optimization techniques, it can determine the best combination of design alternatives that best satisfy stakeholders' quality goals and project constraints.

Bayesian-Belief Network based method (BBNbASM) [11] describes uncertain relations between quality attributes and design decisions by a formal model which can be used to predict the probability of architecture's quality.

Although the above approaches are proposed to support quality attributes to architecture design, they lack of enough support for design process. In contrast to these approaches, our approach offers a more complete support of guidance from requirement to target architecture design and support analysis of quality attributes as well.

V. Conclusions and future work

In this paper, we have proposed and implemented a decision-centric architecture design method which alleviates the complexity of architecture design . This approach provides pragmatic guidance to architects in the design process from architecturally significant requirements to candidate architecture solutions. It supports automated synthesizing candidate architecture and also recommends an optimal candidate, making the design process more efficient and effective.

This paper presents our preliminary work. So far, we have already identified a number of areas that call for further improvement.

(a) In current ABC/DD, we only use qualitative relations to describe the solution's impact on quality attributes. Three-valued score is not able to deal with the complexity of real problems. Although quantitatively measuring such impact surely will incur more burdens, there are many quality attributes which are more suitable to be quantitatively described. As a part of our ongoing work, we are trying to leverage fuzzy logic to tackle this problem and have achieved initial results.

(b) Furthermore, to enhance this approach, we are going to integrate a more sophisticated evaluation mechanism. A promising way is to leverage some existing research on quality evaluation of architecture.

(c) We do not validate the effectiveness of the recommendations. And we plan to produce some measures of the pertinence or an expert's independent opinion to compare with the results produced by our approach.

Acknowledgment

This effort is sponsored by the National Basic Research Program of China (973) under Grant No. 2009CB320703, the Science Fund for Creative Research Groups of China under Grant No. 60821003, and the National Natural Science Foundation of China under Grant No. 61073020. The authors want to express their gratitude to the anonymous reviewers for their valuable comments that helped to improve this work.

References

[1] M. A. Babar and P. Lago, "Design decisions and design rationale in software architecture," *Journal of Systems and Software*, vol. 82, no. 8, pp. 1195 – 1197, 2009, sI: Architectural Decisions and Rationale.

[2] R. N. Taylor., N. Medvidovic., and E. M. Dashofy, *Software Architecture: Foundations, Theory, and Practice*. John Wiley and Sons, Inc., 2010.

[3] M. Galster, A. Eberlein, and M. Moussavi, "Transition from requirements to architecture: A review and future perspective," in *Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, 2006. SNPD 2006. Seventh ACIS International Conference on*, 19-20 2006, pp. 9 –16.

[4] T. Al-Naeem, I. Gorton, M. A. Babar, F. Rabhi, and B. Benatallah, "A quality-driven systematic approach for architecting distributed software applications," in *ICSE '05: Proceedings of the 27th international conference on Software engineering*. New York, NY, USA: ACM, 2005, pp. 244–253.

[5] A. Jansen and J. Bosch, "Software architecture as a set of architectural design decisions," in *Software Architecture, 2005. WICSA 2005. 5th Working IEEE/IFIP Conference on*, 2005, pp. 109 –120.

[6] J. Bosch, "Software architecture: The next step," *Software Architecture*, vol. In EWSA, 3047 of LNCS, pp. 194–199, 2004.

[7] W. Bu, A. Tang, and J. Han, "An analysis of decision-centric architectural design approaches," in *SHARK '09: Proceedings of the 2009 ICSE Workshop on Sharing and Reusing Architectural Knowledge*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 33–40.

[8] A. Jansen, J. Van der Ven, P. Avgeriou, and D. Hammer, "Tool support for architectural decisions," in *Software Architecture, 2007. WICSA'07. The Working IEEE/IFIP Conference on*, 2007, pp. 4–4.

[9] A. Tang, Y. Jin, and J. Han, "A rationale-based architecture model for design traceability and reasoning," *Journal of Systems and Software*, vol. 80, no. 6, pp. 918–934, 2007.

[10] H. Choi, Y. Choi, and K. Yeom, "An Integrated Approach to Quality Achievement with Architectural Design Decisions," *Journal of Software*, vol. 1, no. 3, p. 40, 2009.

[11] H. Zhang and S. Jarzabek, "A bayesian network approach to rational architectural design," *International Journal of Software Engineering and Knowledge Engineering*, vol. 15, no. 4, pp. 695–717, 2005.

[12] R. de Boer, R. Farenhorst, P. Lago, H. Van Vliet, V. Clerc, and A. Jansen, "Architectural knowledge: Getting to the core," *Software Architectures, Components, and Applications*, pp. 197–214.

[13] L. Xu, H. Ziv, T. Alspaugh, and D. Richardson, "An architectural pattern for non-functional dependability requirements," *Journal of Systems and Software*, vol. 79, no. 10, pp. 1370–1378, 2006.

[14] G. Hohpe and B. Woolf, *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2003.

[15] P. Kruchten, P. Lago, and H. van Vliet, "Building up and reasoning about architectural knowledge," *Quality of Software Architectures*, pp. 43–58, 2006.

[16] H. de Bruin and H. van Vliet, "Quality-driven software architecture composition," *Journal of Systems and Software*, vol. 66, no. 3, pp. 269–284, 2003.

[17] L. Bass, P. Clements, and R. Kazman, *Software architecture in practice*. Boston, MA: Addison-Wesley, 2003.

Representation of Reference Architectures: A Systematic Review

Milena Guessi, Lucas Bueno Ruas Oliveira and Elisa Yumi Nakagawa
Department of Computer Systems
University of São Paulo - USP
PO Box 668, 13560-970, São Carlos, SP, Brazil
Email: {milena, buenorlo, elisa}@icmc.usp.br

***Abstract*—Software architectures play a major role in determining system quality, since they form the backbone to any successful software-intensive system. In this context, reference architecture is a generic architecture for a class of software systems used as a foundation for the design of concrete architectures from this class. Because of their fundamental role, reference architectures need to be properly represented in order to be effectively used, and understood by all stakeholders. In this perspective, we have found initiatives proposing different approaches to represent them. However, there is a lack of detailed overview about all these approaches. In this paper, we identify possibly all approaches used or proposed to represent reference architectures. For this, we have used a technique proposed by Evidence-Based Software Engineering (EBSE): the Systematic Review. As achieved results, we can observe different approaches adopted to represent reference architectures; however, there is no consensus about which are more adequate or their effectiveness. Based on this overview, interesting and important perspectives for future research can also be found.**

Keywords: Reference Architecture, Architectural View, Architectural Description Language, Systematic Review.

I. Introduction

Reference architectures have emerged as elements that aim at facilitating and systematizing the development of software systems. However, there is no commonly accepted definition for reference architectures [1]. In this work, we considered that a reference architecture aggregates knowledge of a domain, identifying abstract solutions of a problem and promoting reuse of design expertise by achieving solid, well-recognized understanding of a specific domain.

Particularly, representing reference architectures in order to make them understandable for a wide variety of stakeholders (such as customers, product managers, project managers and engineers) consists of an essential activity for their success. Despite the relevance of reference architectures, it has been observed that their representation, in practice, is often conducted informally. Additionally, there is a lack of works that summarize all approaches that have addressed an adequate representation of these architectures.

In another perspective, during the study of a new knowledge area, researchers usually conduct a bibliographic review (almost always an informal review) to identify publications related to a specific subject. However, this kind of review does not offer any support to avoid bias during the selection of the publications that will be analyzed. In this context, Evidence-Based Software Engineering (EBSE) [2] has investigated and proposed the use of the systematic review, an technique used to summarize, assess and interpret all evidences related to a specific question, topic area, or phenomenon of interest. In this context, an individual evidence (for instance, a case study or an experimental study divulged in a publication) which contributes to a systematic review is called *primary study*.

The main objective of this paper is to identify, summarize and analyze possibly all approaches that have been proposed or used to represent reference architectures. For this, we have adopted and conducted a systematic review[1]. Results have pointed out that this is a novelty topic of research and, considering the relevance of reference architectures in the software development, more effort must be focused on this research area. Besides that, we could identify future and important research lines based on our achieved results.

This paper is organized as follows. In Section II we present the conducted systematic review. In Section III we discuss achieved results. In Section IV we summarize our contributions and discuss perspectives for further work.

II. Systematic Review Application

Our systematic review was conducted in the reference architecture context, aiming at identifying all relevant primary studies related to the representation of this architecture. It was conducted from September 1st, 2010 to October 15th, 2010 and involved three people (one software engineering researcher, one systematic review specialist, and one undergraduate student). In order to conduct our systematic review, we followed the process proposed by Kitchenham [3]. In short, this process is composed by three main phases: (i) planning; (ii) conduction; and (iii) reporting. The phases

[1]Detailed information about our systematic review is available in a technical report found in http://www.icmc.usp.br/~biblio/BIBLIOTECA/rel_tec/RT_363.pdf

are explained in more details during presentation of our systematic review.

A. Planning

In this phase, the objectives and the systematic review protocol are defined. The protocol guides the conduction of the systematic review and is composed by research questions, search strategy, inclusion and exclusion criteria and data extraction and synthesis method.

1) Research Questions: Aiming to find all primary studies to understand and summarize evidences about proposed or used approaches to represent reference architectures, the following research question (RQ) was established:

- RQ1: Which are the approaches proposed or used to represent reference architectures?

Additionally, we defined three secondary research questions to specify the objectives of our search:

- RQ2: Which are the UML techniques proposed or used to represent reference architectures?
- RQ3: Which are the ADLs proposed or used to represent reference architectures?
- RQ4: What is the maturity level of the found approaches, considering their application to describe reference architectures?

2) Search Strategy: In order to establish the search strategy, considering the research questions, we initially identified the main keywords "Reference Architecture" and "Architectural Description". We also identified synonyms for these keywords. Thus, the final search string was: *("Reference Architecture" OR "Reference Model") AND ("Architectural View" OR "Architecture View" OR "Architectural Model" OR "Architecture Model" OR "Architectural Description" OR "Architecture Description")*. In particular, we add "Reference Model" in our search string, since some authors use this referring to reference architectures. Also, we only considered papers written in English, since it is the most common language in scientific papers. Finally, we established ACM Digital Library, IEEEXplore, ISI Web of Knowledge, ScienceDirect, Scopus, and SpringerLink as search sources (i.e., publication databases) to find the primary studies.

3) Inclusion and Exclusion Criteria: The selection criteria are used to evaluate each study recovered from the search sources. Thus, the inclusion criteria (IC) used to include relevant studies in our systematic review were:

- IC1: The primary study proposes or uses an approach to represent reference architectures; and
- IC2: The primary study proposes or uses an ADL to describe reference architectures.

Alternatively, the exclusion criteria (EC) used to exclude the studies that do not contribute to answer the research questions were:

- EC1: The primary study does not address reference architectures;
- EC2: The primary study does not propose or use any approach to represent reference architectures;
- EC3: The primary study does not present an abstract or its full text is not available;
- EC4: The primary study is written in a different language than English;
- EC5: The primary study is directly related to another primary study of the same author. In this case, only the most recent primary study will be considered; and
- EC6: The primary study consists of a compilation of work, for instance, from a conference or workshop.

4) Data Extraction and Synthesis Method: We plan to build data extraction tables related to each research question, when applicable.

B. Conduction

In this phase, the primary studies are identified, selected and evaluated according to the previously established protocol. To support the organization of the primary studies, we used JabRef[2], an open source reference management system. It is worth highlighting that only primary studies published until September 14, 2010 are considered in this paper.

As a result of our search in the publication databases using the search string, 110 primary studies were recovered. After the application of the inclusion and exclusion criteria, only 10 studies were considered relevant for our systematic review. Table I presents these studies. Following, a more detailed analysis was conducted on each primary study included.

TABLE I
INCLUDED PRIMARY STUDIES

Study	Authors	Year
S1	Bashroush et al. [4]	2005
S2	Chen et al. [5]	2010
S3	Fritschi and Gatziu [6]	1999
S4	Hofmann et al. [7]	2009
S5	Lin et al. [8]	2008
S6	Lopez and Blobel [9]	2009
S7	Meland et al. [10]	2009
S8	Nakagawa and Maldonado [11]	2008
S9	Sarathy et al. [12]	2010
S10	Schroth [13]	2008

C. Reporting

In this last phase, we present analytical results of our systematic review. The obtained data extraction and synthesis of knowledge considering each research question is discussed next.

[2]http://jabref.sourceforge.net

1) RQ1: Regarding RQ1 (i.e., approaches proposed or used to represent reference architectures), we have identified 10 different adopted approaches. Table II summarizes each of these approaches. Particularly, studies S4, S5, S9, and S10 represented the reference architecture using a graphic approach other than UML. Also, study S8 proposed an architectural view, the Conceptual View, which describes by means of ontologies each of the terms that are present in the reference architectures. It is worth noting that this diversity points out that there is no consensus about how to represent reference architectures.

TABLE II
SHORT DESCRIPTION OF THE APPROACH PROPOSED OR USED IN EACH PRIMARY STUDY

Study	Approach proposed or used in the primary study
S1	Architectural Description Language ADLARS
S2	Architectural Description Language π-ADL
S3	Architectural Description Language WRIGHT
S4	Model-View-Controller and Moderator Architectural Styles
S5	Layers Architectural Style
S6	Business, Information and Computational Architectural Views
S7	Component Architectural View and Information Model
S8	Module, Runtime, Deployment and Conceptual Architectural Views
S9	Logic and Physical Architectural Views
S10	Community, Process, Service and Infrastructure Architectural Views

2) RQ2: This research question addresses UML techniques proposed or used to represent reference architectures. Table III summarizes the found UML techniques. For instance, Lopez and Blobel [9] (study S6) adopted three architectural views: enterprise, information and computational. Then, the authors used, respectively, the use case diagram, the class diagram and the component diagram of UML to represent each of these views.

TABLE III
UML TECHNIQUES PROPOSED OR USED IN THE REPRESENTATION

Study	Architectural View	UML Technique
S6	Enterprise View	Use Case Diagram
S6	Computational View	Component Diagram
S6	Information View	Class Diagram
S7	Component and Connector View	Component Diagram
S7	Information Model	Class Diagram
S8	Deployment View	Deployment Diagram
S8	Module View	Class Diagram
S8	Runtime View	Component Diagram

The enterprise view offers a perspective of the system's architecture and environment, describing the system's purpose, scope and policies. The computational view shows the logical components and their interactions through interface. The information view, also referenced by study S7 as an information model, offers a perspective on the information's structure and its semantics. The runtime view proposed by Nakagawa and Maldonado [11] is also known as Component and Connector View (C&C) by the "Views and Beyond" method [14]. The C&C view shows the system as a set of cooperating units of runtime behavior and was also proposed in study S7. The deployment view describes the machines, software that is installed on those machines and network connections that are used by the software systems. Finally, the module view shows how a software system is structured as a set of implementation units.

3) RQ3: This research question addresses the ADLs proposed or used to represent reference architectures. In spite of the diversity of ADL available in the literature, there are few initiatives of using ADL to represent reference architectures. Bashroush et al. (study S1) developed the ADLARS (Architecture Description Language for Real-time Software Product Lines), an ADL to be used in the software product line context. This ADL aims at supporting the relationship between the system's feature model and the architectural structures. The feature model encompasses the variability and commonality among the different products within the scope of a family. Thus, the link between the feature model and the architectural structures allows for automatic generation of product architectures from the family reference architecture, making the task of deriving product-specific architectures much more straightforward.

Chen et al. (study S2) applied π-ADL to describe the High-Level Architecture (HLA), which is a reference architecture and a common infrastructure for large scale distributed interactive simulation systems, accepted as an IEEE standard in 2000 [15]. In π-ADL, architectures are composite elements representing systems. In this sense, a composite consists of external interfaces and internal behavior. Therefore, π-ADL supports the description of composite dynamic behaviors. Additionally, the virtual machine for the system dynamic analysis, called π-ADLVM, can be used to test the architectural models written in π-ADL.

Fritschi and Gatziu (study S3) represented the reference architecture by means of WRIGHT [16]. This language presents the basic architectural abstractions of components, connectors and configurations, providing explicit structural notations for each of these elements. Particularly, the configuration is a collection of component instances combined via connectors.

