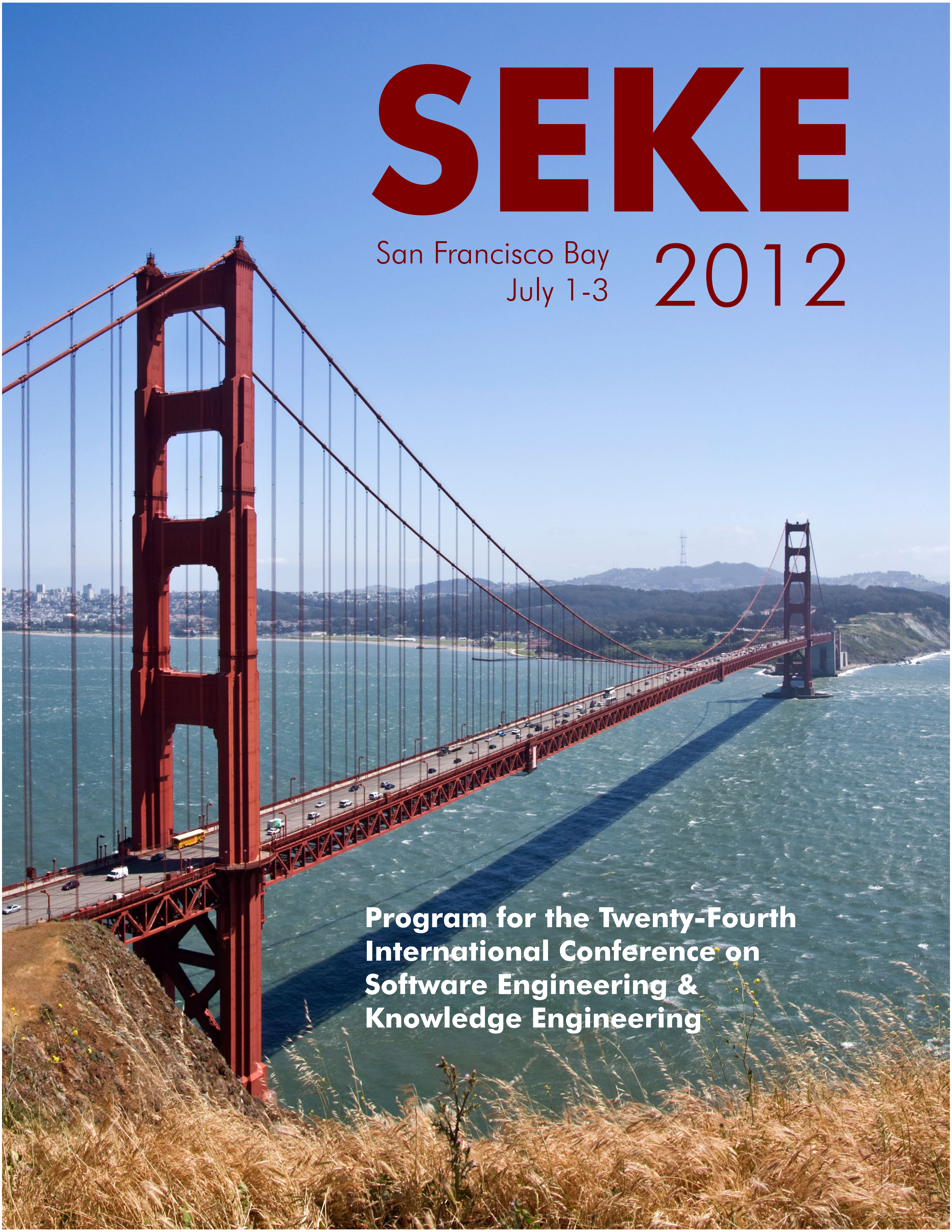
SEKE

San Francisco Bay
July 1-3

2012



Program for the Twenty-Fourth International Conference on Software Engineering & Knowledge Engineering

PROCEEDINGS

SEKE 2012

The 24th International Conference on Software Engineering & Knowledge Engineering

Sponsored by

Knowledge Systems Institute Graduate School, USA

Technical Program

July 1-3, 2012
Hotel Sofitel, Redwood City, San Francisco Bay, USA

Organized by

Knowledge Systems Institute Graduate School

ISBN-10: 1-891706-31-4 (paper)
ISBN-13: 978-1-891706-31-8

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

Foreword

This year marks the 24th anniversary for the International Conference on Software Engineering and Knowledge Engineering (SEKE). For nearly a quarter of century, SEKE has established itself as a major international forum to foster, among academia, industry, and government agencies, discussion and exchange of ideas, research results and experience in software engineering and knowledge engineering. The SEKE community has grown to become a very important and influential source of ideas and innovations on the interplays between software engineering and knowledge engineering, and its impact on the knowledge economy has been felt worldwide. On behalf of the Program Committee Co-Chairs and the entire Program Committee, I would like to extend to you the warmest welcome to SEKE 2012.

We received 219 submissions from 30 countries this year. Through a rigorous review process where a majority (86 percent) of the submitted papers received three reviews, and the rest with two reviews, we were able to select 59 full papers for the general conference (27 percent), 18 full papers for three special tracks (8 percent), and 60 short papers (27 percent), for presentation in thirty nine sessions during the conference. In addition, the technical program includes excellent keynote speech and panel discussions, and three special tracks: Software Engineering with Computational Intelligence and Machine Learning, Petri Nets for SEKE, and Software Testing and Analysis with Intelligent Technologies.

The high quality of the SEKE 2012 technical program would not have been possible without the tireless effort and hard work of many individuals. First of all, I would like to express my sincere appreciation to all the authors whose technical contributions have made the final technical program possible. I am very grateful to all the Program Committee members whose expertise and dedication made my responsibility that much easier. My gratitude also goes to the keynote speaker and panelists who graciously agreed to share their insight on important research issues, to the conference organizing committee members for their superb work, and to the external reviewers for their contribution.

Personally, I owe a debt of gratitude to a number of people whose help and support with the technical program and the conference organization are unfailing and indispensable. I am deeply indebted to Dr. S. K. Chang, Chair of the Steering Committee, for his constant guidance and support that are essential to pull off SEKE 2012. My heartfelt appreciation goes to Dr. Masoud Sadjadi, the Conference Chair, for his help and experience, and to the Program Committee Co-Chairs, Dr. Marek Reformat of University of Alberta, Canada, Dr. Swapna Gokhale of University of Connecticut, USA, and Dr. Jose Carlos Maldonado of University of Sao Paulo, Brazil, for their outstanding team work. I am truly grateful to the special track organizers, Dr. Taghi Khoshgoftaar of Florida Atlantic University, Dr. Marek Reformat of University of Alberta, Canada, Dr. Dianxiang Xu of Dakota State University, South Dakota, Dr. Haiping Xu of University of Massachusetts Dartmouth, Dr. Zhenyu Chen of Nanjing University, China, and Dr. Zheng Li of Beijing University of Chemical Technology, China, for their excellent job in organizing the special sessions. I would like to express my great appreciation to all the Publicity Co-Chairs, Dr. Xiaoying Bai of Tsinghua University, China, Dr. Raul Garcia Castro of Universidad Politecnica de Madrid, Spain, Shihong Huang of Florida Atlantic University, and Dr. Haiping Xu of University of Massachusetts Dartmouth, for their important contributions, to the Asia, Europe, and South America liaisons, Dr. Hironori Washizaki of Waseda University, Japan, Dr. Raul Garcia Castro of Universidad Politecnica de Madrid, Spain, and Dr. Jose Carlos Maldonado of University of Sao Paulo, Brazil, for their great efforts in helping expand the SEKE community, and to the Poster/Demo session Co-Chairs, Dr. Farshad Samimi of Trilliant and Dr. Ming Zhao of Florida International University, for their work. Last but certainly not the least, I must acknowledge the important contributions the following KSI staff members have made: David Huang, Rachel Lu, Alice Wang, and Dennis Chi. Their timely and dependable support and assistance throughout the entire process have been truly remarkable. It has been a great pleasure to work with all of them.

Finally, I hope you will find your participation in the SEKE 2012 programs rewarding. Enjoy your stay in the San Francisco Bay area.

Du Zhang
SEKE 2012 Program Chair

The 24th International Conference on Software Engineering & Knowledge Engineering (SEKE 2012)

July 1-3, 2012
Hotel Sofitel, Redwood City, San Francisco Bay, 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

Jerry Gao, *San Jose State University, USA*
Natalia Juristo, *Madrid Technological University, Spain*
Taghi Khoshgoftaar*, Florida Atlantic University, USA*
Guenther Ruhe, *University of Calgary, Canada*

Conference Chair

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

Program Chair

Du Zhang, *California State University Sacramento, 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*

Thomas Alspaugh, *Georgetown University, USA*

Doo-hwan Bae, *Korea Advanced Institute of Science and Technology, Korea*

Ebrahim Bagheri, *National Research Council Canada, Canada*

Hamid Bagheri, *University of Virginia, USA*

Rami Bahsoon, *University of Birmingham, United Kingdom*

Xiaoying Bai, *Tsinghua University, China*

Purushotham Bangalore, *University of Alabama at Birmingham, USA*

Ellen Francine Barbosa, *University of Sao Paulo, Brazil*

Fevzi Belli, *Univ. Paderborn, Germany*

Ateet Bhalla, *NRI Institute of Information Science and Technology, India*

Swapan Bhattacharya, *Jadavpur University, India*

Alessandro Bianchi, *Department of Informatics - University of Bari, Italy*

Karun N. Biyani, *Michigan State University, USA*

Borzoo Bonakdarpour, *University of Waterloo, Canada*

Ivo Bukovsky, *Czech Technical University in Prague, Czech Republic*

Kai-yuan Cai, *Beijing University of Aeronautics and Astronautics, China*

Gerardo Canfora, *Universita del Sannio, Italy*

Jaelson Castro, *Universidade Federal de Pernambuco - UFPE, Brazil*

Raul Garcia Castro, *Universidad Politecnica de Madrid, Spain*

Cagatay Catal, *Istanbul Kultur University, Turkey*

Peggy Cellier, *IRISA/INSA of Rennes, France*

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*

Shu-Ching Chen, *Florida International University, USA*

Zhenyu Chen, *Nanjing University, China*

Stelvio Cimato, *The University of Milan, Italy*

Peter Clarke, *Florida International University, USA*

Esteban Clua, *Universidade Federal Fluminense, Brasil*

Nelly Condori-fernandez, *University of Twente, The Netherlands*

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*

Ernesto Damiani, *The University of Milan, Italy*

Dilma Da Silva, *IBM, USA*

Jose Luis De La Vara, *Simula Research Laboratory, Norway*

Marian Fernandez De Sevilla, *University of Alcala, Spain*

Scott Dick, *University of Alberta, Canada*

Massimiliano Di Penta, *University of Sannio, Italy*

Jing Dong, *University of Texas at Dallas, USA*

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*

Davide Falessi, *University of Rome, TorVergata, Italy*

Behrouz Far, *University of Calgary, Canada*

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*

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*

Katarina Grolinger, *University of Western Ontario, Canada*

Hao Han, *National Institute of Informatics, Japan*

Xudong He, *Florida International University, USA*

Miguel Herranz, *University of Alcala, Spain*

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*

Jun Kong, *North Dakota State University, USA*

Nicholas Kraft, *The University of Alabama, USA*

Anesh Krishna, *Curtin University of Technology, Australia*

Sandeep Kulkarni, *Michigan State University, USA*

Vinay Kulkarni, *Tata Consultancy Services, India*

Gihwon Kwon, *Kyonggi University, South Korea*

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*

Yuan-Fang Li, *Monash University, Australia*

Qianhui Liang, *Singapore Management University, Singapore*

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*

Jose Carlos Maldonado, *ICMC-USP, Brazil*

Antonio Mana, *University of Malaga, Spain*

Vijay Mann, *IBM, India*

Riccardo Martoglia, *University of Modena and Reggio Emilia, Italy*

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*

Kia Ng, *ICSRiM - University of Leeds, United Kingdom*

Ngoc Thanh Nguyen, *Wroclaw University of Technology, Poland*

Allen Nikora, *Jet Propulsion Laboratory, USA*

Edson Oliveira Jr., *State University of Maringa, Brazil*

Kunal Patel, *Ingenuity Systems, USA*

Xin Peng, *Fudan University, China*

Antonio Piccinno, *University of Bari, Italy*

Alfonso Pierantonio, *University of L'Aquila, Italy*

Antonio Navidad Pineda, *University of Alcala, Spain*

Rick Rabiser, *Johannes Kepler University, Austria*

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*

Ivan Rodero, *The State University of New Jersey, USA*

Daniel Rodriguez, *Universidad de Alcala, Spain*

Samira Sadaoui, *University of Regina, Canada*

Masoud Sadjadi, *Florida International University, USA*

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*

Rajan Shankaran, *Macquarie University, Australia*

Michael Shin, *Computer Science/Texas Tech University, USA*

Qinbao Song, *Xi'an Jiaotong University, China*

George Spanoudakis, *City University, United Kingdom*

Jing Sun, *University of Auckland, New Zealand*

Yanchun Sun, *Peking University, China*

Gerson Sunye, *Institut de Recherche en Informatique et Systemes Aleatoires, France*

Jeff Tian, *Southern Methodist University, USA*

Genny Tortora, *University of Salerno, Italy*

Mark Trakhtenbrot, *Holon Institute of Technology, Israel*

Peter Troeger, *Universitat zu Potsdam, Germany*

T.h. Tse, *The University of Hong Kong, Hong Kong*

Giorgio Valle, *The University of Milan, Italy*

Sylvain Vauttier, *Ecole des mines d'Ales, France*

Silvia Vergilio, *Federal University of Parana (UFPR), Brazil*

Akshat Verma, *IBM, India*

Sergiy Vilkomir, *East Carolina University, USA*

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*

Eric Wong, *University of Texas, USA*

Franz Wotawa, *TU Graz, Austria*

Dianxiang Xu, *Dakota State University, USA*

Haiping Xu, *University of Massachusetts Dartmouth, USA*

Chi-lu Yang, *Taiwan Semiconductor Manufacturing Company Ltd., Taiwan*

Hongji Yang, *De Montfort University, United Kingdom*

Ji-Jian Yang, *National TsingHua University, China*

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*

Hongyu Zhang, *Tsinghua 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

Farshad Samimi, *Trilliant, USA*
Ming Zhao, *Florida Int'l University, USA*

Publicity Co-Chairs

Xiaoying Bai, *Tsinghua University, China*
Raul Garcia Castro*, Universidad Politecnica de Madrid, Spain*
Shihong Huang, *Florida Atlantic University, USA*
Haiping Xu, *University of Massachusetts Dartmouth, 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*
Noorjhan 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*
Alice Wang, *Knowledge Systems Institute Graduate School, USA*

Table of Contents

Data Mining

Requirement Engineering

Special Session: Software Engineering with Comp. Intelligence & Machine Learning

Case Study

Aspect-Oriented SE

Program Understanding

Component-based SE

Software Quality

Agent-based Learning

Special Session: Software Testing and Analysis with Intelligent Technology

Service-Centric SE

Petri Nets for SEKE

Security and Privacy

Ontologies and Architecture

Testing

Programming Languages

Patterns and Frameworks

Modeling

Tools and Environment

AI for SE

Human-Computer Interaction

Semantic Web

Software Product Lines

Dependability and Maintenance

Automated Software Specification

Knowledge Acquisition and Visualization

UML

Measurement and Adaptive Systems

Agents and Mobile Systems

Poster/Demo

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

Keynote
On the Naturalness of Software

Professor Prem Devanbu
Department of Computer Science
University of California Davis

Abstract

Natural Language processing (NLP) has been revolutionized by statistical language models, which capture the high degree of regularity and repetition that exists in most human speech and writing. These models have revolutionized speech recognition and translation. We have found, surprisingly, that “natural software”, viz., code written by people is also highly repetitive, and can be modeled effectively by language models borrowed from NLP. We present data supporting this claim, discuss some early applications showcasing the value of language models of code, and present a vision for future research in this area.

About the Speaker

Prem Devanbu received his B.Tech from the Indian Institute of Technology in Chennai, India, before you were born, and his PhD from Rutgers in 1994. After spending nearly 20 years at Bell Labs and its various offshoots, he escaped New Jersey to join the CS faculty at UC Davis in late 1997. He has published over 100 papers, and has won ACM SIGSOFT distinguished paper awards at ICSE 2004, ICSE 2009, and ASE 2011, and the conference best paper awards at MSR 2010 and ASE 2011. He has been program chair of ACM SIGSOFT FSE (in 2006) and ICSE (in 2010). He has served on the Editorial boards of both IEEE Transactions on Software Engineering and the ACM equivalent. He has worked in several different areas over a 25 year research career, including logic programming, knowledge representation, software tools, secure information storage in the cloud, and middleware. For the past years, he has been fascinated by the abundance of possibilities in the veritable ocean of data that is available from open-source software projects. He is funded by grants from the NSF, the AFOSR, Microsoft Research, and IBM.

Panel on Future Trends of Software Engineering and Knowledge Engineering

Du Zhang
California State University, USA
(Moderator)

The International Conference on Software Engineering and Knowledge Engineering (SEKE) is celebrating its 24th anniversary this year. For nearly a quarter of century, while SEKE has established itself as a major international forum to promote research and practice in software engineering and knowledge engineering, the computing fields have undergone profound changes. Today, our daily lives are intimately intertwined with artifacts that are the results of software engineering and knowledge engineering. What will the future hold for SEKE as a field of inquiry in the next ten years? What are the challenges that lie ahead? What can we do as a community to further our agenda on SEKE? Toward illuminating our path to the future, an excellent panel of experts has been assembled. Panelists will share their insight on the future trends of software engineering and knowledge engineering. We hope you will find the panel an inspiring impetus for the continued growth of SEKE in the years to come.

Software Engineering of Autonomic Clouds

Masoud Sadjadi
Florida International University, USA
(Panelist)

Autonomic or self-managing clouds are becoming prevalent software deployment environments for applications ranging from commerce (e.g., banking), to education (e.g., virtual labs), to research (e.g., high-performance computing). Unfortunately, traditional approaches to software engineering are not applicable to the specific characteristics of autonomic clouds, which are becoming a major part of every software application’s solution domain. Therefore, there is a desperate need for a paradigm shift in how software applications are designed, developed, tested, deployed, hosted, and consumed in the clouds. One example of the specific characteristics of autonomic clouds is the concept of on-demand services leasing, which has major impacts on the growth of new businesses, from their inception to booming popularity. To respond to such needs, service providers face major challenges when trying to keep up with their promise of infinite capacity with unconditional elasticity.

Big Data in Software Engineering: Challenges and Opportunities

Taghi Khoshgoftaar
Florida Atlantic University, USA
(Panelist)

The field of software engineering has changed drastically in the past 20 years. Although traditional quality assurance approaches such as unit tests and change tracking remain essential tools, these approaches can be easily overwhelmed by the sheer volume of modules, bugs, programmers, and projects managed in large software development firms. To deal with this "Big Data," a new class of software engineering tools are needed: those from the fields of data mining and machine learning. By employing techniques specifically designed to sift through enormous datasets and identify the elements in need of human attention, data mining tools permit software practitioners to focus valuable human resources where they are needed most. I will discuss a number of topics concerning the use of data mining to manage Big Data in the context of software engineering, including software metric selection, data balance issues, and quality of data.

Knowledge Engineering, Operational Research and AI: the Time to Meet

Eric Grégoire
Université d′Artois, France
(Panelist)

Although they share many paradigms, the Operational Research and Artificial Intelligence fields have often evolved separately. This last decade, both domains have come ever closer, through new insights in constraint solving and SAT-related technologies, allowing problems to be solved that were long considered out of reach. This opens new perspectives for Knowledge Engineering as well.