4) RQ4: This research question addresses the maturity level of the approaches, considering their application to describe reference architectures. In order to answer this question, we classified each study in one of the following categories: (i) proposal, if the primary study proposes an approach to represent reference architectures, but does not present any case study; (ii) case study, if the primary study

presents at least one case study; and (iii) application, if the approach is already been used in a project or in the development industry. Table IV shows the maturity level of each approach. It is observed that only two studies presented a case study. In this sense, it seems interesting to concentrate efforts to increase the maturity level of these approaches and to verify if they are adequate for the representation of reference architectures.

TABLE IV
MATURITY LEVEL OF THE FOUND APPROACHES

Maturity Level	Total	Percentage	Primary Studies
Proposal	4	40%	S1, S4, S5, S10
Case study	2	20%	S2, S8
Application	4	40%	S3, S6, S7, S9

III. DISCUSSION

Results of our systematic review point out that, in spite of relevance of reference architectures to the software development, there are few recent works that show concern with their adequate representation. Therefore, more studies are necessary that investigate which are and how other, if any, architectural views can be used to properly describe reference architectures. Also, investigation of well-known graphic notations, such as UML, would be relevant, aiming at standardizing the representation of such architectures. Additionally, the use of ADLs in the context of reference architectures should be further investigated concerning their application and suitability.

Considering knowledge arisen from this work, it is possible to identify interesting and important research lines that can be investigated in future work. For instance, investigation or proposition of software tools to support the representation and modeling of reference architectures; conduction of case studies in order to increase the maturity level of the found approaches; investigation of how well-known graphic notation could be applied in order to represent different detail levels of reference architectures; and, establishment of a consensus of the best approaches to the representation of reference architectures.

Regarding limitation of this work, other research sources could be selected, such as Google Scholar, and other keywords could be added to the search string. Moreover, it is worth highlighting that systematic review conduction is not a trivial task because of the amount of papers that need to be manipulated. Besides that, relevant primary studies written in languages other than English were overlooked in this instance.

IV. CONCLUSIONS

The main contribution of this work was to present a detailed panorama about proposal and use of approaches for the representation of reference architectures. Based on this panorama, another important contribution is to bring forth new research lines on this topic. As main result, we found that the representation of reference architectures has not been sufficiently explored and there are still different perspectives that could be investigated in order to deal adequately with reference architectures, aiming at promoting their dissemination and effective use.

Acknowledgments This work is supported by Brazilian funding agencies: FAPESP, CNPq and Capes.

REFERENCES

[1] S. Angelov, P. Grefen, and D. Greefhorst, "A Classification of Software Reference Architectures: Analyzing Their Success and Effectiveness ," in *ECSA'09 at WICSA'09*, Cambridge, UK, 2009, pp. 141–150.

[2] T. Dybå, B. Kitchenham, and M. Jorgensen, "Evidence-based software engineering for practitioners," *IEEE Software*, vol. 22, no. 1, pp. 58–65, 2005.

[3] B. A. Kitchenham, "Procedures for performing systematic reviews," Keele University and National ICT Australia Ltd, Tech. Report TR/SE-0401 and NICTA TR 0400011T.1, 2004.

[4] R. Bashroush, T. J. Brown, I. Spence, and P. Kilpatrick, "ADLARS: An Architecture Description Language for Software Product Lines," in *SEW'05*, Maryland, USA, 2005, pp. 163–173.

[5] J. Chen, D. Wu, J. Zhang, and F. Oquendo, "Formal modelling and analysis of HLA architectural style," *Int. Journal of Modelling, Identification and Control*, vol. 9, no. 1-2, pp. 71–82, 2010.

[6] H. Fritschi and S. Gatziu, "A Reusable Architecture to Construct Active Database Systems," University of Zurich, Tech. Report ifi-99.02, 1999.

[7] C. Hofmann, N. Hollender, and D. W. Fellner, "Workflow-Based Architecture for Collaborative Video Annotation," in *OCSC'09 at HCII'09*, San Diego, USA, 2009, pp. 33–42 (LNCS v.5621).

[8] C. Lin, S. Lu, Z. Lai, A. Chebotko, X. Fei, J. Hua, and F. Fotouhi, "Service-Oriented Architecture for VIEW: A Visual Scientific Workflow Management System," in *IEEE SSC'08*, vol. 1, Honolulu, USA, 2008, pp. 335–342.

[9] D. M. Lopez and B. G. Blobel, "A development framework for semantically interoperable health information systems," *Int. Journal of Medical Informatics*, vol. 78, no. 2, pp. 83–103, 2009.

[10] P. H. Meland, S. Ardi, J. Jensen, E. Rios, T. Sanchez, N. Shahmehri, and I. A. Tondel, "An architectural foundation for security model sharing and reuse," in *ARES'09*, vol. 1–2, Fukuoka, Japan, 2009, pp. 823–828.

[11] E. Y. Nakagawa and J. C. Maldonado, "Reference architecture knowledge representation: an experience," in *SHARK'08 at ICSE'08*, Leipzig, Germany, 2008, pp. 51–54.

[12] V. Sarathy, P. Narayan, and R. Mikkilineni, "Next Generation Cloud Computing Architecture: Enabling Real-Time Dynamism for Shared Distributed Physical Infrastructure," in *WETICE'10*, Larissa, Greece, 2010, pp. 48–53.

[13] C. Schroth, "A Service-oriented Reference Architecture for Organizing Cross-Company Collaboration," in *Enterprise Interoperability III*. Springer, 2008, pp. 71–83.

[14] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, R. Nord, and J. Stafford, *Documenting Software Architectures: Views and Beyond*. Addison-Wesley, 2003.

[15] *IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)*, IEEE Std. 1516-2000.

[16] R. J. Allen, "A Formal Approach to Software Architecture," Carnegie Mellon University, Tech. Report CMU-CS-9-144, 1997.

A Model-View-DynamicViewModel and its Performance in a Web-based Component Architecture

Graeme Baillie, Brian Armour, Dave Allan, Robert Milne
CC Technology Ltd
Glasgow, Scotland
{graeme.baillie, brian.armour, dave.allan, robert.milne}@cctechnology.com

Thomas M Connolly, Richard Beeby
School of Computing
University of the West of Scotland
Paisley, Scotland
{thomas.connolly, richard.beeby}@uws.ac.uk

ABSTRACT

The implementation of an appropriate software architecture is crucial in achieving the optimum performance from a system. Web-based applications are becoming increasingly popular in replacing previous Windows-based applications. This has lead to a growth in new technologies and architectures to cope with the new workload and performance demands that web-based applications can require.

This paper will look at a current software company that provides process-heavy web-based application systems providing grant management services to medical and academic research. The purpose of this paper is to analyze the performance of their existing Web Forms based management system against newly developed dynamic component-based architecture using a variant of the Model View Controller (MVC) and Model-View-ViewModel (MVVM) patterns, which we have termed MVDVM (Model-View-DynamicView Model). Accurate and meaningful results will demonstrate how applications can now benefit from this approach.

Keywords – Component based architecture, Dynamic forms, Interoperability, Model View Controller.

I. INTRODUCTION

Web-based software applications are becoming more popular and crucial to the running of businesses in their day-to-day processes. Due to the growth of the world wide web and of Internet services capabilities, software companies are in a much better situation to offer services that previously would have only been possible through an offline platform.

Improvements in Internet connectivity and hardware performance within the last ten years has meant that software developers no longer have to worry as much about bandwidth, processor or memory requirements when implementing process-heavy applications. Although this has been beneficial to the software industry it has lead to many systems being implemented in ways that do not utilize best performance techniques available. This can deliver sub-optimal user experience and utilizes greater server capacity than is necessary at a time when datacenter costs are rising.

This paper will look at a software company that implements process-heavy web applications. The company's current working practice and techniques being used will be analyzed. The techniques will then be compared against a new dynamic component-based solution to identify the change in performance achieved for one of its key software products.

Statistics such as page load time, page size, number of database queries will be measured under varying loads and analysed to assess how these will benefit by applying latest thinking and best practice when developing high-performance software systems

In the next section, a brief overview of the product is provided followed by a discussion of related research. The subsequent sections will discuss the technology platform, the test platform and then present the performance data of the old product and the re-architected product.

II. CASE STUDY OVERVIEW

This paper analyzes a component-based dynamic solution used to manage application forms within CC Grant Tracker, a Grants Management software product. The solution builds and publishes forms, which are then used for the completion, submission and ongoing management of applications. CC Grant Tracker is a solution from CC Technology, a leading global supplier of advanced grant management software solutions, based in Glasgow, Scotland. CC Grant Tracker supports organizations such as charities, academic institutions, public funding bodies and corporate foundations who often employ a regular cycle of grant applications, where professional reviewers decide which applications are worthy of funding. For a large organization, there may be hundreds or even thousands of applications, as well as dozens of reviewers, not to mention stringent auditing requirements. In such cases, simply administering the application process is a major undertaking. CC Grant Tracker manages the full life cycle of grant administration, from initial application through evaluation, approval and ongoing management.

When CC Technology originally created the software product the emphasis of the initial analysis and design phase was based on functionality rather than the performance of the system. The application is heavily web forms-based and the initial version was delivered using the standard web development tools from Microsoft – ASP.NET Web Forms. This delivered a tightly-coupled solution with associated maintenance issues. As the number of clients and the breadth of their requirements have increased the requirement to develop the flexibility and performance of the application forms has become a priority.

CC Technology have been using a re-architected solution with a number of new large clients and the new system has been extremely well received.

III. RELATED RESEARCH

A. Web Forms vs MVC

Microsoft currently endorses two web architectures, Web Forms, which was released in 2002, and the Model-View-Controller (MVC), which was released in 2009. MVC is a standard design pattern that Microsoft has adopted. The MVC pattern helps deliver solutions that separate the different aspects of the application (input logic, business logic and UI logic), while providing a loose-coupling between these elements [1]. Both architectures provide benefits to the developer and end-user depending on the circumstances. For this study we examined both approaches to obtain a better understanding as to which provides the better performance benefits.

1) ***Web Forms*** – The major advantage of a Web Forms implementation is that it allows an application to be developed in a relatively short period of time as there is lower initial design requirements for the solution architecture. However, research and experience shows that there are a number of performance disadvantages inherent to this approach.

Web Forms allow only one form tag to be added per page. This means that in order for the page to get information to or from the server, the entire page must be posted back. A common approach to address this uses Microsoft's Ajax Control Toolkit and update panels. This reduces performance problems related to sending unnecessary data back to the server however the Ajax update methods can be just as performance heavy in bandwidth and processing on the server due to the View State that is required on all updates from a Web Form regardless of a full postback or asynchronously.

The View State holds session information for each page [2]. This adds considerably to the data size of the page and as this is exchanged in all updates this impacts the bandwidth for the solution. Many of the built-in ASP.NET components make heavy use of View State, so it is not unusual for a page to have 10's of Kbytes of View State [3]. This is measured in terms of performance for end-users and bandwidth costs at the data centre. There is therefore no performance gain in using Ajax update panels throughout the application as the net effect can be the same or greater than doing a full postback.

A further serious issue for modern web pages is the abstraction from HTML, as this hinders accessibility, browser compatibility, and integration with JavaScript frameworks like JQuery and PrototypeJS [4]. Of lesser importance to this case study, the postback model makes it harder for search engines to rank ASP.NET pages high.

2) ***MVC*** – The Model-View-Controller (MVC) architectural pattern was first proposed by Trygve Reenskaug in the late 1970s (the earliest source available is [5] but MVC was not publicly documented until 1988 [6]). It was first used in Smalltalk [7] and is an effective approach for supporting multiple presentations of data. Users can interact with each presentation in a style that is suitable to the presentation. The data to be displayed is encapsulated in a model object and each model object may have a number of separate view objects associated with it, where each view is a different display representation of the model. Each view has an associated controller object that handles user input and device interaction. MVC was devised to target desktop applications, but because of its relatively loose formulation it was easily adapted for the Web. MVC (or Model 2 using the Sun Microsystems terminology [8]) is now used widely in web frameworks (e.g., ASP.NET, Django, Ruby on Rails, Code Igniter, Apache Struts, JavaServer Faces). In Web MVC (which differs somewhat from traditional MVC), a model encapsulates application data in a way that is independent of how that data is rendered by the application. The view accepts some number of models as input and transforms them into appropriate output that will be sent to the browser. The controller connects the models and views, typically by gathering model data and sending it to the view for rendering. The view accepts data as input and produces a string as output, which may include information about its type (e.g., HTML, XML or JSON). After being manipulated by some post-processing filters, the string is sent directly to the browser. Because the view's output is treated as an opaque string, it is difficult for the framework to reason about the structure of the content. These views are complicated by the need to provide both Ajax and non-Ajax versions of a site [9]. In ASP.NET, MVC is implemented as shown in Figure 1(a).

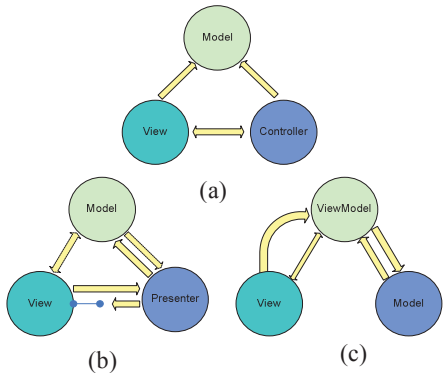


Figure 1: MVC Architectures: (a) Model-View-Controller; (b) Model-View-Presenter; (c) Model-View-ViewModel

In addition to managing complexity, the MVC pattern makes it easier to test applications than it is to test Web Forms. For example, with Web Forms a single class is used both to display output and to respond to user input, which when testing has to be instantiated along with all its child controls and additional dependent classes. Moreover, tests in Web Forms require a Web server. In contrast, the MVC framework decouples the components and makes heavy use of interfaces, which makes it possible to test individual components in isolation from the rest of the framework. The loose coupling between the three main components of an MVC application also promotes parallel development. For instance, one developer can work on the view, a second developer can work on the controller logic, and a third developer can focus on the business logic in the model.

A variant of MVC is the Model-View-Presenter (MVP) pattern [10], as shown in Figure 1(b). In MVP, the Presenter has the same responsibilities as MVC's Controller, acting as a mediator between the view and the model. It receives user input requests from the view and evokes changes on the model in response. The Presenter is able to query the view for data but to enforce separation of concern (SoC), the presenter is decoupled from the view by holding a reference to the view's interface rather than its implementation [11] [12]. Fowler describes a different approach for achieving SoC called the Presentation Model [13]. This pattern is similar to MVP in that it separates a view from its behavior and state and introduces a Presentation Model that is an abstraction of a view.

A further variant of MVC is the Model-View-ViewModel (MVVM) design pattern for Microsoft's Windows Presentation Format (WPF) [14], as shown in Figure 1(c). The MVVM pattern is a more specific version of the Presentation Model pattern [21]. MVVM also separates the view from the logic. However, unlike Controllers in MVC (Figure 1a) and Presenters in MVP (Figure 1b), an MVVM ViewModel has no awareness that a view even exists [15]. The ViewModel is an abstraction of the view but does not need a reference to the view like MVP does. The view uses the ViewModel as a data context and binds properties to fields in the ViewModel, providing a very loosely coupled design. This separation allows a graphics design team to focus on the view while a software development team can focus on implementing a stable and good ViewModel. As the ViewModel does not have to have a reference to the view, the logic can be tested without the view and it is also easy to make different views for GUI evaluations and compare them without changing the ViewModel. Note, Esposito and Saltarello [16] regard MVVM as the same pattern as Fowler's Presentation Model.

Zeller and Felton [9] introduce a stateless, framework-agnostic web application development style, which they call SVC (Selector-based View Composition) (SVC). With SVC, a developer defines a web page as a series of transformations on an initial state where each transformation consists of a *selector* (used to select parts of the page) and an *action* (used to modify content matched by the selector). SVC applies these transformations on either the client or the server to generate the complete web page. The authors contend this approach has two advantages: (i) SVC can automatically add Ajax support to sites, allowing developers to write interactive web applications without writing any JavaScript; (ii) developers can reason about the structure of the page and produce code to exploit that structure, increasing the power and reducing the complexity of code that manipulates the page's content.

There are many other approaches to SoC. Presentation-Abstraction-Control (PAC) [17] in which the UI is formed by a tree of agents for each of the three components (Presentation, Abstraction and Control). In Naked Objects [18], a UI is automatically generated from the model by analyzing the model interface using Java's reflection features. As a result, the model must contain all of the application domain logic. On the other hand, in Visual Proxy [19] the model/view separation is abandoned. While the model and view objects are still separate objects, they are implemented within the scope of a single class. The model and view layers are tightly coupled, rather than decoupled. Holub argues that the purpose of a model is to provide services to the view and if they are separated, the model cannot provide all the services for the view.

IV. TECHNOLOGY PLATFORM

CC Technology has a proven history in the use of Microsoft development tools and therefore it was logical to deliver the original system using ASP.NET Web Forms which was the prevalent Microsoft technology in 2004 when the solution was implemented. The Microsoft Ajax Control Toolkit was implemented to limit the amount of postbacks needed to the server. For database connections a custom ADO.NET entity generator connects to a SQL Server database. This platform has worked well, the technologies interact successfully and developers can develop and maintain a web-based solution without having detailed low-level web knowledge.

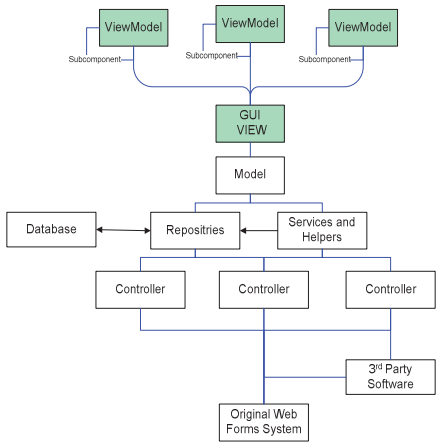


Figure 2: New MVDVM Architecture

The new component-based approach uses what we have named an MVDVM (Model-View-DynamicViewModel) architecture. The MVDVM architecture is an MVC-based architecture that is heavily influenced by the MVVM (Model View View-

Model) architecture, as Figure 2 shows. It contains a set of view-models and handles each one in a similar way. The major difference between the MVVM and MVDVM architectures is that the latter model dynamically builds and persists the view-models at runtime rather than having statically designed view-models, as in MVVM. The architecture is extremely powerful because of the ability to have encapsulated components (view-models) that persist themselves, thereby requiring no code to save and load components, producing an architecture that is extremely extensible. Each component that is added has no dependency or relationship to other components, allowing a separation of concerns and removing the risk of breaking other components when introducing new ones. By having this ability the only development that has to occur is writing new view-models. The view-models' logic and behaviour are captured in the view, which means that they can be updated or created without the need to compile. This in turn allows role-based logic to be performed on smaller subcomponents than would usually be possible. The view-models can be any user control such as text boxes, drop down lists or more complex composite user controls with contained business logic. A component can also seamlessly interact with other non-MVDVM systems and other databases meaning that saving and loading is not limited to a component's own data. The architecture allows the user to specify at runtime HTML to render each individual component to multiple document types to allow hard copies of forms. The difference between what the MVDVM architecture offers from other form designer tools and architectures is that it has no bounds or limitations as to what the components can do, other than what the .NET framework offers. The entire framework is self contained, which allows seamless integration with other architectures and systems. The scalability of controls and ease of extensibility make this architecture a good alternative to the web forms implementation.

The architecture is also advanced through use of client-side logic in JQuery components. As JQuery is now part of Microsoft's release with MVC, the integration in the development environment is seamless and therefore easy to use. Having JQuery allows quicker and more efficient behaviour and postbacks than was previously possible.

The final major change to the technologies used was the move away from the company's own entity generation tool for Database access. A repository pattern was implemented into the software so that at any time we could change the database connection technique without consequence. Initially Microsoft's Entity framework was evaluated. However, due to the early version of the software it was decided that the more established and Nhibernate model should be used.

V. Test Platform

To ensure accurate measurement of the performance of the original Web Forms and the new MVDVM architecture a common set of tests were developed. The tests were performed under the same situations and with the same metrics. A number of standard web pages in each system were tested with single users and also with increasing number of users (load testing). Both tests resulted in page load speed, number of pages per second and database information being obtained.

An accurate and consistent set of results was achieved by running a dedicated environment comprising a LAN, a test client containing load testing software, a web server running an instance of the forms software and a database server. To ensure that hardware performance was not compromising the results, the memory and CPU usage were monitored on each computer to ensure they stayed within 60% utilization. MS Visual Studio 2010 Ultimate Edition Load testing suite was used to generate relevant graphs and performance data.

VI. Server Performance

Individual tests were carried out on both architectures. The test cases used pages on each system with the same function and content, albeit with different implementations.

Area	Page Load Time (Average(seconds))	Time to receive first byte (seconds)
Opening a form	0.413	0.145
Financial page	1.361	1.113
Standard page	0.372	0.030
Saving form (including redirect)	0.635	0.588

Figure 3: MVDVM Page Times (in seconds)

Area	Page Load Time (Average)	Time to receive first byte
Opening a form	1.526	0.940
Financial page	1.556	0.599
Standard page	1.744	0.668
Saving form (including redirect)	0.656	0.114

Figure 4: Web Forms Page Times (in seconds)

The performance results in Figures 3 and 4 clearly show that the MVDVM architecture is faster on an individual load. It must be noted that there is a change in logic between the two solutions. The MVDVM application forms bind the models dynamically on every page load. Thus for every load and page exit data is read and written to the database. The Web Forms

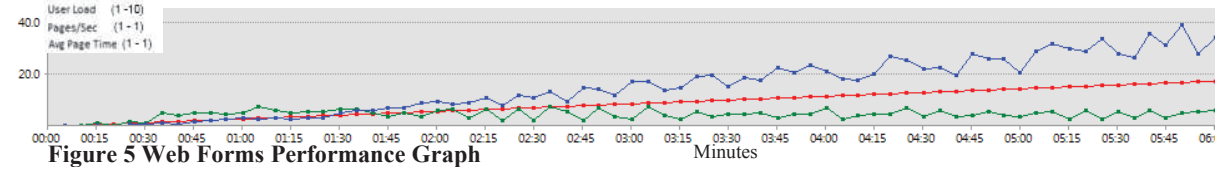


Figure 5 Web Forms Performance Graph

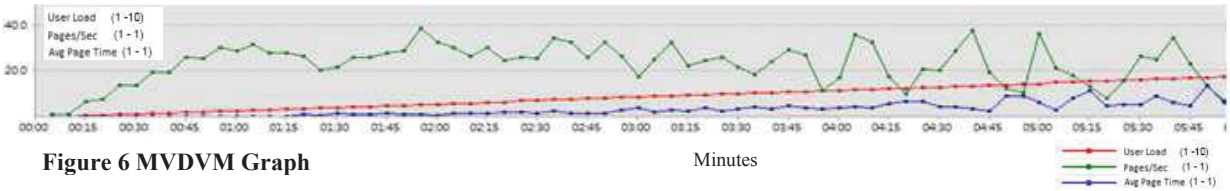


Figure 6 MVDVM Graph

solution does not write to the database until a user hits submit, which may be after n*page visits. This is reflected in the load times for the financial page. This is a complex page where hundreds of requests are being sent to the database. Although the MVDVM is still faster for this page the number of connections is the reason the difference is not as substantial.

The single tests are a good indication of the efficiency of each architecture. However, live systems are characterized by uneven and high loads. For a production system the number of concurrent users could be in the hundreds. Load testing was carried out on the same pages as noted in Figures 3 and 4. The resulting graphs are shown in Figures 5 and 6 (Vertical axis represents number of users, number of pages per second and average page load per second and all using corresponding ranges outlined in figures 7 and 8) . The graphs show that the MVDVM out performs the Web Forms by a large margin. Each test began with one simulated user running a web test. This proceeded through each page in a defined order. Every ten seconds five more users were added to the current list of users. The results are shown in Figures 7 and 8 for each run of the test.

	Range on graph	Min	Max	Avg
User load (num users)	1000	1	176	88
Pages/sec (seconds)	10	0	7.8	4.53
Ave page load (seconds)	100	0.25	39.7	16.2

Figure 7: Web Form Graph Averages

	Range on graph	Min	Max	Avg
User load (num users)	1000	1	176	88
Pages/sec (seconds)	10	1	39	23
Ave page load (seconds)	100	0.0.35	13.9	3.04

Figure 8: MVDVM Graph Averages

Figure 5 shows that the average page load speed increases in relation to the number of users. As the number of users increases the number of pages loaded per second severely decreases.

The MVDVM results are quite different. The pages per second start very high and remain so throughout the test. At the points of lowest performance where the graph dips the response times are still acceptable for an end-user. These dips occur where a large number of users (more than 100 users) tried to call an action concurrently. This is a quirk of using a test suite and the phenomena are unlikely to occur in production at this usage level. Usually in this case it would be the hardware performance that would fail or cause performance issues. The page load speed also remains very constant and, apart from the dips, remains lower than one second for each page throughout the test.

VII. Client Performance

Client performance can be characterized by the download time for the pages and of the rendering time.

Figure 9 shows the details for an average page in the performance tests. The details emphasize the earlier comments about how large or inefficient the HTML can become due to the View State and the ASP.NET automatically generating HTML. The Javascript size is also very high as the previous forms were rendering pure Javascript. The Javascript was not compressed, which is also inefficient, causing larger page sizes.

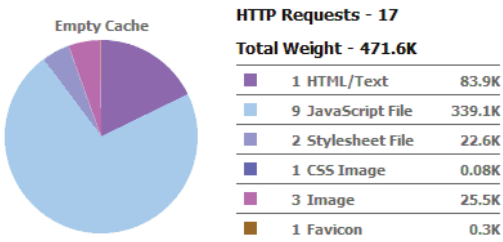


Figure 9: Web Forms Average Page Detail

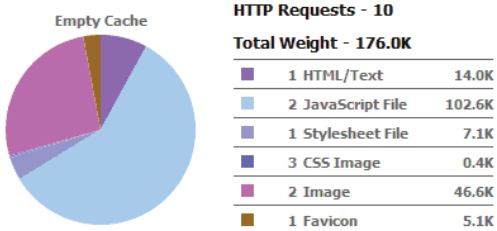


Figure 10: MVDVM Average Page Detail

The detail of an average MVDVM page in Figure 10 shows that the page size is smaller. The ability to control the HTML and use JQuery to write minimal amounts of code has resulted in a reduction of the page size from 471k to 176k.

The HTML code in the Web Forms output is controlled by ASP.NET and is not standards-compliant, whereas the MVDVM output is controlled by the developer. This control allows the developer to address client performance issues, which Google describe as 5 performance factors [20]: (i) optimizing caching, (ii) minimizing round-trip times, (iii) minimizing request overhead, (iv) minimizing payload size, (v) optimizing browser rendering. The MVDVM solution improves on each of these factors through the communications and page content.

VIII. Database Performance

A statistical analysis of the database transaction times compares the performance between the two architectures. The results are split into two parts: using all the data and using only the data that had duration times greater than zero. The results show every query that happened when the tests were run. Because of this there are many light weight queries that had durations of less than 1 ms, these show as 0ms durations.

A. Analysis using all of the data

Using the Web Forms architecture, reads were performed an average of 48.60 times (SD = 1061.05) with a range of 0 to 32,350 whereas using the MVDVM architecture reads were

performed an average of 24.54 times (SD = 283.12) with a range of 0 to 6,440.

Using the Web Forms architecture writes were performed an average of 0.0145 times (SD = 0.30) with a range of 0 to 17 whereas using the MVDVM architecture writes were performed an average of 0.0093 times (SD = 0.20) with a range of 0 to 22.

Transactions using the Web Forms architecture had an average duration of 1.01 ms (SD = 48.06) with a range of 0 to 9,157 ms where transactions using the MVDVM architecture had an average duration of 0.1223 ms (SD = 2.04) with a range of 0 to 93 ms.

Paired samples t-tests indicated that the MVDVM architecture performed significantly less read commands ($t(65,527) = 5.607$, $p < 0.000$), significantly less write commands ($t(65,527) = 3.733$, $p < 0.000$) and took significantly less time to perform read and write commands ($t(65,527) = 4.729$, $p < 0.000$) than the Web Forms architecture.

B. Analysis using only the data that had duration tims greater than zero

Using the Web Forms architecture reads were performed an average of 2,047.19 times (SD = 6,955.46) with a range of 0 to 32,350 whereas using the MVDVM architecture reads were performed an average of 990.40 times (SD = 1,664.82) with a range of 0 to 6,440. Using the Web Forms architecture writes were performed an average of 0.4087 times (SD = 1.63) with a range of 0 to 17 whereas using the MVDVM architecture writes were performed an average of 0.1095 times (SD = 0.879) with a range of 0 to 22.

Transactions using the Web Forms architecture had an average duration of 47.08 ms (SD = 324.75) with a range of 0 to 9,157 ms where transactions using the MVDVM architecture had an average duration of 5.70 ms (SD = 12.74) with a range of 0 to 93 ms.

Paired samples t-tests indicated that the MVDVM architecture performed significantly less read commands ($t(1,406) = 5.339$, $p < 0.000$), significantly less write commands ($t(1,406) = 5.996$, $p < 0.000$) and took significantly less time to perform read and write commands ($t(1,406) = 4.764$, $p < 0.000$) than the Web Forms architecture.

IX. CONCLUSION

This paper has presented a comparison of two web architectures used in process-heavy web applications. The first, Web Forms, is an established standard Microsoft approach and the second, MVC, is a relatively recent approach gaining some acceptance in the development community. This paper has proposed a variant of MVC, called MVDVM and has demonstrated that this architecture delivers performance benefit over the traditional Web Forms based architecture. The performance has been shown to be improved at the database, web server and client. With each of the tests it has been shown to be faster or more efficient.

The results of the case study have proven that under these circumstances the performance benefits are large and the move to an MVC architecture are justified.

X. ACKNOWLEDGEMENTS

This project received financial support from the Knowledge Transfer Partnerships programme (KTP). KTP is funded by the Technology Strategy Board.

XI. REFERENCES

[1] G. Krasner, and S. Pope. A description of the model-view-controller user interface paradigm in the smalltalk-80 system. Journal of Object Oriented Programming, Vol 1, Issue 3, pp. 26–49, 1988.

[2] D. Esposito. The ASP.NET View State, MSDN Magazine, February 2003.

[3] S. Mitchell. Understanding ASP.NET View State, http://msdn.microsoft.com/enus/library/ms972976.aspx, May 2004.

[4] D. Esposito. Comparing Web Forms and ASP.NET MVC, MSDN Magazine, July 2009.

[5] Reenskaug, T., *Thing-Model-View-Editor*, Xerox Parc, 1979.

[6] Krasner, G. and S. Pope. A cookbook for using the Model-View-Controller user interface paradigm in Smalltalk-80. Journal of Object Oriented Programming, 1988, Vol 1, Issue 3, pp. 26-49.

[7] A. Goldberg, and D. Robson. Smalltalk-80: The Language and Its Implomentation. Reading, MA: Addison-Wesley, 1983.

[8] G. Seshadri. Understanding JavaServer Pages Model2 Architectire. JavaWorld, December 1999.

[9] W. P. Zeller, and E.W. Felten. SVC: Selector-based View Composition forWeb Frameworks, Proceedings of the 2010 USENIX conference on Web application development, 20-25 June 2010, Boston, MA, USA.

[10] M. Potel. MVP: Model-View-Presenter - The Taligent Programming Model for C++ and Java. Taligent Inc. 1996

[11] J-P. Boodhoo. Design Patterns: Model View Presenter. *MSDN Magazine*, August 2006.

[12] G. Hall. Pro WPF & Silverlight MVVM: Effective Application Development With Model-View-ViewModel (Expert's Voice in WPF)**.** Apress Academic. 2010

[13] M. Fowler. Presentation Model, Essay, July 2004.