Computational Issues in Social Networks

Swapna S. Gokhale
University of Connecticut, USA
(Panelist)

Online social networks (OSNs) have had an enormous impact on the way people communicate and share information. Today, the population of Facebook exceeds that of the United States and Lady Gaga has more Twitter followers than the entire population of Australia! OSNs not only provide social channels for communication, but they also offer critical marketing and customer profiling tools for businesses. This revolution has precipitated a deep desire to understand the structure of OSNs, identify the latent patterns that may exist within these networks, and leverage these structures and patterns to build novel applications and services. While sociologists have researched such social networks for decades, never before has such a vast quantity of structured social network data been available for analysis. Social network analysis is thus a rapidly emerging field that combines algorithmic, graph theoretical, and data mining techniques to map, measure, and find patterns in the relationships and communication flows in massive OSN datasets. This talk will summarize the recent, state-of-the-art research in OSN analysis on topics such as topology characterization, information and influence diffusion, community detection, inferring relationship strength, microblog analysis, friend and link prediction, data anonymity, workload characterization, and security and privacy, and outline avenues for further exploration.

Web Intelligence: Representation and Processing of Knowledge with Uncertainty

Marek Reformat
University of Alberta, Canada
(Panelist)

Uncertainty is an integral component of information and knowledge. Many concepts we deal with are without precise definitions, or with unknown facts, missing or inaccurate data. Such a situation is also present on the Internet where many sources of information could be corrupted, or partially and temporally inaccessible. Our dependence on the Internet is growing with every day. We relay on it doing research, learning new things, and finding what is happening in the world and in our neighborhood. But, how much imprecision and ambiguity is out there? How many sources of data are trustworthy? How much we can relay on the web to discover new things? Additionally, uncertainty is not only associated with data and information stored on the web – the users also bring ambiguity and imprecision. In many cases, the users' behavior and decisions depend on current circumstances, users' judgments, their understanding of situations, and their needs and requirements – things that are "equipped" with ambiguity. In order to make the web a user-friendly environment where the users can easily and quickly find things they are looking for, new web utilization tools have to be developed. They should be able to deal with numerous alternatives provided by the Internet, as well as with imprecision. The purpose of this topic is to provoke discussion how critical is to address the issue of imprecision and what methods, tools and approaches would be possible solutions.

Sparse Linear Influence Model for Hot User Selection on Mining a Social Network

Yingze Wang[1], Guang Xiang[2] and Shi-Kuo Chang[1]
[1]Department of Computer Science, University of Pittsburgh, Pittsburgh, PA 15260, USA
{yingzewang,chang}@cs.pitt.edu
and
[2]School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, USA
{guangx}@cs.cmu.edu

Abstract— Social influence analysis is a powerful tool for extracting useful information from various types of social media, especially from social networks. How to identify the hub nodes (the most influential users) in a social network is a central research topic in social influence analysis. In this paper, we develop the sparse linear influence model (SLIM) to automatically identify hub nodes in an implicit network. Our model is based on the linear influence model with two main advantages: 1) we don't need to know the underlying network topology or structure as prior knowledge (e.g., connections among users); 2) it can identify a set of hub nodes specific to a given topic. To solve SLIM, which is a non-smooth optimization, we adopt the accelerated gradient descent (AGM) framework. We show that there is a closed-form solution for the key step at each iteration in AGM. Therefore, the optimization procedure achieves the optimal convergence and can be scaled to very large dataset. The SLIM model is validated on a set of 50 million tweets of 1000 users on twitter. We show that our SLIM model can efficiently select the most influential users on different sets of specific topics. We also find several interesting patterns of these influential users.

Keywords-Sparse learning, Data mining, Social influence analysis

I. INTRODUCTION

Modeling the diffusion of information has been one of core research areas for analyzing social network data [1], [2], [3]. A central question in modeling information diffusion is how to find hub nodes in the network. In this paper, we define "hub nodes" to be the most influential users for a certain topic. In the past a few years, a great effort has been devoted to identifying hub nodes [4], [5], [6], [12], [14], [15]. Most of these algorithms rely on a strong assumption that the entire topology of the network is available as prior knowledge and the information can only disseminate over the given network structure. For example, one of the state-of-the-art methods [6] ranks users based on the number of followers and PageRank, which depends on the network structure. However, in many problems, the network structure is unavailable or hard defined. Throughout this paper, we make a basic assumption: the information on multiple topics spreads through an implicit network but we can only observe the volume (the number of nodes infected) for each topic over time.

Very recently, *linear influence model* (LIM) [7] was proposed to analyze the social network with an unknown network structure. LIM models the global influence of each node through an implicit network. In particular, LIM captures the global temporal effect by modeling the number of newly infected nodes as a linear function in all other nodes infected in the past. Although LIM can roughly model the influence for each node under our basic assumption, it cannot tell which nodes are central for an interesting topic.

In this paper, we extend the LIM model so that we can identify hub nodes (or the most influential users in our problem) in an implicit information diffusion network. Our work combines the LIM model with sparse learning method. In recent years, sparse learning has become a popular tool in machine learning and statistics. By jointly minimizing a smooth convex loss and a sparsity-inducing regularizer (e.g. L1-norm), sparse learning methods can select the most relevant features from high-dimensional data. Utilizing the sparse learning framework, we introduce a special sparsity-inducing regularizer, group Lasso penalty [8], in the LIM model and propose the corresponding SLIM model (Sparse Linear Influence Model). Our SLIM will automatically set the influence factors for those non-influential nodes to zero and hence select the remaining nodes as influential ones.

Due to the non-smoothness of the penalty, the optimization problem for SLIM becomes quite challenging. To solve this optimization problem, we adopt the accelerated gradient descent framework (AGM) [9, 13]. We show that for the proximal operator, which is the key step in AGM, there is a closed-form solution. Then, we directly apply fast-iterative shrinkage-thresholding algorithm (FISTA) to solve this optimization problem, which achieves an optimal convergence rate in $O(1/T^2)$, where T is the total number of iterations.

The proposed SLIM model can be applied to many various sources of social media. In this paper, we specifically apply our SLIM model on the twitter data sets. Twitter has gained huge popularity since the first day as it launched and has drawn increasing interests from research community. Using the SLIM model, we can efficiently select the most influential twitter users for different products and brands, which can provide a lot of useful information for companies and potentially make advertisement for effectiveness.

The rest of this paper is organized as follows. We present the background of LIM model and sparse learning in Section II. In Section III, we introduce our SLIM model for hub nodes selection. An efficient and scalable optimization technique is given in Section IV. In the final Section V, we present the experimental results on real twitter data.

II. Background

A. Linear Influence Model (LIM)

We follow the notations in [7]. Assume there are N nodes (corresponds to N users) and K contagions diffused over the network (corresponds to K topics). Assuming the entire time intervals are normalized into T units: $\{0, 1, 2, \ldots, T\}$. In the original paper of LIM, it uses an indicator function $M_{u,k}(t)$ to present whether the node u got infected by the contagion k between the time interval $t-1$ and t. Therefore, the value for $M_{u,k}(t)$ will be either one or zero. In this paper, we relax this assumption by defining $M_{u,k}(t)$ to be the number of times that the node u got infected by contagion k in $[t-1, t]$. Furthermore, let $V_k(t)$ denote the total volume of contagion k between $t-1$ and t. Under the linear influence model:

$$V_k(t+1) = \sum_{u=1}^{N}\sum_{l=0}^{L-1} M_{u,k}(t-l) I_u(l+1) + \epsilon_k(t+1) \quad (1)$$

Where L is the maximum lag-length and $I_u(l)$ is the non-negative influence factor of user u at the time-lag l and $\epsilon_k(t)$ is the i.i.d. zero-mean Gaussian noise. To model the influence for each user, we need to obtain a robust estimator of $I_u(l)$. Following LIM, we could organize $V_k(t)$, $M_{u,k}(t)$ and $I_u(l)$ in a matrix form. In particular, we define the volume vector $\mathbf{V} \in \mathbb{R}^{KT}$ to be the concatenation of $\mathbf{V}_1, \ldots, \mathbf{V}_K$ where each $\mathbf{V}_k = (V_k(1), \ldots, V_k(T))$; the user's influence vector $\mathbf{I} \in \mathbb{R}^{NL}$ to be the concatenation of $\mathbf{I}_1, \ldots, \mathbf{I}_N$ where each $\mathbf{I}_u = (I_u(1), \ldots, I_u(L))$. The matrix $\mathbf{M} \in \mathbb{R}^{KT \times NL}$, whose elements are $M_{u,k}(t)$, is organized so that (1) can be written in a matrix form

$$\mathbf{V} = \mathbf{M} \cdot \mathbf{I} + \boldsymbol{\epsilon}, \qquad \mathbf{I} \geq 0 \quad (2)$$

Where $\boldsymbol{\epsilon} \in \mathbb{R}^{KT}$ is the vector of noise. For the details of constructing $\mathbf{M}$, please refer to [7].

Based on (2), we can formulate the problem of predicting $\mathbf{I}$ by a non-negative least square problem:

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}\|\mathbf{V} - \mathbf{M} \cdot \mathbf{I}\|_2^2 \\ \text{subject to} & \mathbf{I} \geq 0 \end{array}$$

Where $\|\cdot\|_2$ is the vector l_2-norm. From the estimated $\mathbf{I}$, we obtain the pattern of the influence for each user.

B. Sparse Learning

We present the necessary background for sparse learning, starting with the high-dimensional linear regression model. Let $\mathbf{X} \in \mathbb{R}^{n \times p}$ denote the input data matrix and $\mathbf{y} \in \mathbb{R}^n$ denote the response vector. Under the linear regression model,

$$\mathbf{y} = \mathbf{X}\mathbf{w} + \boldsymbol{\varepsilon}$$

Where $\mathbf{w} \in \mathbb{R}^p$ is the regression coefficient to be estimated and the noise $\boldsymbol{\varepsilon}$ is distributed as $N(0, \sigma^2\mathbf{I})$, To select the most predictive features, Lasso [10] provides a sparse estimate of $\mathbf{w}$ by solving the following optimization problem:

$$\hat{\mathbf{w}} = \text{argmin}_{\mathbf{w}} \frac{1}{2}\|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2 + \lambda\|\mathbf{w}\|_1$$

Where $\|\mathbf{w}\|_1 = \sum_{j=1}^{p} |w_j|$ is the l_1-norm of $\mathbf{w}$ which encourages the solutions to be sparse, and λ is the regularization parameter that controls the sparsity-level (the number of non-zero elements in $\hat{\mathbf{w}}$). For the sparsity-pattern of $\hat{\mathbf{w}}$, we could obtain a set of important features which correspond to those non-zero elements in $\hat{\mathbf{w}}$.

When features have a natural group structure, we could use the group Lasso penalty [8] to shrink each group of features to zero all-together instead of each individual feature. In particular, let $\mathcal{G}$ denote the set of groups and the corresponding group Lasso problem can be formulated as:

$$\hat{\mathbf{w}} = \text{argmin}_{\mathbf{w}} \frac{1}{2}\|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2 + \lambda \sum_{g \in G} \|\mathbf{w}_g\|_2,$$

Where $\mathbf{w}_g$ is the sub vector of $\mathbf{w}$ for features in group g; $\|\mathbf{w}_g\|_2 = \sqrt{\sum_{j \in g} \mathbf{w}_j^2}$ is the vector l_2-norm. This group Lasso penalty achieves the effect of jointly setting all of the coefficients within each group to zero or nonzero values.

III. Sparse Influence Linear Model (SLIM)

Utilizing the group Lasso penalty introduced in the previous section, we propose a new model, called SLIM (sparse linear influence model), which can automatically select the hub nodes without the prior knowledge of the network structure. We extend the LIM model by introducing another group lasso penalty on the user influence vector. In particular, we solve the following optimization problem:

$$\begin{array}{ll} \widehat{\mathbf{I}} = \text{argmin} & \frac{1}{2}\|\mathbf{V} - \mathbf{M} \cdot \mathbf{I}\|_2^2 + \lambda \sum_{u=1}^{N} \|\mathbf{I}_u\|_2 \\ \text{subject to} & \mathbf{I} \geq 0 \end{array} \quad (3)$$

Where each $\mathbf{I}_u = (I_u(1), \ldots, I_u(L))$ is the influence vector for the u-th node. We take the l_2-norm on each $\mathbf{I}_u$ so that it will encourage all the elements in $\mathbf{I}_u$ to go to zero together.

With the estimated $\widehat{\mathbf{I}}$, we can directly identify those most influential nodes. Let us denote the set of hub nodes by $\widehat{\mathcal{U}}$:

$$\widehat{\mathcal{U}} = \{u : \|\mathbf{I}_u\|_2 > 0\}.$$

The cardinality of $\widehat{\mathcal{U}}$ (i.e., the number of hub nodes) is controlled by the regularization parameter λ. When $\lambda \to \infty$, then all $\mathbf{I}_u$will become zero and none of the nodes will be selected as hub node (i.e., $\widehat{\mathcal{U}} = \emptyset$). On the other hand, if $\lambda \to 0$, $\widehat{\mathcal{U}} = \{1, \ldots, N\}$. Put another way, the smaller λ is, the more nodes will be selected as hub nodes. In practice, we could tune the regularization parameter to achieve the desirable number of hub nodes.

Although the formulation for SLIM is a convex optimization problem, the non-smoothness arising from $\|\mathbf{I}_u\|_2$ and the additional non-negativity constraint make the computation quite challenging. In the next section, we present an efficient and scalable solver for SLIM which could be applied to solve web-scale problems.

IV. Optimization for SLIM

In this section, we present an efficient optimization algorithm for solving SLIM in (3). We first introduce some necessary notations. Let $f(\mathbf{I}) = \frac{1}{2}\|\mathbf{V} - \mathbf{M} \cdot \mathbf{I}\|_2^2$ be the smooth part of the objective in (3). The gradient of $f(\mathbf{I})$ over $\mathbf{I}$ takes the following form:

$$\nabla f(\mathbf{I}) = \mathbf{M}^T(\mathbf{MI} - \mathbf{V})$$

In addition, $\nabla f(\mathbf{I})$ is Lipschitz continuous with the constants $L = \lambda_{\max}(\mathbf{M}^T\mathbf{M})$, which is the maximum eigenvalue of $\mathbf{M}^T\mathbf{M}$. In other word, we have for any $\mathbf{I}_1, \mathbf{I}_2$,

$$\|\nabla f(\mathbf{I}_1) - \nabla f(\mathbf{I}_2)\|_2 \leq L\|\mathbf{I}_1 - \mathbf{I}_2\|_2.$$

With these notations in place, we use the fast accelerated gradient method framework. In particular, we adopt the fast-iterative shrinkage-thresholding algorithm (FISTA) [9] as shown in the next Algorithm.

It is known that this algorithm has the optimal convergence rate of $O(1/t^2)$.

To apply this algorithm, the main difficulty is how to compute the first step. Firstly, we observe Step 1 can be written as:

$$\mathbf{I}^t = \mathrm{argmin}_{\mathbf{I}\geq 0} \quad \tfrac{1}{2}\|\mathbf{I} - \mathbf{W}\|_2^2 + \tfrac{\lambda}{L}\textstyle\sum_{u=1}^{N} \|\mathbf{I}_u\|_2 \quad (4)$$

where $\mathbf{W} = \mathbf{J}^t - \frac{1}{L}\nabla f(\mathbf{J}^t)$.

FISTA Algorithm for SLIM

```
Input: $\mathbf{V}$, $\mathbf{M}, L = \lambda_{\max}(\mathbf{M}^T\mathbf{M})$

Initialization: $\mathbf{J}_0 \in \mathbb{R}^n, t_0 = 1$

Iterate for $t = 1, 2, \ldots$, until convergence

    Step 1:
    $\mathbf{I}^t = \mathrm{argmin}_{\mathbf{I}\geq 0} \langle \mathbf{I}, \nabla f(\mathbf{J}^t)\rangle + \frac{L}{2}\|\mathbf{I} - \mathbf{J}^t\|_2^2 + \lambda\sum_{u=1}^{N}\|\mathbf{I}_u\|_2$

    Step 2: $\theta_{k+1} = \frac{1+\sqrt{1+4\theta_k^2}}{2}$

    Step 3: $\mathbf{J}^{t+1} = \mathbf{I}^t + \frac{\theta_k - 1}{\theta_{k+1}}(\mathbf{I}^t - \mathbf{I}^{t-1})$

Output: $\mathbf{I}^t$
```

Since (4) is separable in terms of $\mathbf{I}_u$, we solve it by each $\mathbf{I}_u$ independently, i.e.,

$$\mathbf{I}_u^t = \mathrm{argmin}_{\mathbf{I}_u \geq 0} \quad \frac{1}{2}\|\mathbf{I}_u - \mathbf{W}_u\|_2^2 + \frac{\lambda}{L}\|\mathbf{I}_u\|_2 \qquad (5)$$

The closed form solution of the above minimization problem can be characterized by the following Theorem.

Theorem 1. The optimal solution for the following optimization can be represented as following,

$$\mathrm{argmin}_{\mathbf{x}\geq 0} \quad \frac{1}{2}\|\mathbf{x} - \mathbf{w}\|_2 + \lambda\|\mathbf{x}\|_2 \qquad (6)$$

Let $\mathcal{P} = \{i : w_i > 0\}$ be indices for the positive values in $\mathbf{w}$ and let $\mathcal{P}^C$ be the complement of $\mathcal{P}$. Then the optimal $\mathbf{x}$ is,

$$\begin{cases} \mathbf{x}_{\mathcal{P}} = \max(1 - \frac{\lambda}{\|\mathbf{w}_{\mathcal{P}}\|_2}, 0)\mathbf{w}_{\mathcal{P}} \\ \mathbf{x}_{\mathcal{P}^c} = 0 \end{cases}$$

Proof. Firstly, we utilize the fact that the dual norm of l_2-norm is l_2-norm, so that we could present $\|\mathbf{x}\|_2$ as

$$\|\mathbf{x}\|_2 = \max_{\|\mathbf{y}\|_2 \leq \lambda} \mathbf{y}^T\mathbf{x}$$

Therefore, (6) can be reformulated as:

$$\min_{\mathbf{x}\geq 0} \max_{\|\mathbf{y}\|_2 \leq \lambda} \frac{1}{2}\|\mathbf{x} - \mathbf{w}\|_2 + \mathbf{y}^T\mathbf{x}$$

Interchange the min and max, we have,

$$\max_{\|\mathbf{y}\|_2\leq\lambda}\left(\min_{\mathbf{x}\geq 0}\frac{1}{2}\|\mathbf{x}-\mathbf{w}\|_2+\mathbf{y}^T\mathbf{x}\right) \tag{7}$$

Now we assume $\mathbf{y}$ is given, we first present the optimal $\mathbf{x}$ as a function of $\mathbf{y}$ using the KKT condition.

In particular, $\min_{\mathbf{x}\geq 0}\frac{1}{2}\|\mathbf{x}-\mathbf{w}\|_2+\mathbf{y}^T\mathbf{x}$ can be decomposed into each component of $\mathbf{x}$:

$$\min_{x_i\geq 0}\ \tfrac{1}{2}(x_i-(w_i-y_i))^2$$

To derive the optimal solution, we introduce the Lagrange multiplier $\mu\geq 0$ for the constraint $x_i\geq 0$. The Lagrange function takes the form

$$L(x_i,\mu)=\frac{1}{2}(x_i-(w_i-y_i))^2-\mu x_i$$

According to stationary condition, we have:

$$x_i=w_i-y_i+\mu$$

The complementary slackness condition implies that:

$$\mu x_i=\mu(w_i-y_i+\mu)=0$$

If $w_i>y_i$, since $\mu\geq 0$, $w_i-y_i+\mu>0$ and the complementary slackness implies that $\mu=0$ and hence $x_i=w_i-y_i$. On the other hand, if $w_i\leq y_i$, we have $\mu=y_i-w_i$ and hence $x_i=0$. In sum, we have:

$$x_i=\max(w_i-y_i,0) \tag{8}$$

Now we plug the above relation back into (7) and obtain that

$$\max_{\|\mathbf{y}\|_2\leq\lambda}\sum_{i=1}^{p}(-\frac{1}{2}y_i^2+y_iw_i)\mathbb{I}(w_i>y_i)+\frac{1}{2}w_i^2\mathbb{I}(w_i\leq y_i), \tag{9}$$

where $\mathbb{I}(\cdot)$ is the indicator function. Let

$$\phi(y_i)=(-\tfrac{1}{2}y_i^2+y_iw_i)\mathbb{I}(w_i>y_i)+\tfrac{1}{2}w_i^2\mathbb{I}(w_i\leq y_i)$$

We present its graphical illustration in Fig. 1.

To maximize (9) over $\mathbf{y}$ under the constraint $\|\mathbf{y}\|_2\leq\lambda$, we discuss the cases when $w_i\leq 0$ and $w_i>0$ separately:

- When $w_i\leq 0$, we could simply set $y_i=0$ so that $\phi(y_i)$ achieves its maximum value. Then according to (8), we have

$$x_i=0$$

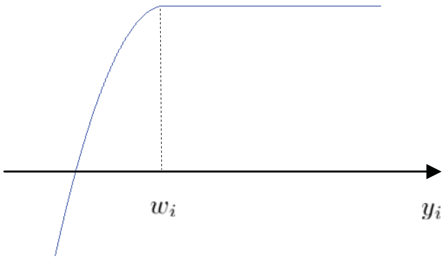


Figure 1. graphical illustration for the function $\phi(y_i)=(-\frac{1}{2}y_i^2+y_iw_i)\mathbb{I}(w_i>y_i)+\frac{1}{2}w_i^2\mathbb{I}(w_i\leq y_i)$

- When $w_i>0$, let $\mathcal{P}=\{i:w_i>0\}$ be the set of their indices. If $\|\mathbf{w}_\mathcal{P}\|_2\leq\lambda$, we could simply set $\mathbf{y}_\mathcal{P}=\mathbf{w}_\mathcal{P}$ so that $\sum_{i\in\mathcal{P}}\phi(y_i)$ achieves its maximum. According to (8), $x_\mathcal{P}=0$. On the other hand, if $\|\mathbf{w}_\mathcal{P}\|_2>\lambda$, we obtain the optimal $\mathbf{y}_\mathcal{P}$ by shrinking $\mathbf{w}_\mathcal{P}$ to the ball $\|\cdot\|_2\leq\lambda$, i.e.,

$$\mathbf{y}_\mathcal{P}=\tfrac{\lambda}{\|\mathbf{w}_\mathcal{P}\|_2}\mathbf{w}_\mathcal{P},$$

Then

$$x_\mathcal{P}=(1-\frac{\lambda}{\|\mathbf{w}_\mathcal{P}\|_2})\mathbf{w}_\mathcal{P}$$

By summarizing above two cases, we obtain the results in Theorem 1, which completes our proof. ■

V. Experiment

In this section, we evaluate the performance of our model on the twitter data. Twitter is an online social network used by millions of people around the world to stay connected to their friends, family members and co-workers through their computers and mobile phones. Users can post a tweet (status update message) to friends and colleagues. One can also follow other users; and her/his followers can read her/his tweets.

A. Dataset Collection

Twitter offers an API [11] to crawl and collect the data. We crawl all of tweets of specific 1000 twitter users between January 2009 and November 2011. These 1000 users are people or companies with professions related to IT and high-tech. We collected the full text, the author, the written time of each tweet. In addition, we collect the profile for each individual user, including the full name, followers count, the location, friends_count, a web page, a short biography, the account created time, the number of tweets. We crawl all the profiles of these 1000 users. We also identify 50 interesting topics from a set of most frequent words among these tweets. Typical examples of the selected topics include popular social media platforms (e.g., Twitter, LinkedIn, Yahoo, etc), products (e.g., iphone, blackberry, etc) and companies (e.g., Microsoft, Groupon). Based on the selected topics, we count the total times for each user who mentioned these topics, then rank all the users by their frequencies and select the top 200 active users out of the total 1000 users. We consider each active user as a node in an implicit network for information diffusion process.

B. *Experimental setup*

In the experiments, we use K = 50 topics, one day as the time unit, and set the time lag L = 10 (i.e., influence of a node decays to zero after 10 days). We allow each node to mention the topic multiple times during a time unit, i. e., $M_{u,k}(t)$ can be more than 1.

We construct the matrix including all of the 50 topics and apply our sparse linear influence model to select the global hot users. Also we use SLIM model on each individual topic to select the hot users respectively.

C. *Experimental results analysis*

1) *Hot twitter users for the total 50 topics:*

We detect global hot twitter users for all of interesting topics: we tune the regularization parameter in SLIM to select the top 35 hot users. After analyzing the selected users, we have two interesting findings:

a) The most influential users spread throughout the world:

We plot the locations of the selected 35 hot users as shown in Fig. 2. We can see that the most influential users spread all over the world, including Europe, Middle East, India, Indonesia, New Zealand, South America and North America. Most of the hot users are located in North America (16 out of 35), followed by the Europe (10 out of 35). There is no hot user in Africa, Australia and Antarctica. Since some countries in Asia like China doesn't allow people using twitter and some countries like Japan uses a different language twitter version, thus there are fewer selected hot users in Asia. Moreover, since the total 1000 twitter users are people or companies related to IT and high technology, Africa has no hot users due to its low level technology development. From this result, we can conclude that, except for some countries (e.g., China) where the twitter is blocked, North America and Europe are the world center for IT high technology development.

b) The followers count for twitter users:

In twitter, user can follow others, thus each user has the followers count. In the Fig. 3, we plot the number of followers for the total 200 active users in an increasing order with blue points. The red points represent the followers count for the selected 35 hot users. From this figure, we can see that most of hot users have a larger number of followers. This observation indicates that, globally, hot users should be the most influential users on the social network with more followers.

2) *Hot twitter users for the individual topics:*

Among the 50 interesting topics, there are several ones which can be grouped for analysis. For example, topics "iphone", "nokia", "samsung" and "blackberry" are brands of personal cell phone. Topics "twitter", "linkedin" and "yahoo" are social media platforms. Given a particular category of topics, we apply SLIM on each topic in this category to select the top 20 hot users. We compare the users' profiles for different topics within the same category and find some interesting patterns.

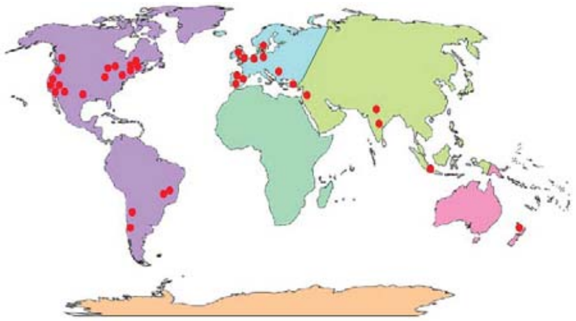


Figure 2. Location of selected hot users for total 50 topics

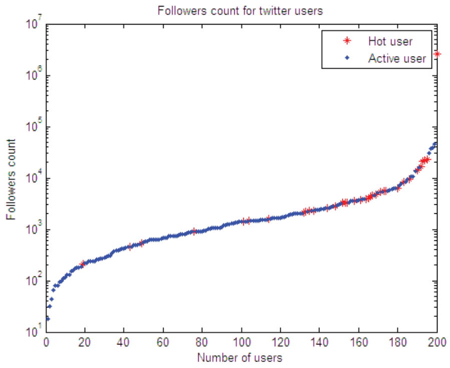


Figure 3. Followers count of the active users and the selected hot users for total 50 topics

a) Location distribution of the hot users for personal cell phone brand topices:

We have topics on different cell phone brands including "iphone", "nokia", "samsung" and "blackberry". For each brand, we analyze the selected hot users' location. The results are shown in Table I. We find out several interesting patterns:

TABLE I. LOCATION DISTRIBUTION OF HOT USERS FOR CELL PHONE BRAND TOPIC

	North America	**South America**	**Europe**	**Asia**	**South Africa**
iphone	13	2	3	1	1
nokia	6	1	11	2	0
samsung	4	0	11	5	0
blackberry	16	0	4	0	0

- "iphone" and "blackberry" have more hot users in North America than "nokia" and "samsung"; while "nokia" and "samsung" are more popular in Europe. One reason behind this phenomenon could be that “iphone” and “blackberry” are made in North American companies, Apple and RIM respectively. “nokia” is a Finland company, so more European people discuss about it.

- Hot users in North America for “iphone” topic are distributed uniformly. But most hot users for “blackberry” topic are located in California (7 hot users) and New York (4 hot users).
- Hot users in Europe for “nokia” are distributed uniformly. But most hot users for “samsung” topic are located in Germany (6 hot users) and India (4 hot users).

b) Friends count of the hot users for various social media platforms:

The topics related social media platforms include “twitter”, “linkedin” and “yahoo”. For each individual topic, we analyze these selected hot users’ friends count, which is the number of friends whom you are following. In Fig. 4, we draw the boxplot of the friends count for the selected 20 hot users in different topics. From this figure, we can see that hot users for “twitter” have more friends count than those for “linkedin” and “yahoo”. “linkedin” is the second and “yahoo” is the third. Twitter is a very large worldwide social network, so the hot users should have most friends. “linkedin” is used for professional networking, thus the hot users may have less business friends.

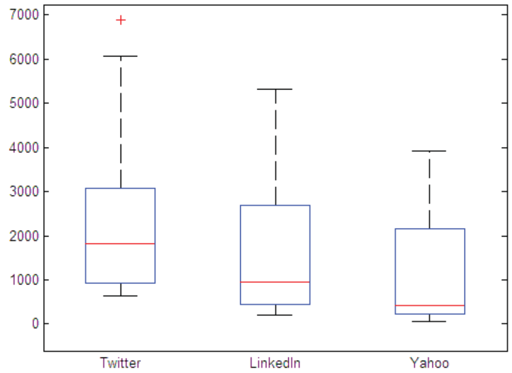


Figure 4. Boxplot of friends count of the selected hot users for social media platform topics

VI. Conclusions and Future works

In this paper, based on linear influence model, we develop an efficient hub nodes selection method SLIM for implicit information diffusion network. By utilizing the sparse learning framework, we introduce group Lasso penalty in the LIM model and formulate SLIM as a convex optimization problem. We adopt FISTA algorithm to solve this optimization problem by showing there is a closed-form solution for the key step in FISTA. The optimization algorithm achieves an optimal convergence rate $O(1/T^2)$. We conduct experiments on a set of 50 million tweets with 1000 twitter users. Our model can efficiently select the most influential users for particular topics. We also find several interesting patterns on the selected hot users for different categories of topics. Our model can be broadly applicable to general diffusion process on social platform, as they do not require knowledge of the underlying network topology. The benefit for selecting hot users is multifold. Firstly, targeting those influential users of specific product topics will increase the efficiency of the marketing strategy. For example, a smart phone manufacturer can engage those hot users to potentially influence more people. Secondly, we can utilize the influential users for interesting topics to improve the quality of opinions gathered. An interesting direction for future work is to extend our model on the dynamic social network by considering the evolution property of networks.

References

[1] M. Cha, H. Haddadi, F. Benevenuto, and K. P. Gummadi, “Measuring user influence in Twitter: The million follower fallacy”, The International AAAI Conference on Weblogs and Social Media, 2010.

[2] J. Leskovec, L. Backstrom, and J. Kleinberg, “Meme-tracking and the dynamics of the new cycle”, The 15th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2009.

[3] D. Liben-Nowell and J. Kleinberg, “Tracing information flow on a global scale using Internet chain-letter data”, PNAS, vol. 105, no. 12, pp 4633-4638, March 25, 2008.

[4] Ilyas, M.U and Radha, H, “Identifying Influential Nodes in Online Social Networks Using Principal Component Centrality”, IEEE International Conference on Communications (ICC), pp 1-5, 2011.

[5] Hao Ma , Haixuan Yang , Michael R. Lyu , Irwin King, “Mining Social Networks Using Heat Diffusion Processes for Marketing Candidates Selection”, Proceedings of the 17th ACM conference on Information and knowledge management (CIKM '08), New York, pp 233-242, 2008.

[6] Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue Moon, “What is Twitter, a social network or a news media?”, Proceedings of the 19th international conference on World wide web (WWW '10). New York, pp 591-600, 2010.

[7] Jaewon Yang and Jure Leskovec, “Modeling information diffusion in implicit network”, IEEE 10th International Conference on Data Mining (ICDM), pp 599 – 608, 2010.

[8] Ming Yuan and Lin Yi, “Model selection and estimation in regression with grouped variables”, Journal of the Royal Statistical Society, Series B, Volume 68, Part 1, pp. 49–67, 2006.

[9] Amir Beck and Marc Teboulle, “A Fast iterative shrinkage-thresholding algorithm for linear inverse problems”, SIAM Journal. on Imaging Science., Vol. 2, No. 1, pp. 183–202, 2009.

[10] Robert Tibshirani, “Regression Shrinkage and Selection Via the Lasso”, Journal of the Royal Statistical Society, Series B, Vol. 58, pp. 267-288, 1994.

[11] Twitter Search API. http://apiwiki.twitter.com/Twitter-API-Document.

[12] J. Weng, E.-p. Lim, J.Jiang, and Q. He, “ Twitterrank: finding topic-sensitive influential twitters”, Proc. of the third ACM international conference on web search and data mining, 2010.

[13] Xi Chen, Weike Pan, James Kwok, and Jaime G. Carbonell, “Accelerated Gradient Method for Multi-Task Sparse Learning Problem”, International Conference on Data Mining (ICDM), 2009.

[14] Masahiro Kimura, Kazumi Saito, Ryohei Nakano, and Hiroshi Motoda, “Extracting influential nodes on a social network for information diffusion”, Data Min. Knowl. Discov. Vol. 20, No. 1, pp. 70-97, 2010.

[15] David Kempe and Jon Kleinberg and Éva Tardos, “Influential Nodes in a Diffusion Model for Social Networks”, IN ICALP, pp. 1127-1138, 2005

Mining Call Graph for Change Impact Analysis[*]

Qiandong Zhang, Bixin Li, Xiaobing Sun
School of Computer Science and Engineering, Southeast University, Nanjing, China
State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences
{zhangqd, bx.li, sundomore }@seu.edu.cn

Abstract

Software change impact analysis (CIA) is a key technique to identify unpredicted and potential effects caused by software changes. Commonly used CIA techniques are to perform reachability on the graph representation of the system to compute the change effects. These CIA techniques often compute a large set of potentially impacted entities, which have many false-positives, and thus are difficult for practical use. In addition, these techniques do not consider the interference among the proposed changed entities, which in practice does exist. Faced with these problems, this paper proposed a new graph based mining CIA technique considering two factors: providing a stable state of the impact set to stop the reachability computation on the graph representation of the system and taking interference among multiple proposed changes into account to improve the precision of the impact results. Case study on the real-world program shows the effectiveness of our technique.

1 Introduction

Software needs to be maintained and changed over time to cope with new requirements, existing faults and change requests, etc. Changes made to software will inevitably have some unpredicted and undesirable effects on other parts of the software. When changes are made to software, they will have some unexpected and potential ripple effects, and may bring inconsistency to other parts of the original software. Software *change impact analysis* involves a collection of techniques to identify the potential effects caused by changes made to software [3]. It plays an important role in software development, maintenance, and regression testing [3].

CIA starts with a set of changed elements in a software system, called the *change set*, and attempts to determine a possibly larger set of elements, called the *impact set*, that requires attention or maintenance effort due to these changes [3]. CIA contains a collection of techniques for determining the effects on other parts of the software for proposed changes [13, 6]. Commonly used CIA techniques include these steps: analyzing the dependencies of the program, constructing an intermediate representation (e.g., call graph), and then conducting reachability analysis based on this representation [15]. The resultant impact set often has many false-positives, with many of its elements not really impacted [7, 12]. Thus this impact set they compute is very large and difficult for practical use [3]. In addition, most of current CIA techniques computes the impact set based on computing the union of the impact sets of each changed entity in the change set. This does not consider the interference among the changed entities in the change set. But in practice, some changes are implemented in combination to finish a change proposal. Hence, there does exist some relationship between these changed elements.

In this paper, our approach to tackle the two problems is similar to a phenomenon called ”water wave ripple”. Given the quiet water, several stones are suddenly thrown into the water, and initially, they all cause ripples on the water surface, respectively. These ripples will meet as they spread, and generate a new spreading center to propagate the ripples. Until some time, the spreading of the ripples will gradually become weaker and ultimately stop. Similar to this natural process, our CIA includes two steps: identifying the ”$center$” of multiple changes ripples and computing the changes effects of this $center$. As statistical results of the CIA techniques in the literature show that most of current CIA techniques compute the impact set at method level [9], our focus is also on method-level CIA. Call graph is widely used to represent call relation among methods and is a very common used representation to compute the method-level impact set [3, 7]. Traditional call graph based CIA tech-

[*] This work is supported partially by National Natural Science Foundation of China under Grant No. 60973149, partially by the Open Funds of State Key Laboratory of Computer Science of Chinese Academy of Sciences under Grant No. SYSKF1110, partially by Doctoral Fund of Ministry of Education of China under Grant No. 20100092110022, and partially by the Scientific Research Foundation of Graduate School of Southeast University under Grant No. YBJJ1102.

nique estimate the impact set based on transitive closure on the call graph [3]. This paper also used the call graph to represent the system and then computed the impact set on this representation. But different from traditional call graph based CIA technique, this paper attempts to compute a more precise impact set in respect to two factors: setting a stable state for the impact set computation on the call graph and considering interference among multiple changed entities in the change set. The case study on the real-world program has shown the effectiveness of our approach.

This paper is organized as follows: in the next section, we introduce our CIA approach. Section 3 gives the experimental evaluation to show the effectiveness of our CIA technique. In Section 4, some related work of *CIA* techniques are presented. Finally, we conclude our CIA technique and show some future work in Section 5.

2 Our Approach

Given the change set, CIA is employed to estimate the impact set of the changes made to the software. In this paper, CIA is performed based on mining the call graph.

2.1 Basics for CIA

As proposed above, our approach uses call graph to represent the system and CIA is performed relying on mining this call graph. In this section, some definitions and some operations in call graph are given. First, we introduce the definition of the call graph [8].

Definition 1 (Call Graph) *A call graph is a directed graph $G = (V, E)$. V is a set of vertices representing methods in the system, and $E \subseteq V \times V$ is a set of edges representing the call relationship between the caller and callee.*

In addition, there are two useful definitions to help mining the call graph: *inner-degree* and *inter-degree* of a subgraph $G^{'}$. They are defined as follows:

Definition 2 (Inner-degree) *Given a subgraph $G^{'}$ of the call graph, inner-degree of $G^{'}$ is the total number of the edges for each vertex in $G^{'}$.*

Definition 3 (Inter-degree) *Given a subgraph $G^{'}$ of the call graph G, inter-degree of $G^{'}$ on G is the total number of edges adjacent to (connected to) the vertices in $G - G^{'}$.*

As discussed in the above phenomenon ”water wave ripple”, there is an important spreading **center** to propagate the ripples of the water wave. Similar to this, there is also a *center* to propagate the change ripples during the process of change impact analysis. On the call graph, we call this $center$ as $core$, defined as follows:

Definition 4 (Core) *Core is a subgraph G' of the call graph G,, which satisfies that the ratio of their inner-degree over the inter-degree is greater than the threshold ε.*

The threshold ε indicates the cohesion of the relationship among the methods within the subgraph $G^{'}$. The bigger the ε is, the higher the cohesion of the relationship among these methods is. In our CIA approach, it is used to take the interference among the changed methods in the change set into account. The ε is at least bigger than 1, which shows the relationship among the methods within the subgraph is stronger than connecting to its outer methods.