[14] J. Smith. WPF Apps with the Model-View-ViewModel Design Pattern. MSDN Magazine, February 2009.

[15] S. Sanderson. Pro ASP.NET MVC 2 Framework (2nd edition). Apress Academic. 2010

[16] D. Esposito and A. Saltarello (2009). Microsoft .NET: Architecting Applications for the Enterprise. Microsoft Press.

[17] J. Coutaz. PAC: an Object Oriented Model for Dialog Design. IFIP Interact, 1987, ppp. 431-436.

[18] R. Pawson. Naked Objects. PhD Thesis. Trinity College Dublin. 2004

[19] A. Holub, Building User Interfaces for Object Oriented Systems. Javaworld, September, 1999 (Available at: http://www.javaworld.com/javaworld/jw-09-1999/jw-09-toolbox.html)

[20] Google, http://code.google.com/speed/page-speed/docs/rules_intro.html, 2011.

[21] J. Follesø. Model-View-ViewModel, in Proceedings of Norwegian Developers Conference, Oslo, Norway, June 17-19, 2009

Analysis of the continuity of software processes execution in software organizations assessed in MPS.BR using Grounded Theory

Carlos Diego Andrade de Almeida
UNIFOR, IVIA
Fortaleza, Brazil
Email: carlosdiegoa@gmail.com

Thiago Crystyan Macedo
UNIFOR
Fortaleza, Brazil
Email: crystyanjlc@gmail.com

Adriano Albuquerque
UNIFOR
Fortaleza, Brazil
Email: adrianoba@unifor.br

Abstract— **Many companies look for in a implementation of a Improvement Program based on software maturity models as a way of increasing their product's quality. For knowing their maturity level, they have to submit themselves for an assessment based in models. To be assessed, a company spends some time into the definition and implementation of the necessary software processes. However, after the assessment most of the companies have some difficulties to continue the execution of some activities of the processes or leave this execution. This paper proposes the analysis of the continuity of software processes into the companies assessed in MPS.BR, the Brazilian Maturity Model, identifying factors that help or make difficult to maintain the adherence of those process to the model and the easiest practices to be maintained in execution, after the company has been assessed. With this work, we hope to make easier for consultants to implement the processes, specially those that have more percentage of leaving or difficulties.**

Keywords-Software processes; Maturity Model

I. INTRODUCTION

The software's importance grows every day and as the quality of the software products is so closely related to the quality of processes that are used to develop them, invest in the processes quality became very important to the software organizations [14].

Among these, the MPS.BR, the Brazilian Maturity Model, fits perfectly into a proposal of improving the quality of software produced in Brazil. According to [10], the program MPS.BR, launched in December 2003, aimed at improving software processes of micro, small and medium size companies.

The reference model, MR.MPS, contains the requirements that the processes of organizational units must meet to comply with the model.

However, like others maturity models, after the assessment companies have trouble to keep their processes adherent to the reference model.

Therefore, it is important to know the main difficulties to maintain such compliance.

This paper presents the results of a survey carried out with professionals involved at MPS.BR, which the objective was to find out the factors that influence positively in the continuity of the processes execution and the practices of the Project Management and Requirement Management processes considered more difficult to maintain the execution on new projects after the assessment.

This paper is organized in four sections. Section 1 is the Introduction. Section 2 presents the literature review. Section 3 presents all about the survey applied on assessed organizations. The section 4 presents the quality research using Grounded Theory. Finally, section 5 concludes the paper and presents further works.

II. LITERATURE REVIEW

As a starting point, the literature review was performed using some steps of systematic review, a kind of secondary study, widespread by [3], in order to identify factors that may influence the continuity of the software processes execution. We set a protocol for implementation of a systematic review based on the protocol developed by [13]. We used tools such as spreadsheets to supporting the registration and analysis of the data collected.

Initially, the literature review was conducted in the main papers published in relevant symposium and workshops to the software quality area in Brazil.

According to [11], [2], [6] and [12], the level of formalization of organizational structure is one factor that can facilitate the implementation of a process improvement program.

For example, [11] observed that the formalization of organizational structures in areas, functions, skills and responsibilities demonstrates a level of organization that allows a simple way to visualize how the company is organized to meet their goals. The formalization should not be understood, in this case, as bureaucracy, but as a mean to facilitate the communication into the organization. The higher the level of organization and formalization of organizational structure, the lower is the risk of short-term view on the implementation of process improvement.

This factor can also affect positively the maintenance of quality, because a company with a more formal organizational structure has the ability to get sucess and maintain awareness of their roles, responsibilities and areas.

Another factor was the level of tools and techniques for reuse of knowledge utilization. [6], [2], [4] and [12] said that a company that wants to improve the maturity of their processes needs to take advantage of existing knowledge in the organization. It might be supported by a knowledge management environment. From the reuse of knowledge the organization can implement improvements to the software processes and accelerates the implementation and execution of them.

This factor is also relevant to the continuity context, because the company that has tools and techniques to support the reuse of knowledge can more easily keep alive the knowledge of the implemented processes, encouraging

their use and keeping clear the need of their activities and tasks. [2] [6] and [12] described the use of return of investment (ROI) indicators as having a strong influence in the maintaining of a Improvement Program. [12] said that during the implementation of the processes, the team clearly felt the need for a mechanism to recognize the benefits associated with improvement efforts. It serves to motivate the members to continue the improvement of the processes aiming to achieve higher levels of maturity. [6] also advise organizations to analyze, often, the ROI achieved with the improvement of software processes, making them visible, because it decreases the risk of stopping the process of continuous improvement.

Others factor were: the level of available resources for the maintenance and improvement of the processes and the commitment of the high managers with the Improvement Program.

According to [6], a company that wants to continually improve their processes needs to provide the necessary resources, because it depends largely on investment in personnel, support tools, as well as in specialist consultants. Thus, the existence of committed high managers, providing the appropriate support, aids the company to achieve continuous improvement of their processes. High managers that keep the team motivated and the processes properly supported will likely remain adherent to the model, obtaining a continuous improvement of the processes' quality [8] [4].

One of the factors cited was the level of commitment of the Improvement Program coordinator to the implementation and improvement of the processes. [11] said that the ability of the company's SEPG (Software Engineering Process Group) to conduct the Improvement Program affects directly the performance of the project.

In some companies, it was found that the attitudes of the SEPG may difficult the monitoring of the activities related to the implementation and maintenance of the processes execution. [11], [6] and [5].

The level of the formalization of communication is other factor described by [6] and [8]. [11] said that the higher the level of organization of communication in the organization, the lower the risk of failure in interpretation and understanding the requirements to implement the activities.

To maintain the continuous improvement of the processes and adherence to the maturity model, also depends of the formalization of communication in the level of projects and inside of the Improvement Program.

There are also two factors that have similarities and are cited by [6], [8] and [9], which are: the level of knowledge about software processes and software quality, and the level of use of software engineering methodologies and concepts.

To [11], a high level of knowledge and understanding of the collaborators, mainly opinion makers, on concepts related to software quality is very important to evaluate and to understand the impact that an Improvement Program can represent to the organization. The higher the level of knowledge about concepts related to software quality and to the maturity model, the lower the risk of underestimating the resources required to the Improvement Program. [5] said that the implementation of software processes involves knowledge-intensive activities. It means that those involved in software process improvement initiatives must have deep knowledge of software engineering and must be able to use this knowledge to guide the implementation of the processes.

The company that has knowledge about software quality and software engineering, distributed among its employees, has a better chance of maintaining their activities adherent to the maturity model.

Other factor which influences the maintenance of the Improvement Program is the level of resistance to cultural changes. The comfort zone that exists before the implementation of the processes and during the maintenance of the Improvement Program is a very important barrier.

The human being has a natural resistance to changes. Whenever the organization leaves the comfort zone, some employees opposed the execution of required tasks and the use of necessary methods and techniques [6]. [4] highlighted another important factor: the level of investment in training on software processes. [11], for example, said that difficulties related to cultural changes can be mitigated with training, lectures and meetings in complementary disciplines.

A continuous investment in training facilitates the continuity of the adherence to maturity models by improving the knowledge related to software quality and software engineering.

One factor that can make difficult the benefits of training initiatives is the high turnover of human resources, because it generates high cost and the knowledge cannot be used to support the Improvement Program [8].

According to [6] and [11] an important factor to the continuity of the Improvement Program is to deal with it as a project.

In fact , [11] said that as an Improvement Program has the philosophy of a project it should be planned and monitored as any software development project, including setting marks/checkpoints and constant monitoring of progress and costs involved in this initiative.

Another factor found in the literature is the importance of consultants to obtain and maintain the software processes adherence to the maturity model.

For [6], the importance of consultants becomes clear because many of them has the opportunity to participate in others organization assessments as a consultant or as an auditor and so they know exactly what is required and the reality of others companies (benchmarking). They observed that a well qualified consultancy facilitates the definition of better business goals to the company and a more adherence to the maturity model.

However, it should be highlighted that the continuity of the Improvement Program depends of how the consultants

are able to teach the collaborators to execute their activities and how the knowledge becomes a real asset of the organization, that is, they can leave and the software process culture remains on the company.

Finally, another factor was the level of utilization of support tools. The more support of tools, lower is the risk to neglect the practices required by the maturity model [11] [2].

III. SURVEY

A. Survey's Methodology

The next step in the research was the completion of a questionnaire to identify the level of influence of factors, identified in the literature review, to maintain adherence to the software process maturity model.

We defined a questionnaire divided into three parts: the first was a set of objective questions, which sought to characterize the professional respondents, the second part listed the factors that influence the maintenance of software processes, according to the literature review, and was asked to respondents how important were the factors for the maintenance and implementation of the Improvement Program. Respondents could add others factors of significance. Finally, the third part of the questionnaire reported the expected results (practices) of processes evaluated at the level G of MPS.BR, which are the Project Management and Requirements Management, seeking to ascertain the more difficult activities to be continued after the assessment has been performed. Participants were encouraged to justify their evaluation if they considered important for his response.

The questionnaire received 27 responses and the profile of respondents was: 44% of them had master degree and 22% had finished the university. 40% had 6-9 years of experience in software quality and 30% had 3 to 6 years. 48% had been a project manager for five or more projects and 28% had managed 3-5 projects. 37% worked as project management for 1-3 years and 25% from 3 to 6 years. 66% had already participated in a software engineering process group, 40% were certified implementers, 33% were consultants and 29% were certified auditors of MPS.BR. 50% of respondents participated in one or two assessments and 30% participated in five or more.

B. Analysis of Results

To help the analysis, a table was built to register the relevance's degree of each factor, the degree of difficult to continue the execution of each practice of the Project Management and Requirement Management processes, after an assessment.

The factors were analyzed using the following scale: 0 - No relevance; 1 - Low relevance, 2 - Reasonable relevance; 3 - High relevance, 4 – Indispensable.

TABLE I. THE LEVEL OF FACTORS' INFLUENCE

Factors	0	1	2	3	4
F01-Organizational structure known by the employees	3,7%	18,5%	33,3%	25,9%	18,5%
F02-Effective structure for communication	0%	0%	22,2%	29,6%	48,1%
F03-High level of knowledge about content related to improvement process and software quality	0%	0%	34,6%	38,4%	26,9%
F04-High level of maturity of the organizational culture to use software engineering methods and techniques	0%	0%	18,5%	44,4%	37,1%
F05-High level of automated tools utilization in supporting the software process	0%	11,1%	14,8%	37,1%	37,1%
F06- Low level of resistance to changes	0%	0%	18,5%	44,4%	37,1%
F07-Commitment of project managers with the Improvement Program	0%	0%	7,4%	11,1%	81,4%
F08-Utilization of indicators that demonstrate the return obtained with the processes' execution	3,7%	0%	14,8%	48,1%	33,3%
F09-Provision of the necessary resources by the High Managers	0%	0%	0%	40,7%	59,2%
F10-Utilization of knowledge management approaches and tools	0%	18,5%	59,2%	22,2%	0%
F11-Guidance of an external consultancy	0%	7,6%	26,9%	50%	15,3%
F12- Effectiveness of the consultancy, that help the company to implement the software process, in transform the SEPG in an autonomous group and with a high level of knowledge	0%	0%	7,4%	48,1%	44,4%
F13-Low level of staff turnover	0%	0%	44,4%	44,4%	11,1%
F14-Maintanance of a strong and effective policy of trainings	0%	3,7%	29,6%	37,4%	29,6%

According to Tab. 1, the factors "F07" and "F09", "F12" and "F02" were considered the most relevant for the continuous execution of software processes, after an assessment. This result corroborated with the literature review findings.

However, the factors "F01" and "F10" were considered as the less important ones for the maintenance needs.

The third part of the questionnaire aimed to evaluate the practices (expected results) of the level G in MPS.BR, corresponding to the processes: Project Management and Requirements Management.

As we said above, each practice (expected result) was evaluated in relation to the level of difficulty to continue to perform it, after a MPS.BR assessment. For this, the following scale was used: 1 - Difficult to continue; 2 - Reasonable difficult to continue; 3 – Easy to continue. In cases when a respondent selects "1 - Difficult to continue" for the practice the questionnaire suggested the participant to justify it.

The Tab. 2 presents the obtained results for the practices (expected results) of the Project Management (PRM) process.

TABLE II. RESULTS OF PRM PRACTICES

Practices (Expected results)	1	2	3
PRM1 – The work scope for the project is defined	0%	6%	94%
PRM2 - The project tasks and work products are dimensioned using appropriate methods	6%	50%	44%
PRM3 – The model and the project's life cycle phases are defined	0%	13%	88%
PRM4 – The effort and cost to perform the tasks and work products are estimated based on historical data or technical references	50%	19%	31%
PRM5 – The budget and project schedule, including the definition of milestones and checkpoints are established and maintained	25%	44%	31%
PRM6 - Project's risks are identified and their impact, likelihood and priority to treatment are determined and registered	25%	38%	38%
PRM7 - The human resources for the project are planned considering the profile and knowledge needed to perform it	19%	56%	25%
PRM8 – The resources and work environment needed to run the project are planned	6%	44%	50%
PRM9 - The relevant data of the project are identified and planned considering collection, storage and distribution aspects. A mechanism is established to access them, including, if pertinent, issues of privacy and security	38%	31%	31%
PRM10 – A general plan to run the project is established with the integration of specific plans	6%	19%	75%
PRM11 - The feasibility of achieving the goals of the project , considering the constraints and resources available, is evaluated. If necessary, adjustments are made	25%	38%	38%
PRM12 - The project plan is reviewed with all stakeholders and the commitment to it is obtained	25%	31%	44%
PRM13 - The project is managed using the project plan and other plans that affect the project and the results are registered	13%	44%	44%
PRM14 - The involvement of stakeholders in the project is managed	19%	44%	38%
PRM15 - Reviews are carried out in project's milestones according to the planning	6%	69%	25%
PRM16 - Record of identified problems and the results of the analysis of relevant issues, including critical dependencies are established and treated with stakeholders	25%	38%	38%
PRM17 - Actions to correct deviations from the planning and to prevent the recurrence of identified problems are established, implemented and followed until its conclusion	25%	38%	38%

The practices (expected results) "PRM4" and "PRM9" were considered as the most difficult to be performed after the assessment.

According to one of the participants the PRM4 is hard to be continued because the maintenance of a database with reliable historical data is a great challenge and is fundamental to this practice (expected result). The difficulty to maintain such a database can make the involved collaborators use only subjective opinions to estimate the project's effort and cost.

The results obtained for the practices (expected results) of the Requirements Management (REM) process is presented in Tab.3.

TABLE III. RESULTS OF REM PRACTICES

Practices (Expected results)	1	2	3
REM1 - The requirements are understood, evaluated and accepted by the requirement suppliers, using objective criteria	0%	38%	63%
REM2 - The commitment of technical staff with the approved requirements is obtained	19%	25%	56%
REM3 - The bidirectional traceability between requirements and work products are established and maintained	56%	38%	6%
REM4 - Reviews in plans and work products of the project are performed to identify and correct inconsistencies in relation to requirements	25%	38%	38%
REM5 – Changes to requirements are managed throughout the project	31%	38%	31%

The practice (expected result) for "REM3" was considered one of the three expected result most difficult to be continued by the respondents. For one of them, a consultant, this result is due to the lack of automated tools that can ensure the bidirectional traceability. The tools may facilitate a little more the continuity of this practice.