In graph theory, *breadth-first-search (BFS)* is widely used in graph search algorithm, defined as follows:

Definition 5 (BFS) *BFS begins from the origin node and explores all the neighboring nodes. Then for each of those nearest nodes, it searches their unexplored neighbor nodes, and so on, until it finds the goal.*

This search algorithm is also used for change impact analysis for searching the ripples of the changes. In the next section, our CIA approach is presented based on these definitions.

2.2 Change Impact Analysis

In our approach, change set is proposed to be composed of a set of changed methods, CIA is then performed relying on mining the call graph, and generates a set of potentially impacted methods. It includes two steps. First, we identify the ”$center$” of multiple changes ripples, which we call *initial impact set*. It is expected that the methods in the initial impact set are less likely to be false-positives. Then, we compute the changes effects of this $center$, and obtain the *final impact set*. And this set is expected to cover all the ripples affected by the changes.

As discussed above, most of current CIA techniques compute the impact set from the individual entities in the change set, without considering the relationship existed among them. In practice, multiple changes are performed in combination to finish a change proposal, hence there exists the relationship among the changed entities in the source code. Such a relationship is called the *interference* in this paper. We take the interference among the entities in the change set into account to estimate the impact set. First we attempt to identify the center of these changes, which includes the most probably impacted entities potentially affected by these multiple changes. These entities are collected into the initial impact set, which is defined as follows:

Definition 6 (Initial Impact Set (IIS)) *Initial impact set is computed using the BFS algorithm from the methods* *in the change set. The BFS in call graph stops searching when the methods in the initial impact set can satisfy the $core$ condition as defined in Definition 4.*

From this definition, we know that the methods in the IIS are generated from the change set, and satisfy the *core* definition as defined above. The methods in the IIS are the methods most likely impacted by these multiple changes, that is, the IIS is expected to have few false-positives. However, it can not meet practical use. On the one hand, some methods in the IIS are still the noise methods (false-positives). These methods are expected to be removed. On the other hand, there are many other methods that are really impacted but not included in the IIS, i.e., false-negatives. These methods should be included in the impact set to guarantee the consistency of the modification. Therefore, similar to the propagation of the ripples of the *water wave ripple* phenomenon, we should expand the IIS to perform the propagation analysis to get a final impact set (FIS), which is hoped to cover more false-negatives and remove some false-positives from the IIS. The definition of FIS is as follows:

Algorithm 1 FISComputation

```
Input:
    $P$: Original program
    $IIS$: Initial impact set
Declare:
    $m$: A method
    $AS$: A set of adjacent methods of the method $m$ in call graph
    $\theta$: A threshold
Use:
    $Adjacent(m)$: it returns a set of adjacent methods of the method $m$ in call
    graph.
Output:
    Final impact set $FIS$
 1: $FIS = IIS$
 2: while $FIS$ is not stable do
 3:     for each $m$ in call graph do
 4:         $AS = Adjacent(m)$
 5:         $S = AS \cap FIS$
 6:         if $m \notin FIS \wedge |S|/|AS| \geqslant \theta$ then
 7:             $FIS = FIS \cup \{m\}$
 8:         end if
 9:         if $m \in FIS \wedge |S|/|AS| < \theta$ then
10:             $FIS = FIS - \{m\}$
11:         end if
12:     end for
13: end while
14: return $FIS$
```

Definition 7 (Final Impact Set (FIS)) *Final impact set is computed based on the propagation analysis from the IIS, and gets to a termination when the FIS is stable, i.e., no update occurs in the FIS any more.*

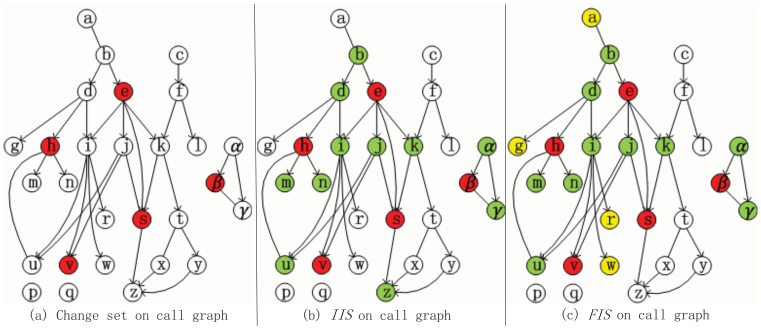
The approach to compute the FIS is shown in Algorithm 1. In the algorithm, there is an important threshold θ indicating the possibility of the methods potentially affected by the IIS. It is used to judge whether a method should be included in the FIS. The value of this threshold relies on the quality of the IIS. In other words, the fewer the false-positives in the IIS is, the bigger the value of the threshold θ is. The range of this threshold is between 0 and 1. As shown in Line 1 in Algorithm 1, the FIS is computed from the IIS. Then computation of FIS includes both finding the false-negatives and removing the false-positives of the IIS. In the algorithm, Lines 6-8 are used to add new methods (false-negatives of IIS) into the FIS; Lines 9-11 are used to remove the methods that are probably false-positives from IIS. Ultimately, there is no update in FIS and the FIS becomes stable. So far, we have completed the whole CIA process. The output of the CIA is composed of the IIS and FIS. In the next section, we give an example to show the CIA process.

2.3 An Example

In this section, we give an example to show the CIA process. We assume that the call graph has been constructed as shown in Figure 1. Each node on the call graph represents the *method* in the program. In Figure 1(a), the *red* nodes represent the changed methods in the change set. CIA is then performed to estimate the ripple effects of the change set. First, we identify the *center* of the change set, which is also the initial impact set. In this step, the value of the threshold ε is set to be 2. According to Definition 6, we use the BFS algorithm to search the graph to find the IIS until a *core* is generated. Thus, the IIS is generated as the *green* nodes as shown in Figure 1(b). Then, we need to use Algorithm 1 to compute the final impact set from this IIS. The threshold θ value in this step is set to be 0.4. FIS is computed from the IIS, and includes both finding the false-negatives and removing the false-positives of the IIS. For example, for the node a, its θ value is 1, which is bigger than 0.4, then we should include this method in the FIS. For the node z, its θ value is 0.3, which is smaller than 0.4, then we will remove it from the IIS. Based on the similar mechanism, other methods are added and removed. Until there is not update in the FIS, FIS is obtained as the ultimate result. If we use traditional call graph based CIA technique [3] to compute the impact set, all the methods in Figure 1 except p and q are collected in the impact set, which may have more false-positives.

3 Case Study

In this section, we present our case study to evaluate the effectiveness of our CIA approach.



(a) Change set on call graph (b) *IIS* on call graph (c) *FIS* on call graph

Figure 1. An example to illustrate the process of CIA

3.1 Setup

We use the subject *NanoXML*[1] as our research object. *NanoXML* is a small XML parser for Java. We extracted four consecutive versions of *NanoXML* from its CVS repository for evaluation. There are about 26 classes, 245 methods, and 7631 lines of code for *NanoXML*. During the evolution of these versions, there is a basic version ($V0$) from which to start.

The focus of our evaluation is on the accuracy of the CIA technique. The accuracy of the CIA is in terms of *precision* (P) and *recall* (R). *Precision* is an inverse measure of false-positives while *recall* is an inverse measure of false-negatives. They are widely used in an information retrieval scenario [2], defined as follows:

$$P = \frac{|Actual \quad Impact \quad Set \quad \cap \quad Estimated \quad Impact \quad Set|}{|Estimated \quad Impact \quad Set|}$$

$$R = \frac{|Actual \quad Impact \quad Set \quad \cap \quad Estimated \quad Impact \quad Set|}{|Actual \quad Impact \quad Set|}$$

Here, *Actual Impact Set* is the set of methods which are really changed during versions evolution. This is obtained by comparing the changed methods between consecutive program versions. *Estimated Impact Set* is the set of methods estimated based on the *change set* using the *CIA* technique. As we can not obtain the information that which changed methods are pre-actively changed to cope with the change request and which changed methods are reactively changed to cope with the impacts induced by these changes, a statistical testing of our CIA technique with a variation of the percentage of the number of changed methods ranging from 10% to 50% of the total number of change methods is performed for each program transaction. In the study, the average number of the methods in the *change set* for each program transaction (i.e., $V_i - > V_{i+1}$) is shown in the first column in Table 1.

3.2 Results

During the process of the evaluation, we should first find appropriate values for the threshold ϵ and θ. For space limitation, we only provide some results with high quality here. More details of the preliminary evaluation results are available online[2]. Based on the empirical results, the precision and recall of both IIS and FIS are reasonable and better with the range of $\epsilon = 2, 3, 4, 5, \theta = 0.2, 0.3, 0.4, 0.5$. So the experimental results only show the precision and recall of these threshold ranges, as shown in Table 1. In addition, for comparison, we also present the precision and recall of the impact set computed based on traditional call graph based CIA technique (TCG) [3]. In the following, we discuss the data collected from the case study.

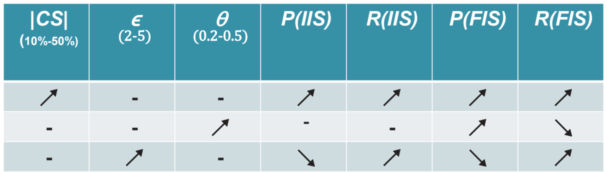
First, we see whether different threshold values have impact on the accuracy of our CIA. In Table 1, the third column shows different ϵ values. The threshold ϵ indicates the cohesion of the relationship among the methods and is used to control the precision of the initial impact set. From this table, we see that in most cases, with the increase of the ϵ values, the recall of both the IIS and FIS is also increased, but their precision values are decreased. From these results, we see that the FIS relies on the IIS, i.e., if the IIS is better, the FIS is better. Thus, the ϵ threshold has impact on both the IIS and FIS generated by our CIA technique. Then, we see the impact of the θ threshold on our CIA technique. This threshold relies on the quality of the IIS, and is used to indicate the possibility of the methods potentially affected by the IIS. As Table 1 shows, with the increase of the θ values, the precision of the FIS is also increased, but the recall is decreased. For example, for transaction $V0- > V1$, in case of $\epsilon = 3$, when $\theta = 0.2$, the precision and recall of FIS is 0.51 and 0.84, respectively. When $\theta = 0.3$, its precision is increased to 0.57, but its recall is decreased to 0.79. With the increase of θ values, the same phenomenon occurs. So the effectiveness of the CIA technique is also affected by the θ threshold. In addition, we see an interesting phenomenon in Table 1, i.e., for different transactions of the versions evolution, the precision and recall of the impact sets with the same ϵ and θ threshold value are different. For example, the precision and recall for

[1] http://nanoxml.sourceforge.net/orig

[2] http://ise.seu.edu.cn/people/XiaobingSun/seke2012.xls

Table 1. The precision and recall results of the empirical study

CS	θ	ϵ	P_{IIS}	$R_{IIS)}$	P_{FIS}	R_{FIS}	P_{TCG}	R_{TCG}
Transaction: $V0- > V1$								
20	0.2	2	0.72	0.55	0.51	0.83	0.47	0.93
		3	0.72	0.56	0.51	0.84	0.47	0.93
		4	0.69	0.61	0.52	0.84	0.47	0.93
		5	0.66	0.63	0.52	0.84	0.47	0.93
	0.3	2	0.75	0.51	0.59	0.79	0.47	0.93
		3	0.71	0.57	0.57	0.79	0.47	0.93
		4	0.71	0.57	0.59	0.77	0.47	0.93
		5	0.67	0.64	0.55	0.82	0.47	0.93
	0.4	2	0.79	0.51	0.72	0.65	0.47	0.93
		3	0.70	0.59	0.63	0.72	0.47	0.93
		4	0.71	0.59	0.65	0.69	0.47	0.94
		5	0.66	0.66	0.58	0.76	0.47	0.93
	0.5	2	0.75	0.49	0.67	0.56	0.47	0.93
		3	0.75	0.55	0.68	0.62	0.47	0.93
		4	0.71	0.60	0.65	0.68	0.47	0.93
		5	0.65	0.65	0.58	0.73	0.47	0.93
Transaction: $V1- > V2$								
13	0.2	2	0.42	0.59	0.31	0.75	0.25	0.85
		3	0.39	0.65	0.29	0.79	0.25	0.85
		4	0.40	0.64	0.29	0.79	0.25	0.85
		5	0.37	0.73	0.31	0.81	0.25	0.85
	0.3	2	0.45	0.58	0.35	0.69	0.25	0.84
		3	0.38	0.59	0.31	0.72	0.25	0.84
		4	0.37	0.59	0.31	0.73	0.25	0.85
		5	0.36	0.66	0.30	0.75	0.25	0.84
	0.4	2	0.47	0.56	0.41	0.63	0.25	0.84
		3	0.39	0.65	0.33	0.72	0.25	0.84
		4	0.38	0.64	0.33	0.73	0.25	0.86
		5	0.36	0.64	0.33	0.72	0.25	0.84
	0.5	2	0.42	0.61	0.40	0.61	0.25	0.85
		3	0.39	0.62	0.34	0.64	0.25	0.84
		4	0.38	0.63	0.33	0.65	0.25	0.85
		5	0.37	0.66	0.33	0.69	0.25	0.84
Transaction: $V2- > V3$								
12	0.2	2	0.85	0.67	0.25	0.93	0.17	0.93
		3	0.80	0.72	0.25	0.94	0.17	0.94
		4	0.81	0.77	0.25	0.93	0.17	0.93
		5	0.52	0.84	0.25	0.94	0.17	0.94
	0.3	2	0.82	0.64	0.57	0.91	0.17	0.94
		3	0.84	0.71	0.58	0.90	0.17	0.93
		4	0.83	0.78	0.56	0.92	0.17	0.94
		5	0.48	0.84	0.37	0.92	0.17	0.93
	0.4	2	0.85	0.66	0.78	0.83	0.17	0.93
		3	0.83	0.72	0.76	0.84	0.17	0.94
		4	0.81	0.76	0.73	0.84	0.17	0.93
		5	0.47	0.82	0.44	0.88	0.17	0.93
	0.5	2	0.87	0.66	0.84	0.74	0.17	0.93
		3	0.79	0.72	0.76	0.79	0.17	0.93
		4	0.80	0.77	0.77	0.83	0.17	0.93
		5	0.51	0.83	0.50	0.88	0.17	0.93



|CS| (10%-50%)	ϵ (2-5)	θ (0.2-0.5)	P(IIS)	R(IIS)	P(FIS)	R(FIS)
↗	-	-	↗	↗	↗	↗
-	-	↗	-	-	↗	↘
-	↗	-	↘	↗	↘	↗

Figure 2. Impact of CS, θ and ϵ on the CIA technique

the transaction $V1- > V2$ are bad but they are very good for transaction $V2- > V3$. As we trace the changes to the transactions of these two version evolution, we know that, for $V1- > V2$ transaction, the changes made to the system are loosely scattered in different parts, in other words, these changes are implemented individually with very few relationships among them. But for $V2- > V3$ transaction, the changes are implemented in only some local parts of the system and some of them are implemented together. This phenomenon shows that interference among the changes affects the effectiveness of the CIA. Hence, from the discussion above, we see that selection of different CS, θ and ϵ have impact on the CIA technique. The tendency of their impacts are summarized in Figure 2. From this figure, we know that when we know more information about the changes, our CIA is better. And different values of these two thresholds can control the precision and recall of the CIA technique.

Then, we see the effectiveness of our CIA technique. Our CIA computes two impact sets (IIS and FIS). Table 1 shows that the precision and recall of the IIS and FIS are different, to say in detail, in most of the time, the precision of the IIS is better than that of the FIS while the recall of the IIS is worse than that of FIS. The results conform to our expectation and are useful in practice. In practice, we can first use the IIS to more precisely check the real change ripples. After this, we use the FIS to check other change ripples to ensure the integrity of the change. In addition, Table 1 also presents the precision and recall of the traditional call graph based CIA technique (TCG) in the last two columns. As it shows, the precision of our technique is much higher than that of the TCG. But the recall is a little worse than the TCG. This indicates that our CIA technique effectively removes some false-positives from the TCG technique at the cost of a little recall.

From the data and analysis above, we obtain some preliminary conclusions: (1) the precision and recall of the impact sets are affected by the ϵ and θ; (2) our CIA technique is particularly useful to tackle multiple changes which have some relationships (interference) among them. (3) our CIA computes the IIS with higher precision and the FIS with higher recall. Compared with traditional call graph based CIA technique, our technique can effectively remove some false-positives, but at the cost of a little recall.

3.3 Threats to Validity

Like any empirical validation, ours has its limitations. Firstly, we have considered the application of our CIA approach to only one Java program. Therefore, to allow for generalization of our results, large-scale experimental evaluation on different projects in different object oriented languages is necessary to address such threats to external validity. However, the program used here is real and non-trivial programs, moreover, the subject program is selected from open projects and widely used for empirical validation [11]. In addition, the change sets are obtained by randomly selecting some differences between consecutive versions. Since there exists random variation, we may not obtain the same results if we repeat our case study. However, we performed a statistical testing of our CIA technique with a variation of the percentage of the number of changed methods in the change set ranging from 10% to 50%. Using this strategy, the results should support fairly reliable conclusion.

4 Related Work

Current researches in CIA have varied from relying on static information [13, 15, 6, 10] to dynamic information [7, 1] in working out the impact set. Our technique mainly focuses on static analysis of the program. Static CIA techniques take all possible behaviors and inputs into account, and they are often performed by analyzing the syntax and semantic dependence of the program [10]. The static analysis includes textual analysis, historical analysis and structural static analysis. Textual analysis is an approach which extracts some conceptual dependence (conceptual coupling) based on the analysis of the non-source code (comments). These coupling measures provide a new perspective to traditional structural coupling measures. Concept coupling is based on measuring the degree to which the identifiers and comments from each other [13, 5]. Historical analysis is performed by mining the information from multiple evolutionary versions in software historical repositories [16]. With this technique, some *evolutionary* dependencies between program entities that can not be distilled by traditional program analysis technique can be mined from these repositories. Evolutionary dependencies show that which entities are (historically) changed together in software repositories, these entities may need to change when one (some) of the entities are changed during future software evolution. CIA is then supported based on these evolutionary dependencies. Structural static dependencies between program entities are very crucial to CIA, i.e., if a program entity changes, other dependent entities might also have to change [15, 10]. This paper is proposed based on the structural static analysis to compute the impact set.

5 Conclusion and Future Work

This paper proposed a novel and effective CIA technique which is based on call graph mining. Compared with traditional call graph based CIA technique, our CIA approach is particularly suitable for tackling multiple changes which have interference among them, and computes a more precise impact set with relative small-size set. Though we have shown the effectiveness of our technique through a real empirical study, it can not indicate its generality for other real environment. And we will conduct experiments on other arbitrary programs to evaluate the generality of our technique. In addition, we would like to prioritize the entities in the impact set to better facilitate its practical use.

References

[1] T. Apiwattanapong, A. Orso, and M. J. Harrold. Efficient and precise dynamic impact analysis using execute after sequences. In *Proceedings of the International Conference on Software Engineering*, pages 432 – 441, 2005.

[2] R. A. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[3] S. Bohner and R. Arnold. *Software Change Impact Analysis*. IEEE Computer Society Press, Los Alamitos, CA,USA, 1996.

[4] M. Gethers and D. Poshyvanyk. Using relational topic models to capture coupling among classes in object-oriented software systems. In *Proceedings of the 2010 IEEE International Conference on Software Maintenance*, pages 1–10, 2010.

[5] H. Kagdi, M. Gethers, D. Poshyvanyk, and M.Collard. Blending conceptual and evolutionary couplings to support change impact analysis in source code. In *Proceedings of the IEEE Working Conference on Reverse Engineering*, pages 119–128, 2010.

[6] J. Law and G. Rothernel. Whole program path-based dynamic impact analysis. In *Proceedings of the International Conference on Software Engineering*, pages 308 – 318, 2003.

[7] O. V. Lhotak. Comparing call graphs. In *Workshop on Program Analysis for Software Tools and Engineering*, pages 37 – 42, 2007.

[8] B. Li, X. Sun, H. Leung, and S. Zhang. A survey of code-based change impact analysis techniques. *Journal of Software Testing, Verification and Reliability, doi: 10.1002/stvr.1475*, 2012.

[9] A. Mithun and R. Brian. Practical change impact analysis based on static program slicing for industrial software systems. In *Proceedings of the 33rd International Conference on Software Engineering*, pages 746–755, 2011.

[10] A. Orso, T. Apiwattanapong, J. Law, G. Rothermel, and M. J. Harrold. An empirical comparison of dynamic impact analysis algorithms. In *Proceedings of the International Conference on Software Engineering*, pages 491 – 500, 2004.

[11] A. Orso and M. J. Harrold. Leveraging field data for impact analysis and regression testing. In *Proceedings of the ACM SIGSOFT Symposium on Foundations of Software Engineering*, pages 128–137, 2003.

[12] D. Poshyvanyk, A. Marcus, R. Ferenc, and T. Gyimothy. Using information retrieval based coupling measures for impact analysis. *Empirical Software Engineering*, 14(1):5 – 32, 2009.

[13] X. Sun, B. Li, S. Zhang, C. Tao, X. Chen, and W. Wen. Using lattice of class and method dependence for change impact analysis of object oriented programs. In *Proceedings of the Symposium on Applied Computing*, pages 1444–1449, 2011.

[14] T. Zimmermann, A. Zeller, P. Weissgerber, and S. Diehl. Mining version histories to guide software changes. *IEEE Transactions on Software Engineering*, 31(6):429 – 445, 2005.

A mobile application for stock market prediction using sentiment analysis

Kushal Jangid*
San Jose State University
Computer Engineering Department
San Jose, CA, USA

Pratik Paul†
San Jose State University
Computer Engineering Department
San Jose, CA, USA

Magdalini Eirinaki
San Jose State University
Computer Engineering Department
San Jose, CA, USA

Abstract

Lately, stock markets have been going through a lot of volatility. Traditional methods of stock market prediction, which involved using historical stock prices to predict future price have shown to be insufficient under certain circumstances. In this paper we present a mobile application that employs a different approach to predicting the stock price, namely the news related to that company. The prediction algorithm determines whether the overall sentiment related to the company, as expressed by the news stories, is good or bad and assigns a sentiment score. The system then uses machine learning to predict the percentage fluctuation of the company's stock based on this score. The algorithm is integrated in a mobile application that helps users try out various market strategies based on our prediction engine. This is achieved by performing simulated trades using virtual money. It also provides real-time stock quotes, and the latest financial news. In this paper we present the overall system architecture and design of this application, as well as details of the prediction process.

1 Introduction

There is a tremendous research going on in the field of stock market prediction. There are a number of artificial intelligence and machine learning techniques that have been used for analyzing price patterns and predicting stock prices and index. Among the most commonly used are Neural Networks, which have the ability to learn non-linear relationships based on trading information. This allows modeling of non-linear dynamic systems such as stock markets more precisely. Support Vector Machines (SVM) seem to be the next most popular approach to stock market prediction. It has been successful in classification task and regression tasks, especially on time series prediction and financial-related applications.

There has also been some research on hybrid algorithms which combine different algorithms to improve the prediction accuracy. One example is the combination of Genetic Algorithm and Support Vector Machine. The experiments conducted indicate that the accuracy achieved by combining the algorithms is better than the results from algorithms individually [3]. The authors [5] claim that Support Vector Machine when combined with Boosting provides better accuracy.

A more recent trend, however, is to depart from using pure machine learning on the numbers, and instead to also focus on other input such as the financial news to determine the stock fluctuation. Most of the people are familiar that company's earnings report and good/bad news associated with the company tend to affect its price. Looking at its importance, people have started combining text mining with a couple of different algorithms in order to predict the stock price movement. One example is where some researchers combined text mining approach with time series algorithm to map the fluctuation in the stock price [10]. There are other examples where the text mining algorithm is combined with SVM [7].

Most of the applications for stock market prediction are web-based, and there are not many available for smart phones. However, as smart phones become an integral part of everyday lives and people are using them as portable devices, the need to trade on the go is evident. After interviewing a couple of people working on Wall Street, we found out that having a prediction system available on your phone

*Author's current affiliation is E*Trade Financial Corporation

†Author's current affiliation is Reply.com

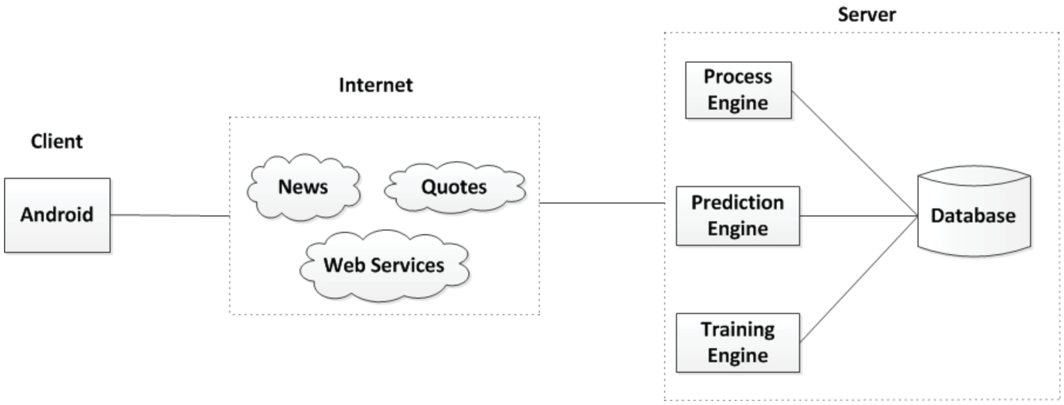


Figure 1. System Architecture

would be really beneficial to the traders. This will assist them in making the decision of buying or selling a companys stock. This motivated our work, designing a mobile client which enables the user to create a watch list of stocks, get latest stock quotes, and request stock predictions. The application also enables the user to simulate stock trading based on the predictions using virtual money but real stock data.

In a nutshell, our prediction system works as follows: the user sends the companys ticker symbol whose prediction is needed. The prediction system scans that particular companys news and generates a sentiment score which tells whether the sentiment related to that company is good/bad and also how much percentage fluctuation in price might occur within three hours. The prediction system is able to give the percentage fluctuation because it has been trained to determine the relationship between the sentiment score and the percentage in stock price change.

The rest of the paper is organized as follows: In Section 2, we provide a review of the related work. We then discuss in detail the architecture of the mobile application in Section 3. In Section 4 we outline the basic steps of the prediction algorithm and in Section 5 we present the system prototype. We conclude with our plans for future work in Section 6.

2 Related Work

One line of research is to use Support Vector Machines (SVM) for stock market prediction. In [3] a hybrid machine learning system based on Genetic Algorithm (GA) and Support Vector Machine (SVM) is proposed. For input, the authors use the correlation among stock prices of various companies as well as various pointers from the area of technical analysis. Their observation was that, the fusion of Genetic Algorithm/SVM is better than the individual SVM System. In [9] the author claims that SVM combined with boosting gives superior results in stock market prediction. It also suggests that a particular algorithm might be better suited to a particular type of stock, say technology stocks, whereas give lower accuracies while predicting other types of stocks, say energy stocks. This paper also states that external unknown factors like election results, rumors, and other factors should be considered for stock market prediction. In another paper [5] the authors state that non-linear combination of linear and non-linear methods give better predictions than using each method separately. This paper combines linear and non-linear regression models with SVM and use their model to predict opening and closing price on the Shanghai Stock Exchange.

Another line of research focuses on the use of Neural Networks (NNs). In [11] various prediction models are compared and the authors conclude that NNs predict market directions more accurately than other existing techniques. In [8] the authors describe how NNs learn with time and errors when performing three predictions.They check the results for low errors and high accuracy, and suggest improving the preprocessing or enlarging the training size and trying again in case of low accuracy and errors.

More recently, the research proposes the use of text mining on financial news in order to predict the stock market [1, 7, 6] . They state that news contents are one of the essential aspects that control the market. The authors in [1] discuss various techniques that are applied to study the consequence of financial news on prediction of stock market. They claim that if the classifier input has both news and stock prices at the same time, it results in more precise results. The work of [7] looks at the role of financial news articles on three textual representations; bag of words, noun phrases, and named entities and their capacity to predict discrete number stock prices twenty minutes after an article release. Using an SVM derivative, the writers prove that their model had a major impact on predicting future stock prices compared to linear regression. In [6], the authors propose a text mining-based financial news analysis system to identify the major news events that affect the stock market and also understand their impact. Most recently, in [2] the authors analyze the text content of Twitter feeds using two mood tracking tools, and use a Granger causality analysis and a Self-Organizing fuzzy Neural Network to predict changes in the Dow Jones Industrial Average closing values.

Apart from the research papers, there exist two Android applications which are related to the proposed system. One of them is Bloomberg Mobile[1]. It is one of the best available in the market supporting features like latest financial news and stock market quotes among others. The unique selling point of the application is that users will be able to create personalized views of the news filtered by industry, region, exclusivity and/or popularity. Another application[2] which is related to our project is about simulation of buying and selling of stocks. It allows the users to track their progress, buy and sell at real, up to date stock prices and manage their portfolio on the go. The unique selling point of the application is the presentation of the stock market in a simulated environment. Our application is unique in that it combines the best features of other applications, such as providing latest financial news, quotes, and a simulated environment for trading using virtual money, but also adds new features such as stock market prediction.

3 System Architecture

A high-level architecture of the proposed system is depicted in Figure 1. This is a three-tier architecture, consisting of the Android client, the Internet, providing access to stock-related news and quotes and enabling the client/server interaction through the Web Services, and the Server, consisting of the Process, the Prediction, and the Training Engines and the system's database.

3.1 Android Client

The Android client provides several types of functionality to the end user: It serves as a stock market interface, providing information about the latest financial news and quotes (using the Internet module). It also allows the end user to create a watch list of stocks and provides a simulated environment for buying/selling stocks using virtual money. This can be regarded as a stock market online game, and the end user can review his/her ranking as compared to other players who are using the application. Most importantly, the Android client interacts with the server's prediction engine to provide predictions for a company's stock price. This information can be used by the end user to add/remove stocks from his/her watch list and portfolio. All this information is sent to the Android Client by the Server through the Web Services.

3.2 Server

The Server consists of three Engines (Process, Training, Prediction), as well as the database, with the Training and Prediction modules being the core of the proposed system.

Training Engine. The Training Engine uses the Internet module to retrieve news and quotes. It then employs data mining algorithms to train the system so that it can then predict the stock price movement. It saves those rules in the database, which can then be used by the Prediction Engine to provide prediction to the Android Client. The training consists of a series of actions. Since the data are time-sensitive, the system needs to be trained frequently so that the predictions depict the latest market trends. Our architecture enables the administrator to decide how often the training occurs. The training process is triggered automatically at the times defined.

The training process can be described as the following sequence of actions, also depicted in Figure 2: The Training Engine requests news for a list of companies through Yahoo! Finance[3]The Training Engine then processes the articles using sentiment analysis, and assigns a sentiment score to each company using the sentiment dictionary. A positive sentiment score means that the stock price is predicted to go up, whereas a negative sentiment score means that it is predicted to go down. The sentiment scores are saved in the database. Next, the stock quotes of those companies are requested and are also saved in the database. The quotes of the same companies are requested after n hours (n is a user-defined parameter). The quotes' source returns real-time stock quotes for those companies. The training engine retrieves from the database the sentiment scores which were generated for those companies as well as the quotes to check for accurate predictions made in the stock price movement. The training engine then uses the sentiment scores of the accurate predictions, and creates correlations of the scores to the percentage change in the respective stock quotes using machine learning techniques (association rules or decision trees). The results, in the form of rules, are saved in the database, to be accessed later by the prediction engine to make predictions on both the trends (stock going up or down) and the percentage change of the stock.

Prediction Engine. The Prediction Engine is used to send stock price predictions to the Android Client. This engine interacts with the database to retrieve and use the training data. However, its functionality is asynchronous to that of the Training engine. In fact, the Prediction Engine is called on-demand, i.e. whenever a user wants a prediction on a

[1] http://www.bloomberg.com/mobile

[2] http://www.chickenbrickstudios/games/mmc

[3] http://finance.yahoo.com

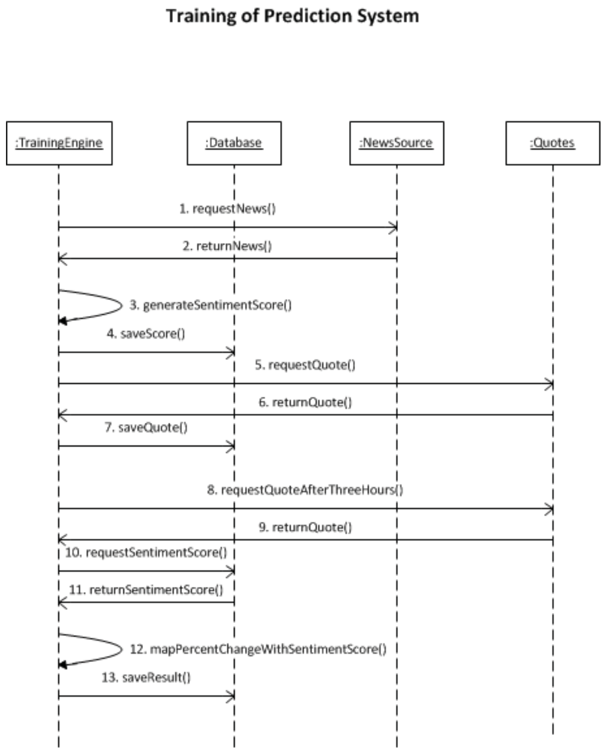

Training of Prediction System
:TrainingEngine
:Database
:NewsSource
:Quotes
1. requestNews()
2. returnNews()
3. generateSentimentScore()
4. saveScore()
5. requestQuote()
6. returnQuote()
7. saveQuote()
8. requestQuoteAfterThreeHours()
9. returnQuote()
10. requestSentimentScore()
11. returnSentimentScore()
12. mapPercentChangeWithSentimentScore()
13. saveResult()

Figure 2. Sequence Diagram of Training Process

specific stock. The sequence of actions of the Prediction Engine are shown in Figure 3. Whenever a user requests for prediction, the Android client sends the request to the Prediciton Engine. The Prediction Engine then gathers the news of that company and generates the sentiment score real-time. It then accesses the database to get the trained rules. Finally, using these rules, the engine predicts how much percentage change in the stock price will occur for that sentiment score.

Process Engine. The Process Engine verifies the user login, and generates a session. It consists of functions to get the users trading information from the database, and provide it to the Android Client. It processes the buying/selling of stocks using virtual money. Finally, it ranks the users based on the profits made in the simulated trading system.

Database. The users login and trading information is saved in the server database. Apart from that, the Training Engine saves the trained values in the database, which will later be used by the Prediction Engine to predict percentage change in a companys stock price.

4 Data Mining Algorithms

We now discuss in more detail the two main components of the prediction process, namely the text mining and sentiment analysis of the news, used to predict the stocks' trends (stock going up/down), and the association rules mining on them, used to predict the percentage of stock change.

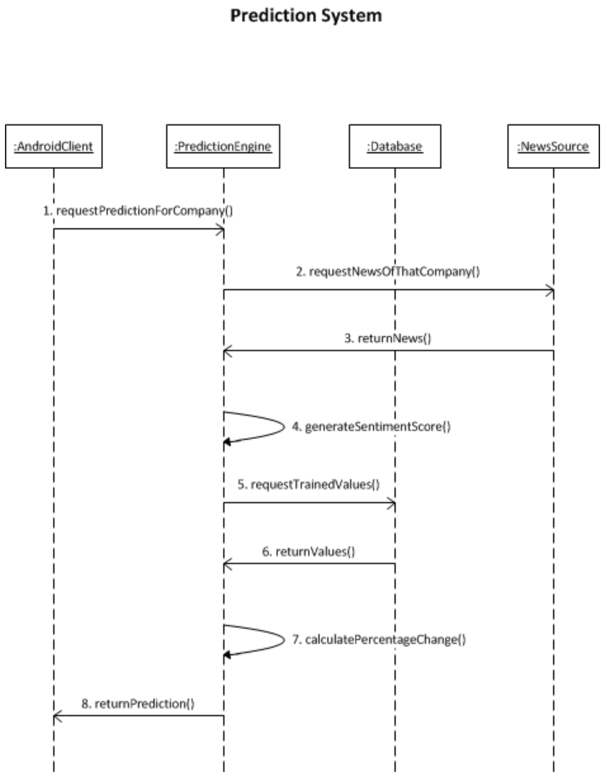

Prediction System
:AndroidClient
:PredictionEngine
:Database
:NewsSource
1. requestPredictionForCompany()
2. requestNewsOfThatCompany()
3. returnNews()
4. generateSentimentScore()
5. requestTrainedValues()
6. returnValues()
7. calculatePercentageChange()
8. returnPrediction()

Figure 3. Sequence Diagram of Prediction Process

Every news story is first pre-processed to remove any stopwords. Then, each article is assigned a sentiment score. To do that, we employ the Subjectivity Lexicon[4], a sentiment dictionary consisting of more than 8000 words. This file provides us with the word w, its type, and its sentiment weight $sw(w)$. The sentiment weight takes any of the following values: strongly positive (4), positive (2), negative (-2), and strongly negative (-4).

When the prediction of a stock price is requested from Android, the application retrieves the latest news related to that particular company c and runs the sentiment analysis algorithm. For each article i , and for each article's word w_j that is included in the dictionary, it updates the company's sentiment score based on this article ($ss_i(c)$) as follows:

$$ss_i(c) = \sum_j sw(w_j)$$

At the end, we have a sentiment score $ss_i(c)$ for each company-related article. If $ss_i(c) > 0$ then we deduct that the news article had positive sentiment ($s_i(c) = 1$). Similarly, if $ss_i(c) < 0$ it had negative sentiment ($s_i(c) = -1$),

[4] http://www.cs.pitt.edu/mpqa/subj_lexicon.html

and if $ss_i(c) = 0$ if was neutral ($s_i(c) = 0$). Finally, the prediction for this company's stock is done as follows:

$$Pred(c) = \begin{cases} up & \text{if } \sum_i s_i(c) > 0 \\ down & \text{if } \sum_i s_i(c) < 0 \end{cases}$$

The system can use the aforementioned algorithm and provide the user with predictions on whether specific stocks will go up or down. However, this data can be also used as input to predict the percentage change in the stock price.

One approach would be to build a prediction model, using machine learning techniques such as decision trees or association rules, that will generate a set of rules linking the sentiment score of each company with a specific percentage change (range) in the stock price. To achieve that, a training process is needed, as described in Figure 2.

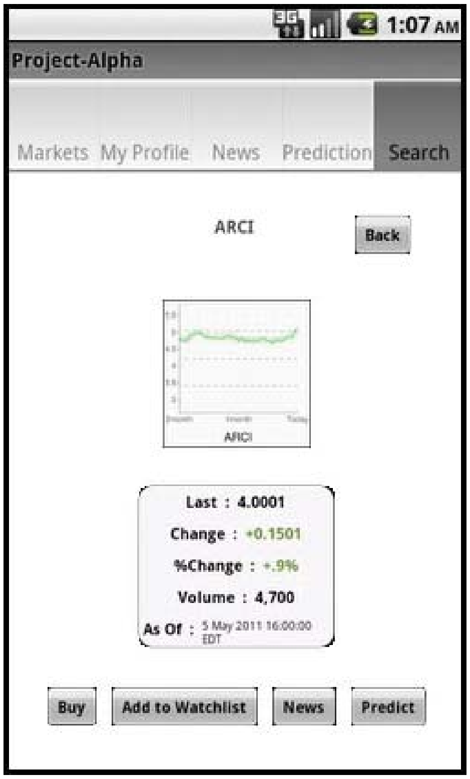
The training of the system is performed as follows: for each company c the system starts by retrieving the latest news and generating a prediction $Pred(c)$ as described above. This prediction is stored in the database. The system periodically retrieves the stock quotes for the same companies and stores the trend (up/down), as well as the percentage of stock change. It also classifies the previous prediction as accurate or not. The process is repeated until we have a statistically significant sample and sufficient prediction accuracy, in which case we can train the system to generate the rules.