IV. QUALITATIVE RESEARCH USING GROUNDED THEORY

A. Grounded Theory Research Methodology

Grounded Theory is a qualitative research method that uses a set of systematic procedures for collecting and analyzing data to generate, develop and validate substantive theories about phenomena, essentially, social, or broad social processes [15]. Application of Grounded Theory in the areas like software engineering and process improvement is even more sparse, as said [16]. But some studies have been highlighted.

Their authors, Glauser and Strauss, argue that there are two basic types of theories: the formal and substantive, as they say [17]. The first type consists of conceptual and comprehensive theories, while the second type is specific for a particular group or situation and is not intended to generalize beyond their substantive area.

According to the line proposed by Strauss, GT (Grounded Theory) is based on the idea of encoding (coding), which is the process of analyzing the data. When coding, concepts are identified (or codes) and categories. A concept (or code) names a phenomenon of interest to the researcher, abstract an event, object, action, or interaction that has a meaning for the researcher [18]. Categories are

groupings of concepts together in a higher degree of abstraction.

The encoding process can be divided into three phases: open coding, axial and selective. The open coding involves breaking, analysis, comparison, conceptualization and categorization of data. According to [15], in the early stages of open coding, the researcher explores the data scrutinizing what you believe relevant due to the intensive reading of texts. In the phase of open coding, incidents or events are grouped into codes by comparing incident-incident.

Also the open coding is performed to create categories that add the codes to reduce the number of units that the researcher will work [19].

After the identification of conceptual categories for open coding, axial coding examines the relationships between categories that make up the propositions of substantive theory [15]. The relationships between the codes can be defined by the researcher. As proposed by [18], these relationships form what the authors call as paradigm: causal conditions, players, strategies and consequences of actions / interactions.

Finally, selective coding refines the process, identifying the core category theory, with which all others are related. The core category should be able to integrate all the others and express the essence of the social process that occurs between those involved. This core category can be an existing or a new one [19].

B. Results Analysis of Grounded Theory

In this qualitative study, were performed three stages of coding proposed by grounded theory, it was possible to find the set of theories to answer the question under study (what influences the continuation or the abandonment of processes compliant with the model after an evaluation?).

From interviews with members of the SEPG of assessed companies, responses were transcribed into documents which collected their quotes. We did not use seed categories - an initial set of codes to begin coding has been created in vivo codes from the text of the questionnaires. In some codes, the quotes are changed to facilitate reading and to collect more citations. These changes have led to codes found that resembled with those components of critical success factors of [16].

The step of open coding and axial is overlap and come together because of the interactivity of the process. The codes and categories identified gone through successive revisions, of which 28 were produced codes associated with three categories and two subcategories, and all under a central category.

The core category called "Factors that influence the maintenance", and this has (using the notation "is a part of " as suggested by [20]), the following categories "Social and Cultural", "Technical", "Resources and Commitment". The Figure 1 presents, graphically, this organization.

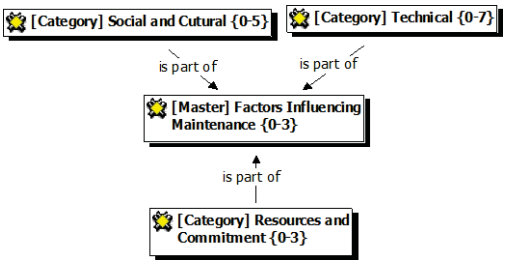


Figure 1. Axial Coding Categories.

As a memo, during the search were found many citations that reported results of a company with a presence and the with a absence of the same factor. Thus, to avoid creating two categories show that the presence and absence of them in companies realities, it was decided to add these quotes in just one factor, which would be named on the presence of the same, but would add that the quote also reported their absence. And you are using the notation "Evidence of difficulty" for when they were found only quotations that show the influence of a negative factor related to maintenance.

In several, the category with more quotes aggregated is one that combines technical factors, in other words, is the one that has a direct connection with the model MPS.BR. Two subcategories were used in those categories, to agregate the coding process areas, given as the more difficult to keep the outcomes of the G level. Figure 2 shows the connection of these codes with the category "Technical" through the use of the notation "is a" according to [20].

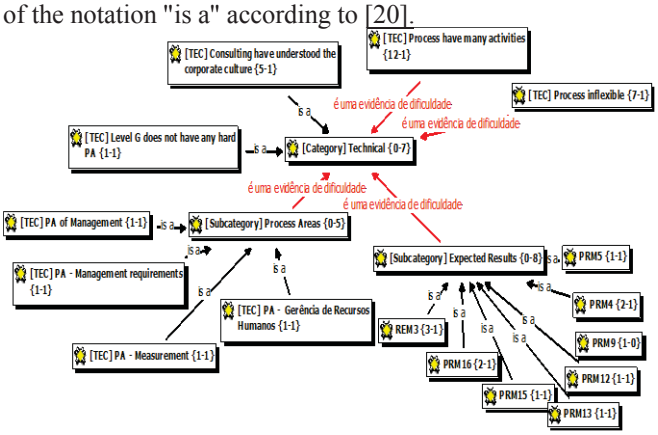


Figure 2. Category Factors "Technical".

In this category codes that stand out are "Process has many activities", “Process inflexible", that complain about the bureaucracy and about the inflexibility of the model. One of the respondents cited "If it were a process more flexible and it had not so many activities, it would be an easier process for the company to keep". Another that stands out is the code that denotes a bidirectional traceability requirement, by the model in GRE3 expected outcome, thus confirming what was said in our quantitative research.

Another category coding was found in the "Social and cultural", which collects the social and cultural factors that may influence the maintenance process of acceding to the model as follows in Figure 3.

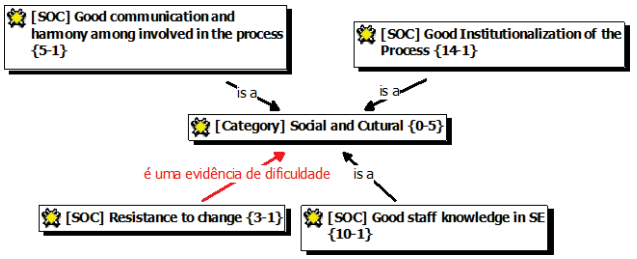


Figure 3. Category Factors "Social and Cultural".

The code that stands out in this category is the "Institutionalization of Good Process", one respondent commented that to avoid abandonment of the processes is necessary to "Add the process and project management to the company's culture."

The last category involves the factors that mention the need for resources, be they personal, cost or time, or the need for commitment of members involved in the process, managers or senior management. Figure 4 represents the connection of these codes.

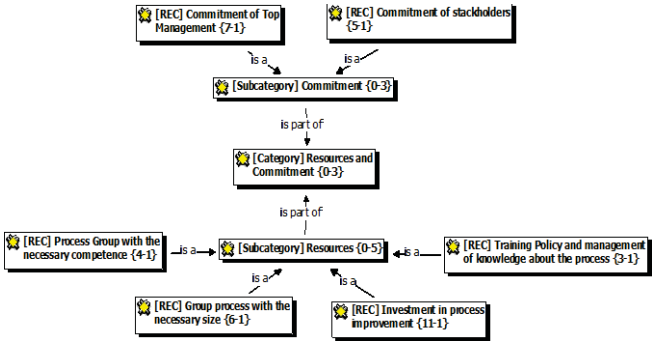


Figure 4. Category Factors "Resources and Commitment"

"Commitment of top management" was the code of this category that received more citations, confirming its importance as an influencing factor. Even the responses relating to this factor was mentioned that "top management commitment is one of the main factors that help maintain. The more committed to senior management, easier to maintain the process ".

V. Conclusion and Further Works

This work presented the result of a literature review, a survey and a qualitative research, using Grounded Theory, related to the difficulties that the software organization faces to maintain the adherence of their software processes to the maturity models, after the assessments.

The results corroborated with the literature findings, that is, the software organizations have a lot of difficulties to continue to perform some practices, considering the competitive scenario of the industry of software.

This work can be used by the software organizations to guide their actions to do not permit the abandon of some practices necessary to the adherence to the models.

As further works, we intend to increase the size of the sample and define a set of actions that may facilitate the implementation of the most difficult practices (expected results).

References

[1] CMU/SEI (2006), "Cmmi for development (CMMI-DEV), version 1.2", technical report CMU/SEI-2006-TR-008. Software engineering institute, Carnegie Mellon University, 2006.(1)

[2] R. C. Silva Filho and A. E. Katsurayama, Experience in the implementation of the reuse management process in software engineering laboratory of COPPE / UFRJ. Rio de Janeiro, RJ: COPPE/UFRJ, 2008(2)

[3] B. Kitchenham, "Procedures for performing systematic reviews", Keele, Staffs: Keele University, 2004.(3)

[4] R. W. Monteiro, R. Cabral, F. Alho , C. Martins and A. R. Rocha, Required resource for software process institutionalization in Prodepa. Computer engineering system program. Rio de Janeiro, RJ: COPPE/UFRJ, 2008

[5] M. Montoni, C. Cerdeiral, D. Zanetti, A. R. Rocha, One approach to driving improvement initiatives for software process. Rio de Janeiro, RJ: COPPE/UFRJ, 2008

[6] T. M. G. Parente, A. B. Albuquerque, Domínio informática: the quality and focus of its strategic plan. IV MPS.BR workshop of consultants, SOFTEX, 2008.

[7] Pressman, R. S. (2006): "Software Engineer" 6ed. São Paulo: McGrow Hill.

[8] J. B. Porto, , A. C. Pereira, J. Pohren, Proposal Process Improvement of WBS Integrated Solutions using MR.MPS and PRO2PI approach. IV MPS.BR workshop of consultants, SOFTEX, 2008

[9] A. R. Rocha, "IA COPPE/UFRJ: Learned lesson on 2008", III MPS.BR Workshop of appraisers, SOFTEX.

[10] SOFTEX, MPS.BR – General Guide, version 1.2. Avaliable on: www.softex.br

[11] D. Yoshida and M. Tavares, Learned lessons by IIITS on leve G MPS.BR implementarion project to Salvador's companies group. IV MPS.BR workshop of consultants, SOFTEX, 2008.

[12] D. Zanetti, et al, Learned lessons on implementation of MPS.BR in COPPE/UFRJ Software Engineering Laboratory. IV MPS.BR workshop of consultants, SOFTEX 2008.

[13] R. C. SILVA FILHO, An approach for evaluating proposals for improvement in software processes. Rio de Janeiro, RJ: COPPE/UFRJ, 2006

[14] International Organization for Standardization and International Electrotechnical Commission. ISO/IEC 12207 Amendment: Information Technology - Amendment 2 to ISO/IEC 12207. Genebra: ISO,2004

[15] BANDEIRA-DE-MELLO, R., CUNHA, C., 2006. "Grounded Theory".In: Godoi, C. K., Bandeira-de-Mello, R., Silva, A. B. d. (eds), Pesquisa Qualitativa em Estudos Organizacionais: Paradigmas, Estratégias e Métodos, Chapter 8, São Paulo, Saraiva

[16] Montoni, M, Rocha, A.R. (2010), Aplicação de *Grounded Theory* para Investigar Iniciativas de Implementação de Melhorias em Processos de Software. IX Simpósio Brasileiro de Qualidade de Software

[17] BIANCHI, E. M. P. G., IKEDA, A. A. (2006). Analisando a Grounded Theory em Administração. IX SEMEAD - Seminários em Administração. São Paulo, Brazil

[18] Strauss, A., Corbin, J., 1998. Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory. 2 ed. London, SAGE Publications

[19] Conte, T. U. 2009: Técnica de Inspeção de Usabilidade baseada em Perspectivas de Projeto Web – Rio de Janeiro: UFRJ/COPPE, 2009

[20] GLASER, B., 1992. Basics of grounded theory analysis. Mill Valley CA, The Sociology Press.

BDI agents to bridge cloud computing and end-users

Case study: An agent-based personal trainer to COPD patients

Kasper Hallenborg
Maersk Mc-Kinney Moller Institute
University of Southern Denmark
hallenborg@mmmi.sdu.dk

***Abstract*—Cloud computing is envisioned a dominant role in the future. Extensive amount of data are stored, applications running in the cloud, and globally accessible. However, users are not interested in observing and processing that amount of information. Thus, a mentalistic model that represents the user's goals could be integrated with the cloud to present processed extracts in a cognitively accessible way. Such an approach is presented with deliberative BDI agents both in the general and in a case study for an agent-based personal trainer for COPD patients.**

Keywords: BDI agents, cloud computing, integration, PHRs

I. INTRODUCTION

Personal/Electronic Health Records (PHR/EHR) is a perfect example for the use of cloud computing, where the multi-user / multi-role perspective, privacy, security, and accessibility issues are addressed, as both health-care professionals and you personally are involved. Clouds spawned from service-oriented architectures and semantic web services, which has approaches trying to abstract and increase flexibility of services, such as WSDL and OWL ontology. However, the composition of services must still rise from a set of basic action by the user [1]. For PHRs the heterogeneous set of users and the complexity requires medical information to be presented in a cognitively accessible way [2]. We propose a deliberative agent model based on the BDI architecture to bridge cloud-based services (SaaS) and pervasive services to the user. The model is applied in a case study for a home training of COPD patients.

II. AGENTS TO THE CLOUD

A. Limitations of a service based model

Traditionally, web services were defined as software systems designed to support interoperable machine-to-machine interaction over a network [3]. Even with semantic descriptions of the services (OWL-S), a service based model does not overcome the limitations in how users naturally will express the goals they want to achieve rather than the actions they wish to be performed [1]. Thus, several key aspects are important if user-centered cloud apps should provide intelligent services.

- *Locus of control* – design of applications must be user-centered and reflect the user's assessment of goals to be achieved.
- *Cognition* – pervasive applications should not rely on the user interaction in respect to current goals.
- *Adaptation* – information resources often reside in many systems (clouds) and needs to be combined either to be relevant or make sense for users.
- *Context* – environments are highly dynamic due to users' mobility and global accessibility.

B. Capabilities of deliberative agents

Semantic web services is built on the basis of synchronous remote procedure calls and has been preferred by industry over more abstract definitions and less strict approaches for agent technologies, which comes from asynchronous message passing architectures [1]. However, the goal-oriented agents have a lot more to offer than services founded in basic actions. From a modeling perspective the autonomous characteristic of agents and the architecture for deliberative agents, e.g. the BDI model, has a more natural mapping to a user-centered design. The BDI model has a very "mentalistic' notion for capturing user's preferences and goals [4], whereas services mostly are designed as information providers that needs further (mental) processing.

Goal-directed behaviors are the nature of deliberative agents, and goals can be abstractly described. Reasoning on percepts and other inputs is required, if user acceptance should be improved. Interoperability is central to web services, but despite profile specifications and dynamic service discoveries, web services are still relying on common standards. The agent community builds on knowledge representation and ontologies to abstractly describe content of messages, which improves the possibilities of adaptation across different systems.