In order to generate rules, we can use appropriate machine learning techniques such as association rules or decision trees. The input to the algorithm is the company's ticker (unique id), its sentiment score, and its prediction and the output is a set of rules correlating the aforementioned data with a percentage range that signifies the change of the stock (note that, in the case of association rules, we should keep only the rules that comply with this format). The rules are in turn stored in the database and can subsequently be used for real-time predictions of percentage changes.

5 System Prototype

Based on the proposed architecture, we built a system prototype. The Client was built using Android SDK and ADT plugin for Eclipse. It communicates with the Server using RESTful WebServices hosted on a cloud. The Prediction Engine and the Training Engine are written in Perl, and the Process Engine is written in Java. The database used is MySQL. For more technical details on the prototype's design, the reader may refer to [4].

The user interface was implemented especially to accommodate the limitations of a mobile application. The first time the user uses the application, he/she is asked to provide some personal data in the registration page. Once the registration is complete, the user has access to the aforementioned functionality, through 5 menu options: Markets, MyProfile, News, Prediction, and Search.



Last : 4.0001
Change : +0.1501
%Change : +.9%
Volume : 4,700
As Of : 5 May 2011 16:00:00 EDT

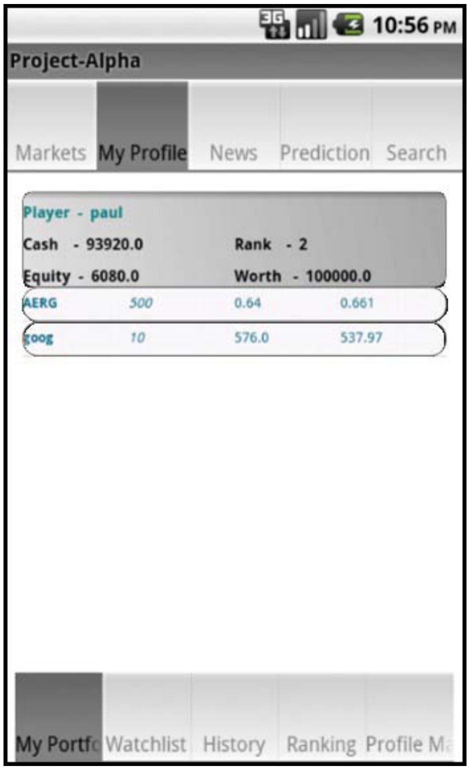
Figure 4. Company details for Appliance Recycling Centers of America (ARCI) stock

The News section allows the user to enter the company name or the ticker of the company whose news the user wants to search. If the text box is left empty, all financial news will appear, ordered by time (newest first).

The Search section allows the user to search for stock quotes. The user may enter the company's name or ticker symbol, or the first letters of any of the above. In the latter case, the user will be given the option to choose among the companies that match this description. Once a company is selected, the system shows the value of the company's stock price, change in the stock, percentage change in the stock, as well as the volume of stocks traded. The user is then given the option to either buy, add to watch list, check the company-related news, or get the prediction for that company's stock. An example is shown in Figure 4.

The user's activity can be reviewed from the MyProfile section. This section provides several options.

The MyPortfolio tab allows the user to see details like their name, rank, amount of money in cash and equities, total current worth, as well as the stocks the user owns. A sample screen is shown in Figure 5. The Watchlist tab shows the stocks the user has put in his/her watch list. These



Player - paul

Cash - 93920.0 Rank - 2

Equity - 6080.0 Worth - 100000.0

AERG	500	0.64	0.661
goog	10	576.0	537.97

Figure 5. User portfolio

are stocks that the user wants to monitor before deciding on buying/selling them. From the History tab, the user may see the list of the stocks he/she has traded, along with details on number of stock, total profit/loss, etc.The Ranking tab allows the user to see their ranking w.r.t. all other users using this application, thus encouraging them to play more (the ranking is based on the total earnings in the simulated environment). Finally, the Markets tab provides real-time data for the major equity indices like NASDAQ, S&P500, Dow Jones (current value and change during the day) and the Profile Management tab allows the user to manage his/her profile.

6 Conclusions and Future Work

In this paper we present the architecture and a prototype of a mobile application which provides users virtual money to buy/sell stocks using real-time data. It allows access to latest financial news, and provides an option to search company specific news. An important component of our system is the stock prediction, which uses sentiment analysis and machine learning to predict the trend and percentage movement of a companys stock. We also incorporated ranking of players based on their performance in order to make the application interesting to the users. The application has been deployed on Android phone and runs smoothly by connecting to the Server using WebServices.

We are currently working on enhancing the application in several ways. We will improve the sentiment dictionary by automatic training: if certain words always lead to accurate/inaccurate predictions, their score will be respectively increased/decreased. We also intend to incorporate social media (e.g. Twitter) in the application. For example, when a user searches for a companys stock, along with the quotes, the application can display the latest tweets related to that company, whereas the sentiments of these tweets can be used in the prediction process. Finally, we will include more news sources as input to the sentiment analysis and update the user interface to show the news related to that company which led to that prediction. This way, the user will have a better idea as to what led to that prediction.

References

[1] S. M. A. Nikfarjam, E. Emadzadeh. Text mining approaches for stock market prediction. In *ICCAE'10*, 2010.

[2] J. Bollen, H. Mao, and X. Zeng. Twitter mood predicts the stock market. *Journal of Computational Science*, 2(1):1–8, March 2011.

[3] R. Choudhry and K. Garg. A hybrid machine learning system for stock market forecasting. In *Proceedings of World Academy of Science Engineering and Technology*, volume 29. May 2008.

[4] K. Jangid and P. Paul. Stock market prediction and simulation tool (ms thesis). Master's thesis, Computer Engineering Dept., San Jose State University, USA, May 2011.

[5] F. Luo, J. Wu, and K. Yan. A novel nonlinear combination model based on support vector machine for stock market prediction. In *WCICA '10*, 2010.

[6] A. Mahajan, L. Dey, and S. K. M. Haque. Mining financial news for major events and their impacts on the market. In *IEEE/WIC/ACM Intl. Conf. on Web Intelligence (WI'08)*, 2008.

[7] R. P. Schumaker and H. Chen. Textual analysis of stock market prediction using breaking financial news: The azfin text system. *ACM Transactions on Information Systems*, 27(2), 2009.

[8] A. Senanayake. Automated neural-ware system for stock market prediction. In *IEEE Conference on Cybernetics and Intelligent Systems*, 2004.

[9] V. H. Shah. Machine learning techniques for stock prediction. In *Foundations of Machine Learning*. 2007.

[10] X. Tang, C. Yang, and J. Zhou. Stock price forecasting by combining news mining and time series analysis. In *IEEE/WIC/ACM Intl. Conf. on Web Intelligence (WI'09)*, 2009.

[11] P. Yoo, M. Kim, and T. Jan. Machine learning techniques and use of event information for stock market prediction: A survey and evaluation. In *Intl. Conf. on Computational Intelligence for Modelling Control and Automation and Intl. Conf. on Intelligent Agents Web Technologies and Internet Commerce*, volume 2. IEEE, 2007.

Using Semantic Relatedness and Locality for Requirements Elicitation Guidance

Stefan Farfeleder
Institute of Computer Languages
Vienna University of Technology
Vienna, Austria
stefan.farfeleder@tuwien.ac.at

Thomas Moser
CDL Flex
Vienna University of Technology
Vienna, Austria
thomas.moser@tuwien.ac.at

Andreas Krall
Institute of Computer Languages
Vienna University of Technology
Vienna, Austria
andi@complang.tuwien.ac.at

***Abstract*—Requirements engineers strive for high-quality requirements which are the basis for successful projects. Ontologies and semantic technologies have been used successfully in several areas of requirements engineering, e.g., for analysis and categorization of requirements. We improve a semantic guidance system which derives phrases from a domain ontology by taking two observations into account: a) domain terms that are semantically related are more likely to be used together in a requirement; and b) the occurrence of domain terms is usually not distributed uniformly over the requirements specification. We define suggestion rankings that make use of these properties resulting in automatically proposing semantically and spatially related domain terms to the requirements engineer. We implement these rankings in our tool and provide an evaluation using three projects from the embedded systems domain. We achieve significant suggestion quality improvements over previous work.**

***Keywords*—domain ontology, semantic relatedness, requirements specification, requirements elicitation, guidance.**

I. Introduction

Any specification errors that propagate from the requirements definition phase into later development phases, such as design or testing, have high impact and typically are expensive to fix. Therefore requirements engineers work hard to find errors in requirements and to have complete, correct and consistent requirements.

Requirements specified using natural language text have inherent ambiguities due to different ways to interpret them by stakeholders. There have been many strategies to reduce the ambiguity in requirement statements. They include restricting the allowed grammar to a subset, e.g., Attempto Controlled English [1], and using predefined vocabularies, e.g., the Language Extended Lexicon [2].

Ontologies have been used for requirements engineering in several ways. Körner and Brumm [3] use domain-independent ontologies to detect linguistic problems in specifications. Other works capture knowledge about the problem domain in the ontology [4][5]. There the ontology acts as a vocabulary of domain terms with additional links between the terms that define their relationships. This enables the analysis of requirement statements with regards to the domain knowledge represented in the ontology, e.g., to find missing requirements. The support for inferring knowledge and reasoning allows automating tasks that otherwise would have to be done manually.

Additionally to the analysis of already specified requirements, [6] showed that a domain ontology can also be used as a guide during the specification of new requirements. By directly using ontology information during requirements definition, this combines the advantages of ontology-based analysis techniques with a reduction of specification effort. The idea is specifying requirements correctly the first time rather than specifying them incorrectly, and then analyzing and improving them. In this work we build upon this guidance system and add a method that actively tries to estimate what the requirements engineer intends to type by taking semantic information and locality into account. The improved system prefers suggestions that are semantically related to concepts already used in the current requirement and that have been used in nearby requirements. These enhancements aim at generating context-sensitive suggestions that are really useful to the requirements engineer.

The remainder of this work is structured as follows. Section II discusses related work; section III motivates our research and presents the research questions. Then our proposed approach is described in section IV, and its evaluation follows in section V. Finally section VI concludes and lists future research topics.

II. Related Work

PROPEL [7] is a tool that provides guidance for defining property specifications which are expressed as finite-state automata. For the definition of a property the user is guided by a *question tree*, a hierarchical sequence of questions. There are separate question trees for a property's behavior and its scope. Based on the answers the tool chooses an appropriate property template. The tool is used to elicitate requirements in the medical domain. Compared to our work the tool elicitates formalized requirements but is less generic, i.e., limited to a small set of requirements.

Kitamura et al. [4] present a requirements elicitation tool that improves requirements quality by ontology-based analysis. The tool analyzes natural language requirements and maps words to domain ontology concepts. According to these occurrences and their relations in the ontology, requirements are analyzed in terms of completeness, correctness, consistency and unambiguity. Compared to this paper's approach the ontology information is only used after requirements have been specified and not during the specification.

Recommendation systems filter items from a large set of data and recommend them to a user based on information about the user's preferences. Maalej and Thurimella [8] provide an overview of possible applications of recommendation systems to requirements engineering. Additionally to the requirement statements themselves and vocabularies - both related to our work - they foresee usage in the area of recommending quality measures, requirements dependencies, people, etc.

SRRS [9] is a recommendation system which supports choosing a requirements elicitation method for security requirements. The requirements engineer needs to assign priorities to ten key characteristics (e.g., unambiguity, traceability, scalability) according to which the system then suggests the most appropriate method.

Machado [10] uses semantic web techniques in the field of requirements engineering to extract semantic information from requirement documents. The extracted information is used to track changes between different document versions and for semantic queries. Martoglia [11] collects semantic information from a large number of documents about software quality. Using similarity measures the system supports suggesting documents that are related to a given natural language query. Unlike our own approach there is no ontology support to specify new requirements in these works.

Literature distinguishes between semantic *relatedness* and semantic *similarity*. Similarity is a specific kind of relatedness. Reusing an example from Resnik [12], a *car* is *similar* to a *bicycle* - both are a kind of vehicle - but the *car* is *related* to *gasoline*, because it *requires* gasoline to drive. Semantic similarity only considers subclass-of relations between concepts while semantic relatedness takes all kinds of relations into account, e.g., meronymy (part-of) and functional relationships (*require* in the example above).

Budanitsky and Hirst [13] compare five measures of lexical semantic relatedness and similarity that are based upon WordNet[1]. WordNet contains synonymy, hyponymy (is-a, subclass-of), several kinds of meronymy (part-of) and antonymy (opposite-of) relations. The measures are compared to human judgment and in the context of an application to detect malapropism, the usage of lexically similar but semantically incorrect words. Unfortunately four out of five measures only handle similarity.

III. Research Issues

In previous work [6] we presented a semantic guidance system for requirements elicitation. It proposes phrases derived from the knowledge contained in a domain ontology. While the system works well for small ontologies, we found practical issues when applying it to more requirements and bigger ontologies due to the large number of generated proposals:

- The requirements engineer needs to type more characters until filtering limits the proposals to a manageable amount.
- A requirements engineer not familiar with the domain does not benefit from seeing a very long list of proposals, e.g., when a domain term slipped his mind.

Clearly there must be a better method than showing all matching proposals in alphabetical order. During the study of these problems we thought about what constitutes "good" proposals. We observed the following two properties.

A. Semantic Relatedness Property

While writing a requirement statement it is possible to use already specified requirement parts, combined with the ontology knowledge, to predict the remainder of the statement, at least to a certain degree. Consider a requirement starting with "The Safing Controller shall be able to". We can support the requirements engineer by suggesting concepts that are semantically related to the concept *Safing Controller* for the following parts of the requirement, e.g., *SafeAct mode* in Fig. 1.

We identified the following common types of semantic relations in requirements (this list is intended to be exemplary and not exhaustive):

- Function: Requirements of the form "*subject* shall [be able to] *verb object*". Given an ontology link between *subject* and *object*, we can suggest *object* if the requirements already contains *subject*.
- Restriction: Requirements of the form "if *event* then *subject* shall [...]" or "during *state subject* shall [...]". Given an ontology link between a concept in *event/state* and *subject*, we can suggest *subject* if the requirements already contains *event/state*.
- Architecture: Requirements of the form "*system* shall have *subsystem*". Given an ontology link between *system* and *subsystem*, we can suggest *subsystem* if the requirements already contains *system*.

In this work we assume having a suitable domain ontology containing such links. Ontology extraction techniques (e.g. [5]) can be used to extract domain ontologies from text documents.

B. Locality Property

The occurrences of many domain terms used in a requirements specification are not randomly distributed over the entire set of requirements but are clustered around a certain location in the document. This is only natural considering that most requirements specifications are organized into chapters each describing an individual part of the system.

As a quick check for this observation we measured the distributions of the requirement indices for concepts that occur at least twice and computed the standard deviations. For example, a concept occurring in two subsequent requirements has a standard deviation of 0.5, one occurring in requirement 1 and in requirement 100 a standard deviation of 49.5. On average, the standard deviations of those distributions were 11.60, 35.03 and 9.87 for the three requirements sets used in

[1] http://wordnet.princeton.edu/

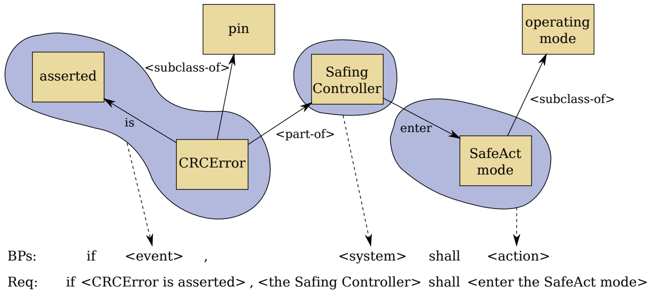

pin
operating mode
asserted
<subclass-of>
Safing Controller
<subclass-of>
is
CRCError
<part-of>
enter
SafeAct mode
BPs: if <event> , <system> shall <action>
Req: if <CRCError is asserted> , <the Safing Controller> shall <enter the SafeAct mode>

Fig. 1. Domain Ontology, Suggestions, Boilerplates and Requirement

our evaluation. This is considerably lower than for randomly distributed indices (28.58, 81.69 and 24.82).

We call two concepts that occur in nearby requirements *spatially related*.

The idea of this research is to find out whether we can make use of those properties and whether this actually improves the guidance system. We have two research questions:

- *RQ1*: Does the guidance system improve if we preferably suggest semantically related concepts?
- *RQ2*: Does the guidance system improve if we preferably suggest spatially related concepts?

We are optimistic that both questions can be answered positively and define the following hypotheses:

- *H1*: Suggesting semantically related concepts improves the guidance system.
- *H2*: Suggesting spatially related concepts improves the guidance system.

IV. Guidance System

This section briefly summarizes the existing semantic guidance system and then goes on to introduce the new contributions, starting with section IV-D.

The goal of the guidance system is assisting the requirements engineer with specifying requirements. It does that by proposing textual phrases which are derived from the ontology information. Fig. 1 depicts a part of the ontology and how the guidance is related. The boxes at the top and the arrows between them are ontology contents while the blue areas represent the derived phrases (“CRCError is asserted”, “the Safing Controller” and “enter the SafeAct mode”). Axioms are labelled with brackets to make the distinguishable from named relations. A requirement using these suggestions can be seen at the very bottom.

A. Boilerplate Requirements

Our approach uses *boilerplates* for requirement statements. This term was coined by J. Dick [14] and refers to a textual requirement template. A boilerplate consists of a sequence of attributes and fixed syntax elements. A common boilerplate is “⟨*system*⟩ `shall` ⟨*action*⟩”. In this boilerplate ⟨*system*⟩ and ⟨*action*⟩ are attributes and `shall` is a fixed syntax element. It is possible to combine several boilerplates by means of concatenation (Fig. 1: “`if` ⟨*event*⟩`,`” and “⟨*system*⟩ `shall` ⟨*action*⟩”). This allows keeping the number of required boilerplates low while at the same time having a high flexibility. During instantiation textual values are assigned to the attributes of the boilerplates; a boilerplate requirement is thus defined by its boilerplates and its attribute values.

We use boilerplates for our approach due to two reasons: a) the usage of different attributes allows providing context-sensitive guidance (section IV-C), i.e., proposing different suggestions depending on the current attribute, and b) using boilerplates helps specifying requirements that are syntactically uniform when using a small number of templates (compared to the number of requirements).

There are numerous other template-based approaches for requirements specification, most of them being more formal than boilerplates. Post et al. [15] report on successfully applying a formal specification pattern system defined by Konrad and Cheng [16] on automotive requirements. However, the authors limit themselves to a certain class of requirements, the behavioral requirements. Our approach aims at covering all kinds of textual requirements.

B. Domain Ontology

A common definition of the term *ontology* is that it is a *formal, explicit specification of a shared conceptualization* [17]. A domain ontology focuses on a specific subject, here the system under construction. For requirements engineering the aspect of sharing is of particular interest, it means that stakeholders agree on terminology and term relations. By making use of this information we lower the risk for requirements ambiguity.

The following ontology entities are used for the guidance system:

- Concepts: Ontology concepts are the terms the requirements engineer uses in the requirements. This includes actors, components, events, states, etc. of the system under construction. In Fig. 1 the concepts are “asserted”, “CRCError”, “pin”, “Safing Controller”, “SafeAct mode” and “operating mode”.
- Relations: Relations are links between concepts. We use two kinds of relations:
  - A named relation represents a functional relationship between a subject concept and an object concept. The relation name is expected to be a transitive verb. Named relations are used to derive suggestions.
  - Anonymous relations simply indicate that two concepts are related. This is used to prefer semantically related suggestions.