III. THE AGENT MODEL

Deliberative agents extend the basic agents model given by Jennings and Woolridge [5] with knowledge representation and symbolic reasoning capabilities [6]. The locus of control is captured in terms of the autonomous characteristic of agents. A reasoning engine covers the need for cognition. Adaptation and context-awareness is well aligned with reactive and proactive behaviors of agents and their responsiveness in general. The BDI model by Rao and Georgeff [7] is the most applied architecture for deliberative agents. The scope of the demo and the presented model is to cover use of cloud computing as an information source. The operational logic of BDI agents was formalized by Rao with the AgentSpeak(L) language [8]. The BDI agent is given the tuple $\langle \mathcal{B}, \mathcal{D}, \mathcal{I}, \Pi \rangle$. Where $\mathcal{B}$ is the belief base, $\mathcal{D}$ is the desire base, $\mathcal{I}$ the intention base, and Π is the set of selection functions and plans, which are defined as practical rules in the form $\varphi \leftarrow \epsilon | \pi$. φ is the goal, π the concrete plan of action that should lead to the goal, and ϵ the event that triggers the selection function. The agent-cloud communication is covered by the social capabilities. Agents would either reactively observe changes in the cloud or proactively request or query the cloud. It corresponds to `subscribe`, `request`, and `query` FIPA protocols. The triggering events for a state change will

come from `inform` messages, that will trigger the selection function on the belief base, which formally is given by

$$\frac{S_{\mathcal{B}}(\langle \mathcal{B},\mathcal{D},\mathcal{I},\Pi\rangle)=\langle \texttt{inform},id,at\rangle}{\langle \mathcal{B},\mathcal{D},\mathcal{I}\rangle \rightarrow \langle \mathcal{B}\cup\{at\},\mathcal{D},\mathcal{I}\rangle} \quad (1)$$

Where *id* is the identifier of the agent and *at* is the predicates expressed in first order logic, as explained in [8]. In the deliberation engine of the agent it will consider the desire base if there are relevant plans that should be added to the intention base, and such a plan would be considered applicable if a unifier θ exists for the plan, so the plan is a logical consequence of the current belief base with respect to the event. Formally, we can describe it as an update to the intention base of the agent.

$$\frac{\langle \varphi,C\rangle\in\mathcal{D},\text{impact}(\beta,\varphi,start),\mathcal{B}\vDash C\theta,\mathcal{I}\nvDash\varphi\theta}{\langle \mathcal{B},\mathcal{D},\mathcal{I}\rangle \rightarrow_{\theta} \langle \mathcal{B},\mathcal{D},\mathcal{I}\cup\{\langle \varphi\theta,C,\pi\rangle\}\rangle} \quad (2)$$

where C is the commitment of the belief base, and $\text{impact}(\beta,\varphi,start)$ is evaluation of a change in belief base, β, that will lead to the desire of the plan φ being followed.

IV. A CASE STUDY

Healthcare systems are quite different around the world, especially between Europe and the US, but the need to store health related information and PHRs are universal. An overview of the system for the case study is presented in Fig. 1.

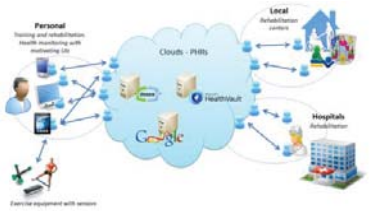


Figure 1: Overview of healthcare system with PHRs in the clouds

It is commonly known that the share of elders will double due to demographics changes and costs of healthcare services will explode. In particular expenses to chronic diseases, such as chronic obstructive pulmonary disease (COPD/COLD), diabetes, cardiovascular disorder and Alzheimer's. There were 726,000 hospitalizations for COPD in the US in 2000, and the cost of COPD is $32.1 billion [9]. Especially, for COPD patients can benefit from training. Due to breathing trouble they do not only get anxious while exercising, but also fear walking the neighbor (even if capable), which lead to social exclusion. Ideally, exercise training would be individual, customized, assisted, and supervised by physiotherapists. Unfortunately, such training sessions are only provided to a very few patient. Recent year of advancement in sensor technologies enables opportunities for monitoring training at home.

A. Our approach

COPD patients are no different than the rest of us – it is very hard to keep up the motivation for physical exercising, unless someone constantly motives you or the results/effects are very visual. Thus, we need a responsive and proactive feedback that can motivate to keep up the training. Not all COPD patients are capable of walking outside, so exercises at home are required for many patients. Exercise programs are provided by physiotherapists, and we are developing a prototype based on the Kinect camera that can verify exercises and stream data to a PHR Google Health account. Integration with Google Health is rather simple, as it is part of the Google Data Protocol. The API for the Kinect camera allows you to analyze the joint angles of the body so it can be compared to an exercise program. Fig. 4 shows the Kinect's view of joints in the body.

Figure 2: Kinect view of a body

Figure 3: Avatar

From the professional healthcare system the medical records and diagnose data of COPD is registered in the personal health records. However, as training programs are not directly supported in the Google Health the agent must reason on the extracted information and compare it to exercise plans in the desire base of the agent. The agent will update the belief base based on inputs that are received from the cloud according to (1). If the evaluation of the change in the belief base, enables a new plan to be followed it will be added to the intention base of the agent according to (2). The applicable plan will contain tasks to update the UI and feedback to the user. The prototype uses text-to-speech for an audio feedback, and we experiment with mimic software to create an avatar video. A snapshot of the avatar is shown in Fig. 5.

V. CONCLUSION

We have presented a BDI-based agent model to close the bridge between cloud computing and user-centered applications that aim to present rich and complex information from clouds in a cognitively accessibly visualization to the end-users. A prototype system is built for home training of COPD patients using extracts of information from the cloud to give motivational user feedback, so they continue training.

REFERENCES

[1] Ian Dickinson and Michael Wooldridge, "Agents are not (just) web services: considering BDI agents and web services," in *Proc. of the 2005 Workshop on Service-Oriented Computing and Agent-Based Engineering (SOCABE'05)*, Utrecht, The Netherlands, 2005.

[2] Ali Sunyaev, Dmitry Chornyi, Christian Mauro, and Helmut Krcmar, "Evaluation Framework for Personal Health Records: Microsoft HealthVault vs. Google Health," in *Proc. of 43rd Hawaii International Conference on System Sciences*, Koloa, Kauai, Hawaii, 2010.

[3] David Booth et al. (2004, February) Web Services Architecture. [Online]. http://www.w3.org/TR/ws-arch/

[4] Stefania Costantini, "Agents and Web Services," *The ALP Newsletter*, vol. 21, no. 2-3, August 2008.

[5] Michael Wooldridge and Nick Jennings, "Intelligent Agents: Theory and Practice," *Knowledge Engineering Review*, vol. 10, no. 2, 1995.

[6] Michael Wooldridge, *Reasoning about Rational Agents*.: MIT Press, 2000.

[7] Anand S. Rao and Michael P. Georgeff, "BDI Agents: From Theory to Practice," in *In Proceedings of the first international conference on Multi-agent systems (ICMAS-95)*, 1995, pp. 312-319.

[8] Anand S. Rao, "BDI Agents speak out in a logical computable language," *Lecture Notes in Computer Science* , vol. 1038, no. 1038, pp. 42-55, 1996.

[9] (2004) COPD International. [Online]. http://www.copd-international.com

Cloud Engineering approach in business innovation

Giorgio Valle
Dipartimento Scienze Informazione
Università degli Studi di Milano
Milan, Italy
giorgio.valle@unimi.it

Bruno Apolloni
Dipartimento Scienze Informazione
Università degli Studi di Milano
Milan, Italy
bruno.apolloni@unimi.it

I. Poster Introduction: Cloud Engineering

The research in the area of Customer Relation Management (CRM) applied to two prototype studies on *learning* and *mobility* offers an opportunity to experiment new multimedia data management, data analysis, and distributed collaboration. Software engineering practices based on cloud technology techniques acquire from the Web what is needed for rapid prototyping and to satisfy the needs of customers for market validation of pricing policies and marketing strategies.

This approach allows to specify the projects' concepts by reaching stakeholders of needed Information & Communication Technologies (ICT) to implement a prototype relying on software components available elsewhere. Two sample cases are presented to clarify this cloud-engineering approach:

1) Program for Recovery Insufficient Grades in High-School (PRISC);
2) Sustainable Mobility through Social Mobility (SM^2) to reduce dependence on one's owned cars to satisfy mobility needs.

The PRISC strategy focuses on the learner and on how proficiency is assessed through grades. The action steps are:

- understanding is focused on the students' study on schoolbooks;
- assessment is delegated to exercises and tests managed on screens of mobile telephones or PCs in an interactive Web environment;
- students with excellent grades act as tutors by exploiting social networks.

The SM^2 strategy focuses on exploiting the empty seats available in most cars running in metropolitan cities with air pollution problems. Key features:

- direct negotiation via mobile devices between car drivers and ride seekers;
- short range contacts insuring prompt satisfaction;
- GPS ride surveillance.

From the standpoint of a University environment, the attention that Cloud computing receives on scientific publications [1] and on the daily press as well, indicates a major potential for business innovation besides job opportunities for young graduates once a win-win situation is established which could leverage on cloud technology techniques to acquire on the Web what is needed for rapid prototyping of business models concepts, while reducing time-to-market to satisfy the needs of customers.

Many companies provide Cloud technology: CA Technologies, IBM, Microsoft, Google, Seeweb are those we could approach. The specific focus on CRM brought to select the academic license granted by SFDC Inc. (www.Salesforce.com) for this prototype implementations because of its 11 years experience gathered in offering the "Sales Force Automation" solution in web-based modality and on-demand. The functionality covering "Service and Support Management" is of specific value for our studies allowing us to practice SaaS (Software as a Service). While the access to the programming infrastructure and the services of hosting and administration allows to develop PaaS (Platform as a Service).

The development programming environment refers to a Java language named APEX which allows to write code for any application beyond the initial CRM objective.

II. Program for Recovery Insufficient Grades in High-School (PRISC)

High School Education is a major concern for Governments since economy globalization and technological innovation in industrial processes require staff and managers able to anticipate unpredictable market changes. At the same time the economy recession and lower government budgets force to prototype new educational formats where ICT and social networking could make education more effective and profitable.

To deliver an overall better education in High-Schools requires a coordinated action on the stakeholders: teachers, students, textbook publishers, school administration board, parents, laboratories and their equipment [2]. Since PRISC strategy focuses on the learner and on how proficiency is assessed through grades, the learners involved in PRISC action are both those with bad grades and those with excellent grades who are meant to act as tutors.

At present bad grades recovery happens either with additional classroom-based teaching organized by the School and free of charge to students, or by private action where parents look for teachers (or university students) for private tutoring usually paid cash and with no receipts.
In its initial phase, PRISC focuses on a few subjects: Mathematics, Physics, Computer Science. The action succeeds when a student with bad grades in specific subjects succeeds in recovering them in order to proceed to next higher level class.

All CRM procedures are managed on the cloud. The resulting system is inherently distributed and the central management coordinates the outsourced components (users do not know where data and procedures are stored). The PRISC coordinator interacts with a set of web-services provided by the cloud interface. He doesn't know directly the actual students (both with insufficient or with excellent grades). He only interacts with standard tools reckoning the student activities and with a standard platform to access and store the teaching materials. Thus there is no need for any venture capital investment, since

the financial balance only comes from a pay-per-use fee that is tightly commeasured to the benefits of PRISC activity.

In particular:

a) the study on text-books of specific topics not well understood involves cooperation with specific text-book Publishers to edit relevant exercises related to their books:
b) modern ICT digital video-interactive communications over screens of mobile phones or PCs involves Mobile Telecommunications Carriers
c) the web-based support to a learning community (where students with good grades help other students to recover from bad-grades) involves Social Networks platforms
d) implementation involves partnership with Education stakeholders at national and local government both in public and in private sectors.

III. THE SM^2 (SUSTAINABLE MOBILITY THOUGH SOCIAL MOBILITY) PROJECT

In metropolitan areas with air pollution problems the SM^2 strategy focuses on exploiting the empty seats available in most cars. The distinguishing strategy through which SM^2 plans to hit its target is the raising up of a Social Mobility community capturing the consensus of the customers thanks to widespread ICT instruments such as Internet and GPS on the one hand and socio/economic motivations on the other [3].

The approach is distributed at a great extent, with cloud computing acting as an ideal middleware. Indeed, the community autonomously emerges from the interaction of commuters without any previous direct contact. They simply interact through an *id* that attests their mobility in the same geographic range and requires information tools in the cloud to match them on the empty seats on other commuters' cars.

The model is intrinsically distributed since the managers of the transactions are exactly the actors who access remote resources through a portal installed through APIs on their mobile terminals. This will allow to achieve the following basic goals:

a. Extemporaneity of the transaction. We assume that users wait for maximum 10 minutes between when they plan to move and the planning of an alternative ride . Otherwise they will decide to move with their own means.

b. Efficiency of the transaction. Operational and technological efficiency are needed to guarantee the above time limit.

c. Reliability of the transaction. Both the reliability and the security of the service need to be considered together with emergency procedures in case of anomalies.

d. Green mobility issues require the monitoring of energetic and ecological efficiency as keen quality parameters.

The GPS availability allows to subdivide the community in users groups by location and consequently by time-range so that we could combine *demand and offer* in an optimal way both in terms of quality of the services (QoS) and of energetic consumption. Key implementation aspects are:

1) a distributed information support for the formulation of the ride requests and offers. The peripheral hardware will be constituted by new generation mobile terminals endowed with the Android operative system. An important part of this support will be the message ciphering and anti-intrusion systems and the transaction sampling and log compacting systems.
2) a cloud database for the collection of the requests, their interface with the availabilities, and finally the dispatching of the service orders to the transport suppliers. The DB should also contain the personal data of the community members with their preferences profile and credibility, updatable on line. The DB will store the traces of the transactions, including GPS traces, for a time sufficient for any security control need, but not exceeding the limits of the privacy of the members.
3) the design and implementation of the security system guaranteeing both the passenger and the driver in a non invasive but sufficiently robust way. Main points are:

   3a) the transaction enrolment from the driver-passenger assignment to the end-of-transport message by both the actors. The transaction will be followed through the service messages (call – offer – assignment - starting point agreement - end-of-transport message) and the GPS traces of the driver and of the passenger compared with the optimal routes suggested by satellite navigators

   3b) a library of rules for the agent system constituted by passenger, driver, and central server. With this library we will face both unpredictable events caused by traffic and real events linked to any anomalous behaviour of either the passenger or the driver. We will use fuzzy rules to be calibrated both in batch mode and online with neuro-fuzzy learning mechanisms[4].

IV. CONCLUSIONS

The lesson learned is how to deliver leaner, faster and more agile services to users – students and citizens - alongside with business stakeholders – Schools and Municipal Transportation companies – which are under pressure to make their business processes cheaper and more responsive to the customer's environment.

At present these two case studies simply constitute candidate test sites for Social Clouds instances which must be engineered in order to check the effectiveness of the benefits promised by their ecological paradigm [5].

The main benefit lies in the possibility of enabling a fast prototype implementation, in time to report results of these studies to the visitors of this poster.

REFERENCES

[1] "IEEE International Conference on Cloud Computing (CLOUD)".Thecloudcomputing.org http://www.thecloudcomputing.org

[2] Valle, G. & Epifania, F. (2008). An Interactive VideoJournal to complement Learning & Training. In Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications 2008 (pp.1164-1169). Chesapeake, VA: AACE.

[3] G. Elofson, P. M. Beranek, and P. Thomas. An intelligent agent community approach to knowledge sharing. Decision Support Systems, 20(1):83–98, 1997.

[4] D. Applebaum. Levy processes: From probability to finance and quantum groups. Notices of AMS, 51(11), 2004.

[5] R. Pezzi. Information Technology tools for a transition economy. http://www.socialcloud.net/papers/ITtools.pdf

SOFTWARE QUALITY IN TERMS OF ACADEMIC PROGRESS OF DEVELOPERS

Miriam Vázquez-Escalante, Jose Antonio Flores-Saucedo,
Facultad de Ingeniería,
Universidad Autonoma de San Luis Potosi
San Luis Potosí S.L.P., Mexico
vaem16@gmail.com flores.s.antonio@gmail.com

Hector G. Perez-Gonzalez, Juan Carlos Cuevas-Tello
Facultad de Ingenieria,
Universidad Autonoma de San Luis Potosi
San Luis Potosi S.L.P., México
hectorgerardo@uaslp.mx cuevas@uaslp.mx

***Abstract*— Since its inception, the undergraduate program in Computer Engineering from the Universidad Autonoma de San Luis Potosi in Mexico has been characterized for their hard work throughout the student's education. This work has placed this academic program at the first position in the national ranking, according to the national evaluation center (**CENEVAL**) of Mexico. To achieve this, an important factor has been the pattern of development of reseedgeh and technological development in most of their courses. The software projects are prerequisites for entitlement to qualifying. This paper analyzes a sample of projects developed over fifteen years (1995-2009) to assess quality and identify their strengths and weaknesses. We analyze the evolution of software quality in terms of stage developer training throughout their academic progress and the changes in general characteristics of the software developed over the period analyzed. The first analysis allows us to draw conclusions about the development of student potential, and the second one to evaluate the development of academic programs and apprenticeship schemes. Metric proposals were considered by the Software Engineering Institute (SEI) to characterize the projects.**

Keywords- Software Metrics; Software Quality; Maintainabilty;

I. Introduction

Software (sw) products of quality are performed by sw engineers along with strict methodologies that ensure the maximum reduction in the probability of defects. These guidelines should be followed at all stages of development To be characterized as good quality, it is not enough that a sw program be functional in accordance with the purpose for which it was made, but involves a series of metrics to be taken into account. Maintainability of sw products is a quality that can be accomplished. This applies to all products generated during the process [2]. This reseedgeh analyzes the evolution of quality in sw programs in terms of academic progress of their developers. We analyzed a sample of over 450 sw projects developed by students majoring in Computer Engineering at the Universidad Autonoma de San Luis Potosí (UASLP) for fifteen years (1995 to 2009). The students after five years graduate with an academic level that puts them in first national place according with the national evaluation center of Mexico.