  In Fig. 1 there are two named relations: “is” between concepts “CRCError” and “asserted”, and “enter” between “Safing Controller” and “SafeAct mode”.
- Axioms: Axioms are relations between concepts with a special meaning. An *equivalence* axiom represents the knowledge that two concepts refer to the same phenomenon in the domain (synonyms). A *subclass-of* axiom states that one concept is a subclass of another one.

Both kinds of axioms lead to an inheritance of relations from the equivalent or parent concept. The *part-of* axiom imposes a hierarchical structure on the ontology contents which facilitates navigation.

Additionally the ontology contains links that classify concepts into one of the boilerplate attributes (the link between "Safing Controller" and ⟨*system*⟩ in Fig. 1).

C. Suggestions

From the ontology knowledge the guidance system infers three kinds of suggestions:

- Concept: This basic kind of suggestion consists of the name of an ontology concept, optionally prefixed with the determiner "the".
- Verb-Object: From a named ontology relation our system proposes the relation's verb in infinitive form followed by the name of the relation's destination concept (again optionally prefixed with "the"). This suggestion is intended to follow "shall" or "shall be able to" formulations often found in requirements.
- Subject-Verb-Object: From a named ontology relation our system proposes the relation's source concept, followed by the relation's verb in third person singular form, and the relation's destination concept - both concept names optionally prefixed with "the". This suggestion is intended to be used in clauses starting with "if", "while" or similar words.

In Fig. 1 an example is provided for each suggestion kind: "the Safing Controller" for Concept, "enter the SafeAct mode" for Verb-Object and "CRCError is asserted" for Subject-Verb-Object. It is not always grammatically correct to add a determiner before a concept name, e.g., before "asserted". Thus our approach checks the part-of-speech (a classification of a word into one of several types, e.g., noun, verb, adjective) to determine this.

The three suggestion kinds are used for different boilerplate attributes. Table I shows the mapping between suggestions and attributes. For the attributes *event* and *state* both Concept suggestions and Subject-Verb-Object are used due to grammatical reasons, e.g., a noun should be used for `during` ⟨*state*⟩ but `if` ⟨*state*⟩ requires a clause. The Verb-Object suggestions are used for the ⟨*action*⟩ attribute which generally follows modal verbs like "shall"; the remaining boilerplate attributes use only the Concept suggestions.

D. Semantic Relatedness of Suggestions

We mentioned functional, restrictional and architectural relations in requirements earlier. These relations have in common that they do not correspond to simple is-a ontology links. While there exist requirements where is-a links are useful, e.g., *"The system shall have the following states: ... "*, they are rare. From this point of view semantic relatedness is more important to us than semantic similarity. Using the car example mentioned earlier, a possible requirement is *"The driver shall be able to refuel the car with gasoline."* When proposing phrases to requirements engineers, we want to make use of semantic relatedness between concepts, e.g., given a requirement that already contains *car* we would rather suggest *gasoline* than *bicycle*.

Unfortunately from the review of related literature it seems that researchers concentrate more on similarity than on relatedness. We were able to find complex functions to compute semantic similarity using up to six different factors [18]: link type, node depth, local density, link strength, node attributes and cluster granularity degree. Most of them only make sense in an is-a taxonomy and do not apply well to the more generic graph we are using. We experimented with using different weights depending on link types but that had no measurable effect. In the end we decided to go for simplicity: We measure the relatedness of two concepts simply by using the edge count of the shortest path between two concepts.

We define C to be the set of concepts, T to be the set of link types in the ontology, $L \subseteq C \times C \times T$ to be the set of all ontology links and the direct distance δ_l between two concepts s and d to be

$$\delta_l(c, c') = \begin{cases} 1 & \text{if } \exists\, t \in T : \langle c, c', t\rangle \in L \vee \langle c', c, t\rangle \in L, \\ \infty & \text{otherwise.} \end{cases}$$

We use ontology links in an undirected manner to be more flexible with the order in which concepts are stated in a requirement, e.g., we can handle "*subject* shall *action* if *event*" even though the link might be directed from *event* to *subject*. Moreover we do not require instantiating boilerplate attributes from left to right. Using δ_l we can compute the shortest path δ_p between two concepts. In Fig. 1 the distance between *Safing Controller* and *SafeAct mode* is one, the distance between *operating mode* and *assert* is four.

We define R to be the set of requirements and S to be the set of suggestions. Further we define $cr : R \to \mathcal{P}(C)$ and $cs : S \to \mathcal{P}(C)$ to map a requirement or, respectively, a suggestion to the concepts it uses. Next we define the distance δ_s between a requirement r and a suggestion s to be

$$\delta_s(r, s) = \max(\min_{c \in cr(r)} \min_{c' \in cs(s)} \delta_p(c, c'), 1).$$

It is the minimum distance between any concept used in the requirement and any concept used in the suggestion. We cap it at one to avoid first proposing suggestions containing concepts already used in the current requirement.

E. Locality of Suggestions

Requirements documents are often structured into sections each covering a different aspect of system functionality. Terms specific to that functionality occur mostly there. In order to measure the spatial relatedness of two requirements, we consider the requirements to be a list and define an index function $ind : R \to \mathbb{N}$. Requirements that have similar indices are spatially related. When specifying a new requirement, the requirements engineer must indicate the index where the new requirement should be inserted into the list. This is similar to choosing a section where the requirement should be added to.

We define the locality factor loc of a requirement r and a suggestion s in the following way:

$$loc(r,s) = \min_{r' \in R} \begin{cases} |ind(r) - ind(r')| & \text{if } cr(r') \cap cs(s) \neq \emptyset, \\ \infty & \text{otherwise.} \end{cases}$$

It is the index distance to the nearest requirement that shares at least one concept with the suggestion.

F. Suggestion Algorithm

Based on the previous observations we define four different suggestion rankings. The first one (alpha) sorts suggestions alphabetically and serves as a baseline. The second (sem-rel) and third one (locality) sort using the semantic relatedness function δ_s and locality function loc, respectively. The fourth order (ML) uses a linear combination of both functions. A machine learning approach is used to obtain good coefficients for the combination.

Our algorithm for suggestions in a requirement r given a (possibly empty) word w the requirements engineer is currently typing for boilerplate attribute a consists of the following steps:

1) Compute all suggestions and filter out those not starting with w.
2) Sort remaining suggestions by either
   a) Matching boilerplate attribute a (suggestions marked by "✓" in Table I), and then alphabetically, using δ_s or loc; or
   b) Using the weight computed by ML coefficients

Using these orders we achieve that semantically and spatially related suggestions are ranked and shown before other suggestions.

For the ML ranking we use linear regression models, one for each boilerplate attribute. The models are trained using the following features: suggestion type, matching boilerplate attribute, value of δ_s and loc and whether a suggestion is correct (prediction value 1.0) or not (0.0) in the context of the other features. A higher value computed by such a model is considered to be a better match. We use linear regression models because they are very efficient at computing the weights of all suggestions.

V. Evaluation

To test hypotheses H1 and H2 we implemented the enhancements in our tool DODT (Domain Ontology Design Tool). We set up an evaluation with three different industrial projects: DMS (Doors Management System) which controls aircraft doors, an airbag controller that decides if an airbag should be deployed and a power-switch used in the powertrain domain. For each project a domain ontology was extracted from domain documents using term and relation extraction techniques and some manual pruning and refinement. Table II gives information about the requirements and the domain ontologies of the projects.

TABLE I
Attribute Suggestion Mapping

Attribute	Con	VO	SVO
⟨*system*⟩, etc.	✓		
⟨*action*⟩		✓	
⟨*event*⟩, ⟨*state*⟩	✓		✓

TABLE II
Project Measures

Item	DMS	Airbag	Power-train
Reqs.	99	283	86
Concepts	233	591	267
Relations	164	425	115
Axioms	100	571	163

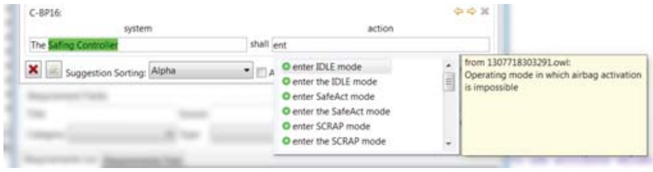
TABLE III
Suggestion Results

	alpha		sem-rel		locality		ML	
len	avg	σ	avg	σ	avg	σ	avg	σ
				DMS				
0	134.9	135.1	81.43	101.4	102.7	124.9	69.39	103.9
1	6.99	14.32	4.02	9.09	5.50	14.09	3.69	7.22
2	1.66	3.00	1.03	2.56	1.34	2.92	0.89	1.95
3	0.98	2.44	0.70	2.29	0.85	2.35	0.61	1.26
4	0.68	1.39	0.48	1.19	0.61	1.32	0.56	1.13
5	0.57	1.33	0.43	1.18	0.53	1.28	0.50	1.16
				Airbag				
0	353.0	332.7	161.9	228.9	204.7	264.0	100.1	196.5
1	29.00	67.16	13.74	31.79	18.54	55.27	8.53	31.40
2	5.94	28.01	2.77	8.06	3.56	16.76	1.89	7.35
3	3.15	5.40	1.81	4.02	1.79	3.77	1.15	2.96
4	1.18	1.98	0.90	1.79	0.88	1.72	0.61	1.42
5	0.64	1.42	0.49	1.28	0.50	1.23	0.33	0.89
				Powertrain				
0	224.9	200.7	154.4	175.2	153.8	176.1	107.5	154.8
1	28.95	56.13	19.50	48.31	15.55	38.69	9.42	28.99
2	2.80	6.02	2.03	5.18	2.36	5.48	1.15	2.86
3	1.95	5.27	1.40	4.68	1.65	4.84	0.84	2.27
4	1.74	5.30	1.19	4.62	1.45	4.80	0.86	2.39
5	1.67	5.37	1.15	4.68	1.37	4.81	0.83	2.36

We took the following approach to measure the difference between the baseline suggestion order (alpha) and the orders evolving from the research questions RQ1 (sem-rel), RQ2 (locality) and the ML order. First we identified all places where a requirement phrase p (one or more words of the requirement) matched one of our suggestions. In case several suggestions matched at the same position, we used the longest one. Then we computed the suggestion list our algorithm yielded and measured at which position in this list the expected phrase p was ranked. This we repeated for all substrings of p of length zero to five and for all suggestion orders. This simulates the behavior when a user types in the first p characters of the expected phrase and selects a suggestion. The requirements were added to our guidance system in the same order that is used in the projects. We argue the average list position is a good measure for the performance of a suggestion order as the user starts seeing the first N entries (depending on the window size) and has to scroll down for further entries. Fig. 2 shows a screenshot of our tool presenting suggestions. To avoid overfitting to the evaluation data we used a three-fold cross-validation for the ML order, i.e., for each project we trained the model using data from the other two projects.

Table III presents our results. It shows the arithmetic means and standard deviations of the list positions for all suggestion orders, projects and lengths. For example: If an Airbag project user enters the first two letters of a phrase, the expected phrase will be on average at position 5.94 for the alpha order and at position 1.89 for the ML order.

All three suggestion orders using semantic relatedness



C-BP16:

system

action

The Safing Controller shall ent

Suggestion Sorting: Alpha

- enter IDLE mode
- enter the IDLE mode
- enter SafeAct mode
- enter the SafeAct mode
- enter SCRAP mode
- enter the SCRAP mode

from 1307718303291.owl:
Operating mode in which airbag activation is impossible

Fig. 2. DODT Screenshot showing Suggestions

and/or locality are an improvement over the baseline order with regards to the position of the correct suggestion when comparing means. The ML order generally performs best except for DMS at lengths 4 and 5 where sem-rel performs better. We tested statistical significance using a Student's t-test with a confidence level of 95% (rejecting the null hypothesis of identical distributions). The improvements are significant for ML (DMS: len$\leq$4, Airbag: all, Powertrain: all), for sem-rel (len$\leq$3, all, len$\leq$2) and for locality (len$\leq$1, all, len$\leq$2). For greater lengths the improvements are not significant. With these restrictions in mind, we claim our hypotheses to be true. We achieve the biggest improvements at length zero (start of a new phrase) because here the alpha suggestion order is basically random while our proposed orders rank related suggestions first. At longer phrases this advantage decreases because filtering often reduces the suggestions to a small set of very similar ones which is more difficult to rank in a good way, especially if there is a long common prefix.

A. Threats to Validity

The experiment was performed in a fully automated way and is thus repeatable. The quality of the suggestion order depends on the ontology quality, especially on the links for the semantic relatedness. The domain ontology creation process involved selection and validation by a domain expert. While our evaluation consists of three different domains, ontologies and requirement sets, the claim of general usefulness of this approach needs to be tested in a larger study with more projects.

VI. Conclusion and Future Work

In this paper we presented two enhancements to a semantic guidance system. The first one is taking the semantic relatedness between concepts in a requirement statement into account. The second enhancement is taking the locality of domain terms with regards to their occurrence location in the requirements document into account. The probability for reuse of semantically and spatially related concepts is increased. We defined ranking on suggestions to exploit these properties. An evaluation using three sets of industrial requirements showed that both properties lead to an improvement of the suggestion quality.

More research needs to be done on deriving phrases from ontology concepts and relations. There are grammatical forms in our requirements, e.g., gerunds, which could be supported additionally. However, this needs to be evaluated carefully as there is a trade-off between convenience for the requirements engineer and achieving requirements consistency by using the same phrases in all requirements. Finally we plan to research specialized ontology extraction methods to extract exactly the kind of concepts and relations we are interested in.

Acknowledgments

The research leading to these results has received funding from the ARTEMIS Joint Undertaking under grant agreement N^{o} 100016 and from specific national programs and/or funding authorities. This work has been supported by the Christian Doppler Forschungsgesellschaft and the BMWFJ, Austria.

References

[1] N. Fuchs, U. Schwertel, and R. Schwitter, "Attempto Controlled English - not just another logic specification language," in *LOPSTR 1999*. Springer, 1999, pp. 1–20.

[2] J. Leite and A. Franco, "A strategy for conceptual model acquisition," in *Requirements Engineering, 1993., Proceedings of IEEE International Symposium on*. IEEE, 1993, pp. 243–246.

[3] S. Körner and T. Brumm, "Natural language specification improvement with ontologies," *International Journal of Semantic Computing (IJSC)*, vol. 3, no. 4, pp. 445–470, 2010.

[4] M. Kitamura, R. Hasegawa, H. Kaiya, and M. Saeki, "A supporting tool for requirements elicitation using a domain ontology," in *ICSOFT 2009*. Springer, 2009, pp. 128–140.

[5] I. Omoronyia, G. Sindre, T. Stålhane, S. Biffl, T. Moser, and W. Sunindyo, "A domain ontology building process for guiding requirements elicitation," in *REFSQ 2010*. Springer, 2010, pp. 188–202.

[6] S. Farfeleder, T. Moser, A. Krall, T. Stålhane, I. Omoronyia, and H. Zojer, "Ontology-driven guidance for requirements elicitation," in *ESWC 2011*. Springer, 2011, pp. 212–226.

[7] R. Cobleigh, G. Avrunin, and L. Clarke, "User guidance for creating precise and accessible property specifications," in *14th International Symposium on Foundations of Software Engineering*. ACM, 2006, pp. 208–218.

[8] W. Maalej and A. Thurimella, "Towards a research agenda for recommendation systems in requirements engineering," in *2009 Second International Workshop on Managing Requirements Knowledge*. IEEE Computer Society, 2009, pp. 32–39.

[9] J. Romero-Mariona, H. Ziv, and D. Richardson, "SRRS: a recommendation system for security requirements," in *Proceedings of the 2008 International Workshop on Recommendation Systems for Software Engineering*. ACM, 2008, pp. 50–52.

[10] B. N. Machado, L. de Oliveira Arantes, and R. de Almeida Falbo, "Using semantic annotations for supporting requirements evolution," in *Proc. SEKE*. Knowledge Systems Institute, 2011, pp. 185–190.

[11] R. Martoglia, "Facilitate IT-providing SMEs in software development: a semantic helper for filtering and searching knowledge," in *Proc. SEKE*. Knowledge Systems Institute, 2011, pp. 130–136.

[12] P. Resnik, "Using information content to evaluate semantic similarity in a taxonomy," in *Proceedings of the 14th International Joint Conference on Artificial Intelligence*. Morgan Kaufmann Publishers Inc., 1995, pp. 448–453.

[13] A. Budanitsky and G. Hirst, "Evaluating WordNet-based measures of lexical semantic relatedness," *Computational Linguistics*, vol. 32, no. 1, pp. 13–47, 2006.

[14] E. Hull, K. Jackson, and J. Dick, *Requirements engineering*. Springer, 2005.

[15] A. Post, I. Menzel, and A. Podelski, "Applying restricted english grammar on automotive requirements - does it work? a case study," in *REFSQ 2011*. Springer, 2011, pp. 166–180.

[16] S. Konrad and B. Cheng, "Real-time specification patterns," in *Proceedings of the 27th International Conference on Software Engineering*. ACM, 2005, pp. 372–381.

[17] R. Studer, V. Benjamins, and D. Fensel, "Knowledge engineering: principles and methods," *Data & knowledge engineering*, vol. 25, no. 1-2, pp. 161–197, 1998.

[18] X. Xu, J. Huang, J. Wan, and C. Jiang, "A method for measuring semantic similarity of concepts in the same ontology," in *2008 International Multi-symposiums on Computer and Computational Sciences*. IEEE, 2008, pp. 207–213.

Phases, Activities, and Techniques for a Requirements Conceptualization Process

Alejandro Hossian
Technological National University at Neuquén
& PhD on Computer Science Program,
National University of La Plata. Buenos Aires, Argentina
alejandrohossian@yahoo.com.ar

Ramón Garcia-Martínez
Information Systems Research Group
National University of Lanus
Remedios de Escalada, Buenos Aires, Argentina
rgarcia@unla.edu.ar

***Abstract*—The requirements elicitation process, whose main objective is to give birth to the requirements, not only is a technical process to build a particular system but also an important process of social connotations involving different people (stakeholders), a circumstance which causes certain problems arise when carrying out this process of requirement conceptualization. We propose a process of Requirements Conceptualization that are structured in two phases: (a) Problem-Oriented Analysis: aimed at understanding the problem given by the user in the domain in which this takes place, and (b) Product-Oriented Analysis: its aim is to obtain the functionalities that the user intends to obtain from the software product to be developed, taking into account the relationship of these features with the reality expressed by the user in his speech. The techniques for each activity in both phases are introduced.**

Keywords - Requirements Conceptualization; Process; Phases; Activities; Techniques.

I. INTRODUCTION

The requirements elicitation process, whose main objective is to give birth to the requirements, not only is a technical process to build a particular system but also an important process of social connotations [1] that involves different people (stakeholders). It is usual that the process of requirements elicitation causes problems when it is been carrying out [2]. Similarly, with regard to the stakeholders it is clear that the term is used in reference to any person or group that is affected by the system directly or indirectly, between them can be cited to end users who interact with the system and as well as others who may be affected by the implementation of it (maintenance professionals providing other related systems, experts in the domain of the system, business managers, others).

Now, in light of all the constraints that make mention by Sommerville, proper of requirements elicitation process is that there is a need to explore and analyze those features that are inherent to this process and, as such, contribute to characterize the process. Characterized the task of requirement elicitation, it follows that the axis of it focuses on establishing communication between the User and the Requirements Engineer. When developing their work in elicitation, this must capture and model a reality that frames a problem, whose solution must be approached through a software product. Since this is really an intangible element, usually too complex, it is also difficult to capture.