II. Metrics and Methodology

Currently, there are a number of metrics that allow some form of sw measurement, however, as in [3], the author says that clear attributes should be defined that establish a proper system of measures for a final sw product. Moreover, in [4], the author argues that metrics are a good way to understand, monitor, control, predict and test the sw development and maintenance projects. Pfleeger [5] says that practitioners and reseedgehers can use metrics to make better decisions. In general, little attention is given to software metrics in the undergraduate curricula [6]. In the UASLP, these issues are analyzed in a single course at the end of the program. There are some efforts to learn about metrics using systems developed in the professional field [7]. Specialized literature does not show works, which analyze the systems developed by the students to evaluate their quality based on the progress in the study of their academic program.

Initially a sample of 405 projects was selected from the complete set provided by the school. The selection was carried out as follows: The projects were classified according to their subjects. Subsequently these subjects were classified into three levels which divided the curricular map of the academic program. On one level (level I) we included the following subjects: Programming Languages, Data Structures and Algorithms and Object-Oriented Technology, areas in which they are taught the programming fundamentals and the different programming paradigms The second level (level II) was made up of the subjects: Information Management, Information Systems, Operating Systems I and II, being areas in which the student is given freedom to follow the paradigm that is most appropriate to consider. Finally, the third level (level III) was composed of following areas: Software Engineering II (where you are taught the principles of sw quality to develop functional and high quality code), Expert Systems, Teleprocessing and System Programming I and II. These courses are taken in the last year. The latter total was 315 selected projects. Developer information, courses and dates were stored with the corresponding source code.

The following metrics were selected because they demonstrated good accordance with quality factors in the

empirical analysis of the programs developed during the course of software engineering at UASLP: LOC (lines of code), COM-RAT (commentaries ratio); V (Halstead Program Volume); E (EFFORT) which can be interpreted as a measure of the work required to understand sw already developed; V (G) (cyclomatic complexity) and SEIMI (index of maintainability of a project)[7].

TABLE I. METRICS OBTAINED BY ACADEMIC LEVEL

LEVEL	LOC	COM_RAT	V(G)	V	E	SEIMI
1	1676.46	0.05	7.73	809.05	1038783.90	-64.52
2	1314.18	0.05	4.45	664.41	334869.69	-87.09
3	2954.03	0.12	4.11	450.91	703116.77	-46.77

Table 1 shows the average number of lines of code used in projects as the student progresses in their academic program. It starts with an average of 1676 LOCs in the first third and ends with an average of 2954 of LOCs in each of their projects at the end. The volume of a program is defined as V = Nlog2n, where N = N1 + N2, N1 being the number of occurrences of operators and N2 the number of occurrences of operands. n = n1 + n2, n1 being the number of different operators and n2 the number of different operands. Figure 1 shows that the program volume decreases from Level I to Level III.

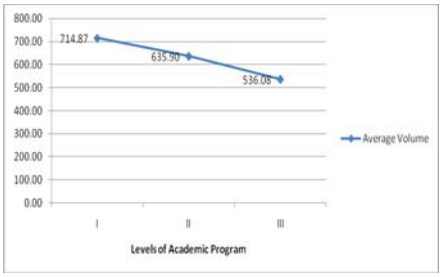
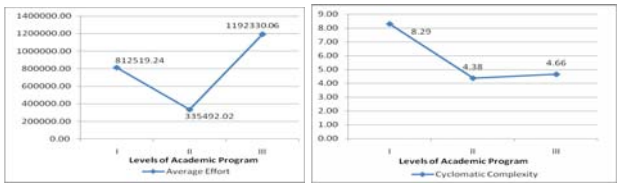


Figure 1. Program volume per level

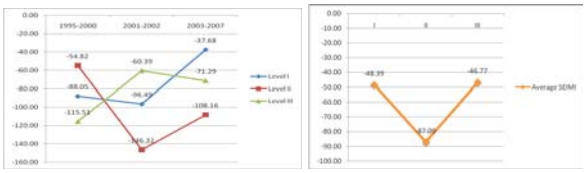
The effort to carry out a program is calculated by E = V * D. Figure 2(a) illustrates that the effort to develop programs at the end of the academic program nearly doubles the amount of the first third. The cyclomatic complexity by method or function represents the degree of ease or difficulty to understand. It is recommended [8] keeping this value below 6. Figure 2(b) shows that programs in the last two thirds of the major academic program present better maintainability.



a) b)

Figure 2. a) Effort to develop programs per level b) Cyclomatic Complexity of programs per level

We also analyzed changes in the quality of programs for students who have completed the academic program during the analyzed period of fifteen years. We used the software maintainability index (SEIMI) proposed by the SEI.[7] As shown in Figure 3(a) in recent years it is shown a very important development for level I in the last period in which there is no knowledge of concepts that contribute to the maintainability of software, this improvement is not present in levels II and III. Using the same index, we analyzed the entire project set. Figure 3(b) shows that the maintainability of the products at the beginning of the academic program decreases in the middle third and increases significantly at the end.



a) b)

Figure 3. a) Average SEIMI values per level and years b) Average SEIMI values per levels

CONCLUSIONS

Based on the information we can conclude that the programs developed by students of UASLP have the following strengths: It is noted that from the second third of the academic program, the average complexity of the program methods is reduced from 8.29 (at baseline) to 4.35 and 4.56 (in the last two levels). This improves code readability. We Identify the following weaknesses: A recommended SEIMI for professional software products exceed the value of 65 [7]. We propose a value of -65 as the minimum maintainability index of software products built by students during their academic formation. Based on this, we can conclude that the best programming practices are not applied in the courses of the middle third of the program and maintainability is reduced by increasing their complexity. It is emphasized that although the complexity of the programs developed at the end of the program is even greater, the maintainability index and hence the product quality level recovers.

REFERENCES

[1] Fairley, Richard E., *Software Engineering Concepts*, McGraw-Hill, 1990.
[2] Braude, Eric J., *SW Engineering: An O-O Perspective*, Alfaomega, 2003.
[3] Juan Fuente, Aquilino A., *Necesidad de Sistemas Formales de Métricas para Proyectos de Software OO,* Univ. de Oviedo. 1999
[4] Briand L.C., Morasca, S. y Basili, V. Property-based software engineering measurement. *IEEE Transactions on Software Engineering*. 1996.
[5] Pfleeger, S. L., Assessing *Software Mesasurement,* IEEE Software.3/ 1997.
[6] C. Munson, John, *SW Engineering Measurement*, Auerbach Publications, 2003.
[7] http://www.codeswat.com/cswat/index.php?option=id=67&Itemid=94, site accessed July 2009.
[8] Sommerville, Ian, *Software Engineering 8*, Pearson Education.

Towards a Novel Statistical Method for Generating Test Sets with a Given Coverage Probability

Cristiane Selem Ferreira Neves
Federal University of Rio de Janeiro
Postgraduate Program in Informatics
Fundão Island, Rio de Janeiro - Brasil
cristianeselem@gmail.com

Fábio Protti
Fluminense Federal University
Computation Institute
Niterói, Rio de Janeiro - Brasil
fabio@ic.uff.br

Eber Assiz Schmitz
Federal University of Rio de Janeiro
Postgraduate Program in Informatics
Fundão Island, Rio de Janeiro - Brasil
eber@nce.ufrj.br

Antônio Juarez Alencar
Federal University of Rio de Janeiro
Postgraduate Program in Informatics
Fundão Island, Rio de Janeiro - Brasil
juarezalencar@nce.ufrj.br

Abstract

The basis path testing is one of the most used techniques for generating white box test cases. This approach depends on the unknown state of the program variables at the time of execution, i.e., each path can be seen as the result of a random experiment, associated with an execution probability. We can define the coverage probability of a test set as the sum of the execution probability of its members. Although running the program a large number of times provide us an approximation for this set, its computational cost would be equal to that of testing the program in itself. This paper presents a method that uses a small set of execution samples to select a minimal set of execution paths, which has the property of its coverage probability being above a required confidence level, and then generate a natural language specification of the test case set. Experimental results show that it is not only simple to be applied but also generates a reliable test case set.

1. Introduction

White-box testing of software modules is the test activity where the test cases are derived from the analysis of the module source code [2, 4, 5]. The basis path testing is one of the most used techniques for generating white box test cases, using a set of execution paths. However it can produce a set with elements that have a small chance to be executed in practice. This poster presents a method that uses a small set of execution samples to select a minimal set of execution paths, which has the property of its coverage probability being above a required confidence level, and then generate a natural language specification of the test case set. The structure of this paper is as follows: Section 2 presents the most important concepts and Section 3 describe in details the method. Section 4 shows the computational experiments and Section 5 presents the conclusions.

2. Conceptual framework

In "basis path testing", the program is associated with a control flow graph, that has unique entry and exit nodes. A path is a sequence of connected nodes that traverse the control flow graph from the start node to the end node. Each path corresponds to a test case element.

Since the number of paths is potentially infinite, leading to an infinite number of test cases, McCabe [3] introduced the concept of independence in a strong connected graph. An independent path is any path in the program that introduces at least a new set of commands or a new condition. Each independent path must include at least one edge that has not been crossed when determining the path. The cyclomatic number $V(G)$ of the control flow graph G is equal to its maximum number of linearly independent paths.

The empirical approach defines the concept of probability by the definition of sample space, which represents the outcome of an experiment. In our case, the sample space is

the set $\mathcal{P}$ of all paths p_i resulting from the execution of the program under test.

The set $\mathcal{P}$ may be infinite or very large. $\mathcal{P}_L$ denotes a sample (set of paths) obtained by running a program N times. In $\mathcal{P}_L$, each path p_i will appear with a certain frequency f_i. We define the probability of path p_i in $\mathcal{P}_L$ as the relative frequency $f^N(p_i) = f_i/N$.

How to choose N in order to obtain a representative sample of $\mathcal{P}$? For large samples, we can use the table of critical values of the Kolmogorov-Smirnov test [1] with desired confidence level α. It is a widely used nonparametric test for the equality of probability distributions, that can be employed to compare a sample (in our case, $\mathcal{P}_L$) with a reference probability distribution (the one associated with $\mathcal{P}$). The Kolmogorov-Smirnov statistic quantifies a distance K_N between the empirical cumulative distribution function of the sample and the cumulative distribution function of the reference.

Running N times can be very expensive. The goal of our method is to obtain a good approximation to $\mathcal{P}_L$, $\mathcal{P}_A$, using a generative algorithm and a much smaller sample size, n. To evaluate the closeness between the distribution frequencies of $\mathcal{P}_A$ and $\mathcal{P}_L$, we resort to the χ^2 test, which compares the result with the corresponding critical value obtained from the table with a desired confidence level α.

3. The method

Step 1: The objective of this step is to draw the control flow graph from the source code of program module M. This means replacing if-structures by decision nodes, while-do and do-while structures by loop nodes, for-do structures and actions by activity nodes. Junction nodes are included in the end, when a node with several inputs needs to join them into a single input.

Step 2: Probabilities must be assigned to decision and loop nodes. Since we do not have a previous knowledge of the program behavior, these probabilities are estimated from the results of a number n of program runs, using as input a random sample from the program's domain. These values will enable us to calculate the probability of an execution path, by simply multiplying the probabilities associated with decision and loop nodes in the selected path.

Step 3: After assigning probabilities to decision and loop nodes, we generate execution paths of program M and calculate the probability of each path. This temporary set of paths is denoted by $\mathcal{P}_{A'}$. Each path is identified by a sequence of nodes being traversed, e.g., the paths 1-2-3-8 or 1-2-3-4-5-6-7-2-3-8. Then, after generating $\mathcal{P}_{A'}$, we obtain the execution path set $\mathcal{P}_A$, formed by the paths from $\mathcal{P}_{A'}$ whose sum of probabilities is larger than a required confidence level α.

Step 4: Each element of the execution path set will be used to generate one element of the test case set. Although a computationally amenable solution could be attempted by the use of any of the above cited approaches, we recommend a simpler ad-hoc procedure, in which the developer translates the required node transitions into natural language specifications for the input values.

4. Validation

The method was applied in a set of fourteen programs written in C/C++. For each program, the path set $\mathcal{P}_L$ along with its associated pdf f^* were generated. Each pdf f^* was then compared to f, obtained by our method. The goodness of fit between each f^* and f in the set was verified using the χ^2 test with a confidence level of 95% (in the modern approach, the result has credibility when the confidence level of the experiment is equal or greater then 95%).

The null hypothesis was formulated as $H_0 : f^* \neq f$

These results showed that H_0 is rejected thirteen times out of fourteen. The only program for which H_0 was not rejected has a particular case of simultaneous recursion, i.e., it executes several times recursive operations in parallel.

5 Conclusions

The evaluation tests confirmed that not only the method is simple to use, but also provides, in a computationally efficient way, a good approximation when comparing $\mathcal{P}_A$ (the set of paths which is the outcome of the method) and $\mathcal{P}_L$ (a large sample of the real set $\mathcal{P}$ of execution paths).

References

[1] P. G. Hoel. *Introduction to Mathematical Statistics*. John Wiley and Sons, California, USA, 1966.

[2] G. Janardhanudu. White box testing. Technical Report 259-BSI, Cigital Inc., Sept. 2009.

[3] T. J. McCabe. A complexity measure. *IEEE Transactions on Software Engineering*, 2(4):308–320, December 1976.

[4] R. S. Pressman. *Software Engineering - a pratictioner's approach*. McGraw-Hill, New York, USA, 2001.

[5] M. K. Singh and L. Rakesh. Mathematical principles in software quality engineering. *International Journal of Computer Science and Information Security*, 7(3):178–184, March 2010.

Architecture for personalized and semantic Information Retrieval: approach based on content's re-indexing using user's profile

Azza Harbaoui, Malek Ghenima, Henda Ben Ghezala
RIADI Laboratory
Nationnal School of Science, La Manouba University
La Manouba 2010

Sahbi Sidhom
LORIA Laboratory
Nancy University
54506 Vandoeuvre, FRANCE

Abstract— **Information Retrieval is a process made by a user to obtain relevant information which meets his needs using an Information Retrieval System (IRS).The main objective of IR systems is to select relevant documents related to user's information need, from collection of documents. This paper intends to address some reflections coming from the result of survey of information retrieval systems in order to create systems that, by taking into account user's profiles, can better satisfy the users of IR systems.**
***Keywords*— information retrieval, user profile, personalized IR, indexing, user re-indexing, ontology**

I. Introduction

As an Academic field of study Information Retrieval (IR) might be defined thus "Information Retrieval is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within a large collections (usually stored on computer)" [2]. Furthermore, in the case of textual documents in which we are interest a significant part of difficulties are due to the difficulty lies in ambiguity inherent to human languages [1] and the carelessness of the user.
This paper is a proposal of some ideas in order to improve information retrieval. It aims to develop a system for personalized IR using ontology classes to represent personal information including user profile, preference, culture, and interests

II adaptative information retrieval

The goal of the adaptive information retrieval is to adapt the process of information retrieval in order to return relevant results to the user. Techniques developed in adaptive information retrieval [4] focus mainly on the assistance in the query's reformulation.

1. *Reformulation of the query*

It aims to generate a new query without explicit retroaction of the user, just by exploring the first documents presented by the system as an answer to the user's query (blind feedback)[6].Different techniques were introduced in the literature, in models of information retrieval such as Vector Space Model, and the probabilistic model. We quote the techniques of the reformulation of the queries by relevance feedback, techniques of disambiguation of the queries sense or the techniques of clustering.

2. *Disambiguation of the query's sense*

The aim of these techniques is to assist the user in order to better express his need by adding additional semantic resources[8] (dictionaries, networks, thesauruses).Most of them are based on exploring interactive interfaces of clarification based on ontology

3. Techniques of Clustering

In order to face major growth of the web, and the difficulties faced by the classic search engines, the aim of the clustering techniques is to group the results into categories such as Grouper [9],Vivissimo, Kartoo, as well as techniques of repertorisation of the web into taxonomy of concepts (Google hierarchy).All these techniques[7] were developed in order to satisfy the user and to simplify his navigation. The theoretical study that we carried out reveals that beside the earnings bring by the techniques of adaptive information retrieval, there still some limitation such as the context, explicit interaction, impact of the user's familiarity with the subject.

III Contextual and personalized information retrieval

According to [10], the contextual information retrieval is defined as *"combine search technologies and knowledge about the query and user context into a single framework in order to provide the most appropriate answer for a user's information needs"*.

In the literature [3], the context of research is bound to dimensions dependent to the user, the query and the environment of research. We are interested in personalized information retrieval which is a specific branch of contextual information retrieval. The goal of personalization [5] in information retrieval is to tailor the search engine results. The first approaches of the contextual information retrieval have focused on the context of the user represented by his profile.

This context includes his interests, his votes, his comments, and also his knowledge. In fact people's future preferences and needs must be able to personalize services, the same way recommender systems do. The personalization is a process which changes the feature, the interface, the content information or the aspect of a system in order to

increase his personal relevance related with the socio-demographic characteristics declared by the user and his observed behaviour contained in what we call user model.

IV Proposed approach

The reflections presented in this paper are situated precisely in the context of personalized access to information. We are now focusing on the construction of the proposed architecture (Fig 1).

The objective is to add semantic value to information retrieval systems towards a new indexing process or re-indexing of the contents. We are interested to imply the user, his profile, his traces, his culture, and his point of view in order to collect all these points into ontology of interest.

The system consists of two main parts: classic IR and personalized IR.The parts are closely interrelated in the sense that the outputs of module are inputs of the other. After implementing the Baseline System containing the process of indexing, the users are classified into two categories: user having a profile and user not having a profile. The user profile data comprising his personal information, preferences, domain of competence, interests, and also comments and votes (that will be used in the process of re-indexing).At the first connection to the system, the construction of the user's profile is done and an instance of profile's ontology is created too. When the user will have a profile, a categorization of users will be made into group of users. It has the advantage of having typical information with the opportunity to refine it as and when.

We implement a generic indexing tool for structured, semi-structured and non-structured documents. We used the vector space model .For non-structured documents or "Full text", we use a test corpus CACM(Communications of the ACM) composed of 3204 documents with the title and links to bibliographic citation in the field of computer science,64 queries and a file of assessments.

We followed the steps bellow to generate the inverted documents file and the queries inverted file:

-Tokenization

-Dropping common terms (Stop words)

-Stemming and lemmatization.

V Conclusion

Although a considerable number of works focused on information retrieval, some important challenges for the research community still remain. This paper provides a brief survey of the evolution of information retrieval from the classic to the contextual, the problems faced. We have proposed architecture of personalization by implying the user with representing his interests and profile into an ontology [8] of interest, then using this ontology for re-indexing of the contents. In fact, a key part of a personalization system is the user model. We have to develop an expressive model in order to encompass all aspects of the user taking into account the environment that the user interacts with and try to identify changes in user interests and needs. Finally we can say that the ultimate goal of personalization system being the satisfaction of the user. To reach this goal, the user has to be implied in the construction process in order to add the semantic value to the information retrieval.

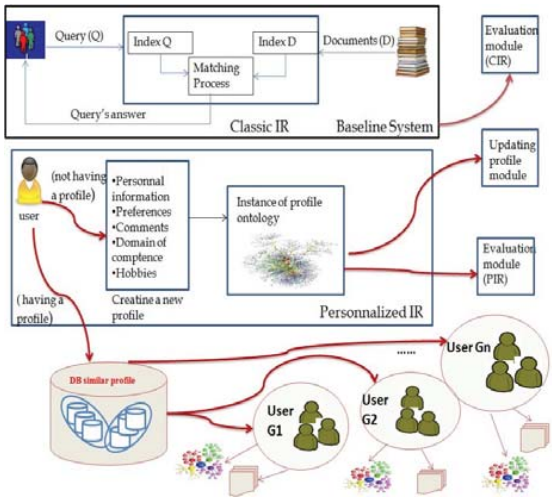


Figure1: Proposed Architecture

References

[1] Baeza-Yates, R. and B. Ribeiro-Neto. Modern Information Retrieval.Addison-Wesley: New Jersey, 2000

[2] C. Bourne and B. Anderson. Dialog labworkbook. PaloAlto, Californie, USA, Second edition, Looked Information Systems, 1979.

[3] C. Van Rijsbergen. A non-classical logic for information retrieval. The computer journal, 29(6), 1986,481.485,

[4] D. Grangier, A. Vinciarelli, H. Bourlard, Information retrieval on noisy text, september 2003.

. [5] I. Ruthven and M. Lalmas. A survey on the use of relevance feedback for information access systems. Knowledge Engeneering review, 2003, 18(2) :95.145.

[6] J. Lafferty et C. Zhai. Probabilistic relevance models basedon document and query generation., volume 13. Kluwer Academic, 2003.

[7] JF. Tanudjaja and L. Mui. Persona : A contextualized and personalized web search. In Proc 35th Hawaii International Conference on System Sciences, Big Island, Hawaii, January 2002, page 53.

[8] J. Trajkova and S. Gauch. Improving ontology-based user pro_les. In Proceedings of the 8th Conference of Recherche d'Information Assistée par Ordinateur, University of Avignon (Vaucluse), France, April 26-28 2004, pages 380. 389.

[9] J. Wen, N. Lao, and W. Y. Ma. Probabilistic model for contextual retrieval. In Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval, Shef_eld, United Kingdom,August 2004, pages 57.63

[10] M. Lalmas, A. MacFarlane, S. M. R¨uger, A. Tombros, T. Tsikrika, and A. Yavlinsky, editors. Advances in Information Retrieval, 28th European Conference on IR Research, ECIR 2006, London, Proceedings, volume 3936 of Lecture Notes in Computer Science. Springer,UK, April 10-12, 2006

Author's Index

Reviewer's Index

A

Alain Abran
Silvia Teresita Acuna
Taiseera Albalushi
Edward Allen

B

Rosa Badia
Doo-hwan Bae
Ebrahim Bagheri
Rami Bahsoon
Xiaoying Bai
Purushotham Bangalore
Benoit Baudry
Fevzi Belli
Ateet Bhalla
Alessandro Bianchi
Karun N. Biyani

C

Kai-yuan Cai
Borzoo Bonakdarpour
Jean-michel Bruel
Gerardo Canfora
Jaelson Castro
Raul Garcia Castro
Cagatay Catal
Christine Chan
Keith Chan
Kuang-nan Chang
Ned Chapin
Gerardo Antonio Castanon Avila
Shu-ching Chen
Zhenyu Chen
Yung-pin Cheng
Stelvio Cimato
Peter Clarke
Esteban Clua
Nelly Condori-fernandez
Julita Corbalan
Fabio M. Costa
Maria Francesca Costabile
Karl Cox
Jose Luis Cuadrado
Juan J. Cuadrado-gallego

D

Dilma Da Silva
Ernesto Damiani
Jose Luis De La Vara
Deepak Dhungana, Lero
Massimiliano Di Penta
Jin Song Dong
Jing Dong
Dirk Draheim
Weichang Du
Philippe Dugerdil
Hector Duran

E

Christof Ebert
Ali Ebnenasir
Raimund Ege
Magdalini Eirinaki
Faezeh Ensan
Onyeka Ezenwoye

F

Davide Falessi
Behrouz Far
Robert Feldt
Eduardo B. Fernandez
Marian Fernandez De Sevilla
Gerardo Fernandez Escribano
Scott D. Fleming
Liana Fong
Renata Fortes
Fulvio Frati

G

Jerry Gao
Kehan Gao
Felix Garcia
Ignacio Garcia Rodriguez De Guzman
Itana Gimenes
Swapna Gokhale

Wolfgang Golubski
Jeff Gray
Desmond Greer
Eric Gregoire
Christiane Gresse Von Wangenheim

H

Xudong He
Miguel Herranz
Howard Ho
Mong Fong Horng
Shihong Huang

J

Clinton Jeffery
Jason Jung
Natalia Juristo

K

Selim Kalayci
Eric Kasten
Taghi Khoshgoftaar
Nicholas Kraft
Sandeep Kulkarni
Vinay Kulkarni
Gihwon Kwon

L

Konstantin Laufer
Juan Carlos Lavariega Jarquin
Jeff Lei
Bixin Li
Ming Li
Tao Li
Chien-hung Liu
Shih-hsi Liu
Xiaodong Liu
Yan Liu
Yi Liu
Hakim Lounis
Joan Lu
Heiko Ludwig

M

Jose Carlos Maldonado
Antonio Mana
Vijay Mann
Hong Mei
Hsing Mei
Emilia Mendes
Ali Mili
Alok Mishra
Ana M. Moreno

N

Antonio Navidad Pineda
Kia Ng
Ngoc Thanh Nguyen
Allen Nikora

P

Kunal Patel
Eric Pardede
Antonio Piccinno
Alfonso Pierantonio

Q

Zhou Qiang

R

Rick Rabiser
Lukasz Radlinski
Damith C. Rajapakse,
Rajeev Raje
Jose Angel Ramos
Marek Reformat
Robert Reynolds
Daniel Rodriguez
Ivan Rodero

S

Samira Sadaoui
Masoud Sadjadi
Ramon Sagarna
Claudio Sant'Anna
Salvatore Alessandro Sarcia

Douglas Schmidt
Andreas Schoenberger
Naeem (jim) Seliya
Tony Shan
Yi-dong Shen
Michael Shin
Yang Qiu Song
George Spanoudakis
Gerson Sunye

T

Jeff Tian
Peter Troger
Genny Tortora
Mark Trakhtenbrot
T.h. Tse

V

Giorgio Valle
Michael Vanhilst
Sylvain Vauttier
Silvia Vergilio
Akshat Verma
Arndt Von Staa

W

Huanjing Wang
Limin Wang
Hironori Washizaki
Victor Winter
Guido Wirtz
Franz Wotawa

X

Haiping Xu

Y

Jijiang Yan
Chi-lu Yang
Hongji Yang
Junbeom Yoo
Huiqun Yu

Z

Cui Zhang
Du Zhang
Jing Zhang
Min-ling Zhang
Yong Zhang
Zhenyu Zhang
Hong Zhu
Xingquan Zhu
Eugenio Zimeo

Poster/Demo Presenter's Index

SEKE 2012 Call For Papers

The Twenty-Fourth International Conference on Software Engineering and Knowledge Engineering

www.ksi.edu/seke/seke12.html

Hotel Sofitel, Redwood City, San Francisco Bay, USA
July 1 - July 3, 2012

Organized by

Knowledge Systems Institute Graduate School

The Twenty-Fourth International Conference on Software Engineering and Knowledge Engineering (SEKE 2012) will be held at the Hotel Sofitel, Redwood City, California, USA, July 1-3, 2012.

The conference aims at bringing together experts in software engineering and knowledge engineering to discuss on relevant results in either software engineering or knowledge engineering or both. Special emphasis will be put on the transference of methods between both domains. Submission of papers and demos are both welcome.

TOPICS

Agent architectures, ontologies, languages and protocols
Multi-agent systems
Agent-based learning and knowledge discovery
Interface agents
Agent-based auctions and marketplaces
Artificial life and societies
Secure mobile and multi-agent systems
Mobile agents
Mobile Commerce Technology and Application Systems
Mobile Systems

Autonomic computing
Adaptive Systems
Integrity, Security, and Fault Tolerance
Reliability
Enterprise Software, Middleware, and Tools
Process and Workflow Management
E-Commerce Solutions and Applications
Industry System Experience and Report

Service-centric software engineering
Service oriented requirements engineering
Service oriented architectures
Middleware for service based systems
Service discovery and composition
Quality of services
Service level agreements (drafting, negotiation, monitoring and management)
Runtime service management
Semantic web

Requirements Engineering
Agent-based software engineering
Artificial Intelligence Approaches to Software Engineering
Component-Based Software Engineering
Automated Software Specification
Automated Software Design and Synthesis
Computer-Supported Cooperative Work
Embedded and Ubiquitous Software Engineering
Measurement and Empirical Software Engineering
Reverse Engineering
Programming Languages and Software Engineering
Patterns and Frameworks
Reflection and Metadata Approaches
Program Understanding

Knowledge Acquisition
Knowledge-Based and Expert Systems
Knowledge Representation and Retrieval
Knowledge Engineering Tools and Techniques
Time and Knowledge Management Tools
Knowledge Visualization
Data visualization
Uncertainty Knowledge Management
Ontologies and Methodologies
Learning Software Organization

Tutoring, Documentation Systems
Human-Computer Interaction
Multimedia Applications, Frameworks, and Systems
Multimedia and Hypermedia Software Engineering

Smart Spaces
Pervasive Computing
Swarm intelligence
Soft Computing

Software Architecture
Software Assurance
Software Domain Modeling and Meta-Modeling
Software dependability
Software economics
Software Engineering Decision Support
Software Engineering Tools and Environments
Software Maintenance and Evolution
Software Process Modeling
Software product lines
Software Quality
Software Reuse
Software Safety
Software Security
Software Engineering Case Study and Experience Reports

Web and text mining
Web-Based Tools, Applications and Environment
Web-Based Knowledge Management
Web-Based Tools, Systems, and Environments
Web and Data Mining

CONFERENCE SITE (HOTEL INFORMATION)

The SEKE 2012 Conference will be held at the Hotel Sofitel, Redwood City, California, USA. The hotel has made available for these limited dates (6/29 - 7/5/2012) to SEKE 2012 attendees a discount rate of USD$85 for single/double, not including sales tax.

INFORMATION FOR AUTHORS

Papers must be written in English. An electronic version (Postscript, PDF, or MS Word format) of the full paper should be submitted using the following URL: http://conf.ksi.edu/seke2012/submit/SubmitPaper.php.
Please use Internet Explorer as the browser. Manuscript must include a 200-word abstract and no more than 6 pages of 2-column formatted Manuscript for Conference Proceedings (include figures and references but exclude copyright form).

INFORMATION FOR REVIEWERS

Papers submitted to SEKE 2012 will be reviewed electronically. The users (webmaster, program chair, reviewers...) can login using the following URL: http://conf.ksi.edu/seke2012/review/pass.php.
If you have any questions or run into problems, please send e-mail to:
E-mail: seke12@ksi.edu.

SEKE 2012 Conference Secretariat
Knowledge Systems Institute
3420 Main Street
Skokie, IL 60076 USA
Tel: 847-679-3135
Fax: 847-679-3166
E-mail: seke12@ksi.edu

IMPORTANT DATES

March 1, 2012	*Paper submission due*
April 20, 2012	*Notification of acceptance*
May 10, 2012	*Early registration deadline*
May 10, 2012	*Camera-ready copy*

Proceedings of the Twenty-Third International Conference on Software Engineering & Knowledge Engineering

SEKE

Miami, July 7-9 2011

Printed by
Knowledge Systems Institute
3420 Main Street
Skokie, Illinois 60076
(847) 679-3135
info@ksi.edu
www.ksi.edu
Printed in USA, 2011
ISBN 1-891706-29-2 (paper)