These problems, taken from the elicitation process, make it difficult for the requirements engineer to develop the stakeholder universe of discourse, as well as the construction of adequate conceptual models [3][4], i.e. these problems, which begin to manifest themselves in the process from requirements elicitation and communication between the user and the engineer, probably will be propagated in the activity of construction of conceptual models. These drawbacks inexorably converge towards obtaining low-quality software.

In this context, the problem is focused (Section 2), we propose a process of requirements conceptualization (Section 3), the techniques proposed for the activities in each phase are a presented (Section 4), and conclusions and future research work is outlined (section 7).

II. PROBLEM DESCRIPTION

The open problem identified in this section, is the need to structure and categorize the information body coming from the elicitation process. The purpose is facilitating the understanding of the problem expressed by the user [5][6], in other words, to conceptualize the requirements. Inadequate treatment of the complexity contained in the user's discourse has been highlighted by several authors [7][8][9][10]. These authors mention the difficulties in building conceptual models based on the information contained in the elicitation process and reflected in the user's speech. Also worth noting that these difficulties give the analysis process to a degree of immaturity which makes it difficult to perform effectively in this activity, while difficult to adopt this approach in organizations [11]. Accordingly and pursuant to the foregoing, the open problem addressed in this paper, is a "perception gap" [5][7] in the transition of a process (requirements elicitation) to another process (Conceptual Modeling). Because of this, is clearly a need to conceptualize the requirements stated by the user in his speech before going to the construction of conceptual models in order to reduce complexity and promote understanding referred to the problem described by the user, contributing to the achievement of better quality of Conceptual Models.

III. PROPOSAL OF PROCESS OF REQUIREMENTS CONCEPTUALIZATION

The solution proposed in this work involves the insertion of an activity of Requirements Conceptualization, which aims to act as a bridge or "link" between the activities of requirements elicitation and the activities conceptual modeling, thereby facilitating the understanding of the problem expressed by the user and therefore obtain higher quality Conceptual models [2][3][5][10][12].

The process of conceptualizing the proposed requirements is done through the so-called Requirements Conceptualization Process which is developed in two phases: (a) Problem-Oriented Analysis, whose goal is to understand the problem posed by the user in the domain in which this takes place, and (b) Product-Oriented Analysis, whose goal is to obtain the functionality that the user intends to obtain from the software product to be developed, taking into account the relationship of these features with the reality expressed by the user in his speech. Figure 3 represents the process of Requirements Conceptualization with focus on interdependence between the phases, tasks and products.

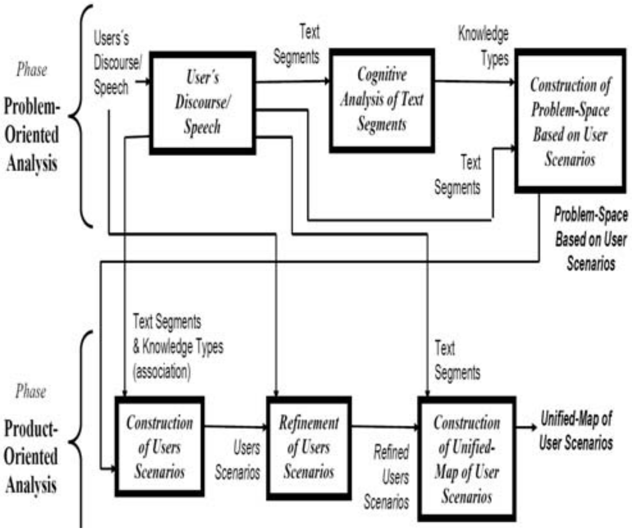
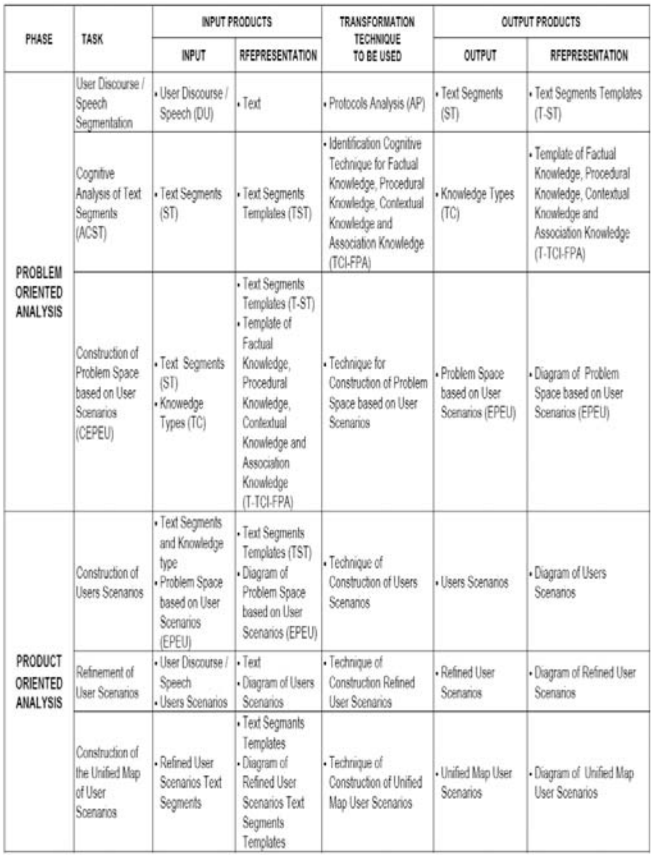

Phase
Problem-Oriented Analysis
Users´s Discourse/ Speech
User´s Discourse/ Speech
Text Segments
Cognitive Analysis of Text Segments
Knowledge Types
Construction of Problem-Space Based on User Scenarios
Text Segments
Problem-Space Based on User Scenarios
Phase
Product-Oriented Analysis
Text Segments & Knowledge Types (association)
Text Segments
Construction of Users Scenarios
Users Scenarios
Refinement of Users Scenarios
Refined Users Scenarios
Construction of Unified-Map of User Scenarios
Unified-Map of User Scenarios

Figure 1. Process of requirements conceptualization; detailing: stages, tasks and products

Problem-Oriented Analysis phase is divided into three tasks: (a) "User Discourse/Speech Segmentation", (b) "Cognitive Analysis of Text Segments", and (c) "Construction of Problem Space based on User Scenarios". The "Discourse of Natural Language User" (which from now on in this paper we will call user speech) is the input for the task "User Discourse/Speech Segmentation" that results in the "Text Segments". These segments are the input to task, "Cognitive Analysis of the Text Segments" generating the respective "Knowledge Types". The "Text Segments" and "Knowledge Types" are the inputs for the task "Construction of Problem Space based on User Scenarios" that will result in "Problem Space based on User Scenarios".

Product-Oriented Analysis phase is divided into three tasks: (a) "Construction of Users Scenarios", "(b) "Refinement of User Scenarios", and (c) "Construction of the Unified Map of User Scenarios". The "Text Segments & Knowledge Types Association" and the "Problem Space based on User Scenarios" constitute the inputs for the task "Construction of User Scenario". These scenarios along with the "User Speech" respectively are the input to task "Refinement of Scenarios User" that generates the respective "Refined User Scenarios". These, and "Text Segments" are the inputs of the task "Construction of the Unified Map User Scenarios", that result in the "Unified Map User Scenarios". The techniques and representations of the tasks in the problem-oriented analysis phase are summarized in Figure 4.



PHASE	TASK	INPUT PRODUCTS		TRANSFORMATION TECHNIQUE TO BE USED	OUTPUT PRODUCTS	
		INPUT	RFEPRESENTATION		OUTPUT	RFEPRESENTATION
PROBLEM ORIENTED ANALYSIS	User Discourse / Speech Segmentation	• User Discourse / Speech (DU)	• Text	• Protocols Analysis (AP)	• Text Segments (ST)	• Text Segments Templates (T-ST)
	Cognitive Analysis of Text Segments (ACST)	• Text Segments (ST)	• Text Segments Templates (TST)	• Identification Cognitive Technique for Factual Knowledge, Procedural Knowledge, Contextual Knowledge and Association Knowledge (TCI-FPA)	• Knowledge Types (TC)	• Template of Factual Knowledge, Procedural Knowledge, Contextual Knowledge and Association Knowledge (T-TCI-FPA)
	Construction of Problem Space based on User Scenarios (CEPEU)	• Text Segments (ST) • Knowedge Types (TC)	• Text Segments Templates (T-ST) • Template of Factual Knowledge, Procedural Knowledge, Contextual Knowledge and Association Knowledge (T-TCI-FPA)	• Technique for Construction of Problem Space based on User Scenarios	• Problem Space based on User Scenarios (EPEU)	• Diagram of Problem Space based on User Scenarios (EPEU)
PRODUCT ORIENTED ANALYSIS	Construction of Users Scenarios	• Text Segments and Knowledge type • Problem Space based on User Scenarios (EPEU)	• Text Segments Templates (TST) • Diagram of Problem Space based on User Scenarios (EPEU)	• Technique of Construction of Users Scenarios	• Users Scenarios	• Diagram of Users Scenarios
	Refinement of User Scenarios	• User Discourse / Speech • Users Scenarios	• Text • Diagram of Users Scenarios	• Technique of Construction Refined User Scenarios	• Refined User Scenarios	• Diagram of Refined User Scenarios
	Construction of the Unified Map of User Scenarios	• Refined User Scenarios Text Segments	• Text Segmants Templates • Diagram of Refined User Scenarios Text Segments Templates	• Technique of Construction of Unified Map User Scenarios	• Unified Map User Scenarios	• Diagram of Unified Map User Scenarios

Figure 4. Phase, task and products

IV. TECHNIQUES FOR PHASE OF PROBLEM-ORIENTED ANALYSIS

This section presents techniques for the phase Problem-Oriented Analysis, which are: Technique for User´s Discourse Segmentation (TS - DU) used to the implementation of task of User´s discourse segmentation (SDU) (section IV.A), Cognitive Techniques to Identificate different types of Knowledge as: factual knowledge, Procedural knowledge, Contextual knowledge and Association knowledge (TCI - CFPCA) for the implementation of task Cognitive Analysis of Text Segments (ACST) (section IV.B) and the Technique for Building the Problem Space Diagram of User´s scenarios (TCD - EPEU) for the implementation of task Building the Problem Space of User´s scenarios (CEPEU) (section IV.C).

A. Technique for User´s Discourse Segmentation (TS - DU)

By means of this technique is implemented the first task that requirement engineer (RE) has to carry in the early stage of the phase Problem-Oriented Analysis, called Technique for User´s Discourse Segmentation (TS - DU). For application of

TS – DU the RE uses as input the User´s Discourse in plain text to segment it sentence by sentence [13], integrating these sentences in Text Segments (ST) are related with real situations described by user. Finally the ST associated to user’s scenarios (EU) are obtained. ST and EU are the output of this technique which is summarized in table 1.

TABLE I. TECHNIQUE FOR USER´S DISCOURSE SEGMENTATION (TS - DU)

Technique:	User´s Discourse Segmentation (TS - DU)
Input:	User´s Discourse
Output:	Text Segments (ST) associated to user´s scenarios (EU)
Step 1.	User´s discourse segmentation (DU) sentence by sentence (In this first step is performed a preliminary analysis of DU looking segmenting in short sentences. This initial segmentation allows a simpler treatment of DU to meet the step 2 of this process. Short sentences are the output obtained for this step)
Step 2.	Integration of sentences in Text Segments (ST) (In this second step integrates the sentences obtained in step 1 into segments of text (ST) describing a situation of reality. These ST are formed by sets of short sentences, and are the output for this step).
Step 3.	Association of Text Segments(ST) to User´s Escenarios (EU) (In this third step, each segment of text obtained is associated with a user scenario obtained in step 2. Therefore, as a result of this process are obtained Text Segments (ST) associated with User Scenarios (EU), which are the output of this technique)

B. *Cognitive Technique to Identificate Factual, Procedural, Contextual and Association Knowledge (TCI - CFPCA)*

This technique implements the second task the RE should develop during the problem-oriented analysis called Cognitive Analysis of Text Segments (ACST). The input of the technique are the Text Segments (ST) associated with User Spaces (EU) [14] that were obtained from the application of the technique for User's Discourse Segmentation (TS - DU) These segments are processed With The notion of Identifying Different types of knowledge (TC), Which are present in the "mental model" of the user based on personal experiences that occur in uncertain contexts [15][16]. The technique begins by Identifying Contextual Knowledge in the text segments (ST), and then continues with the identification of Factual Knowledge, Procedural Knowledge, and Knowledge Association. Finally, the RE Integrates Different types of information These with text segments, seeking to establish which kind of knowledge corresponds to segment text. Knowledge identified in each text segment (ST) are the output provided by this technique, which is summarized in Table 2.

TABLE II. COGNITIVE TECHNIQUE TO IDENTIFICATE FACTUAL KNOWLEDGE, PROCEDURAL KNOWLEDGE, CONTEXTUAL KNOWLEDGE AND ASSOCIATION KNOWLEDGE (TCI - CFPCA)

Technique:	Cognitive Identification of Factual Knowledge, Procedural Knowledge, Contextual Knowledge and Association Knowledge (TCI - CFPCA)
Input:	Text Segments (ST) associated to User Spaces (EU)
Output:	Types of Knowledge (TC) identified in Text Segment (ST) TC
Step 1.	Identification of Types of Knowledge (TC) in Text Segments (ST) (This step identifies the different types of knowledge: Contextual, Factual, Procedural and Association in the text segments (ST)). 1.1. Contextual Knowledge Identification in Text Segments (ST) 1.2. Factual Knowledge Identification in Text Segments (ST) 1.3. Procedural Knowledge Identification in Text Segments (ST) 1.4. Asociation Knowledge Identification in Text Segments (ST)
Step 2.	Integration among Text Segments and Types of Knowledge (In this second step is necessary to integrate text segments (ST) with the types of knowledge identified in the respective ST; for which, drawing up a table indicating the various TC contained in each of the ST. Table conecting ST with respective identified TC is the output of this technique)

C. *Technique for Building the Problem Space Diagram of User´s scenarios*

By means of this technique is implemented the third task to carry outby RE in the phase Problem-Oriented Analysis, called Technique for Building the Problem Space Diagram of User’s scenarios (CEPEU). For the implementation of the TCD - EPEU, RE uses as input the ST associated to EU obtained from the application of the technique TS - DU, and the TC identified in each of the ST obtained from the application of the technique TCI - CFPCA. To begin application of the TCD - EPEU, the RE proceeds to make use of the TC identified in each ST (leaving the association CT for Oriented Analysis Phase of the Product) to obtain the different elements that make up the EPEU, which are : Actors, Relationships, Attributes, Actions and Interactions. The RE then proceeds to identify the Contextual Framework Base (MCB) in which actors will unfold in the built EPEU (first diagram for this purpose). Finally, RE developes the remaining EPEU diagrams reflecting different realities provided by the respective ST. The EPEU diagrams are the output of this technique which is summarized in Table 3.

TABLE III. TECHNIQUE FOR BUILDING THE PROBLEM SPACE DIAGRAM OF USER´S SCENARIOS (TCD – EPEU)

Technique:	Building Problem Space Diagram of User´s Scenarios (TCD – EPEU)
Input:	ST associated to EU and ST-TC Table
Output:	EPEU Diagram
Step 1.	Use of TC for identifying EPEU elements (In this first step RE makes use of the respective TC for identifying the elements of EPEU diagrams for each of the associated ST. The completion of this step is accomplished through the following three substeps) 1.1. Use of Factual TC

	1.2. Use of Procedural TC 1.3. Use of Contextual TC
Step 2.	Building Diagram corrsponding to MCB (In this second step, the RE comes to build EPEU diagram for the MCB. For this, the ER analize ST that allows to contextualize the problem in the area in which occurs the reality described by the user (Department of marketing, Human Resources, etc). this diagram represents the central actors (leaving the incorporation of their attributes and actions for the next step) and relations between them, identified in substep 1.3. Therefore, for developing this step is carried out by the two following substeps) 2.1. Actors incorporation to MCB Diagram 2.2. Relation incorporation to MCB Diagram
Step 3.	Building remaining EPEU (In this third step, RE develope the remaining EPEU diagrams corresponding to the ST which continue to the MCB. For these diagrams, the RE uses EPEU diagram of the MCB and the various elements identified in substeps 1.1 and 1.2. Therefore, for each of the EPEU diagrams is carried out the following four substeps) 3.1. Incorporation of Actors to Diagram 3.1.1. Incorporation of Actors atributes to Diagram 3.1.2. Incorporation of Values of Actors atributes to Diagram 3.2. Incorporation of Relations to Diagram 3.3. Incorporation of Actions to Diagram 3.3.1. Incorporation of action attributes to Diagram 3.3.2. Incorporation of values of action attributes to Diagram 3.4. Incorporation of Interactions to Diagram 3.4.1. Incorporation of Interactions Atributes to Diagram 3.4.2. Incorporation of Values of Interactions Atributes to Diagram

V. TECHNIQUES FOR PRODUCT-ORIENTED ANALYSIS

This section presents techniques for Product Oriented-Analysis, which are: Technique for Construction of User´s Scenarios Diagram (TCD-EU) to implement the task of User´s Scenario Development(CEU) (section V.A), Technique for Refining User´s Scenarios Diagram (TRD-EU) to implement the task of User´s Scenarios Diagram refinement (REU) (section V.B) and Technique for Construction of Unified User´s Scenario Map Diagram (TCD-MUEU) for the implementation of the construction task Unified User´s Scenarios Map (CMUEU) (section V.C).

A. *Technique for Construction of User´s Scenario Diagram (TCD-EU)*

By means of this technique. RE implements the first task to be carried out in Product-Oriented Analysis phase called Construction of User´s Scenario (CEU). For the implementation of the TCD-EU, RE uses as input those ST associated to TC obtained from the application of the technique TS-DU, and each of the EPEU diagrams obtained from the application of the technique TCD-EPEU. To begin to apply the TCD-EU, RE proceeds to make use of the ST with the association CT that alows to get the functionalities of the problem defined by the user, as well as identification those EPEU actors that are necessary for the system to perform these functions. RE develops the blocks for Space Product of User Scenarios (EPrEU) for these EPEU [5] with these functionalities and EPEU diagrams in which the associated functionalities are identified,. Finally, the RE performs a process of association for the purpose of obtaining the linkages among elements of the blocks of EPrEU and EPEU, thus obtaining a single diagram for each EU constituted from both blocks. The diagrams corresponding to EU is the output product provided by this technique, which is summarized in Table IV.

TABLE IV. TECHNIQUE FOR CONSTRUCTION OF USER´S SCENARIO DIAGRAM (TCD-EU)

Tecnique:	Construction of User´s Scenario Diagram (TCD-EU)
Input:	ST with association TC (from Table ST-TC) and EPEU Diagram
Output:	EU Diagram
Step 1.	Using Association TC (In this first step, the ER uses the association CT for the construction of the EU. The completion of this step is performed by means of the following two substeps) 1.1. Funcionalities Identification 1.2. Actors Identification needed to carry out those functionalities
Step 2.	Construction of EPrEU diagram for each EPEU (In this second step, the ER uses obtained functionalities and EPEU diagrams which were identified associated functionalities, to build the Space Product of User´s Scenario Diagram (EPrEU) for these EPEU. Therefore, the EPrEU diagrams with the respective functionalities are the output of this step)
Step 3.	Linking elements of EPEU and EPrEU blocks for each EU. (In this third step, the ER proceeds to establish the "linkage" among the functionalities that make each of the EPrEU diagrams and actors of the corresponding EPEU, to perform these functions)

B. *Technique for Refinement User´s Scenarios Diagram (TRD-EU)*

Through this technique, RE implements the second task to carry out the Oriented Analysis Product phase called Refining User Scenarios (REU). The TRD-EU is applied jointly by the RE and User. The input products are the original User Speech (DU) and the Ue obtained in the previous technique. As a output result are obtained Refined User Scenarios (EUR). The application substep TRD-EU includes joint review (User and ER) of the original DU, which is carried out based on an analysis of consistency (to identify inconsistencies), which are classified into incompleteness and contradictions. These inconsistencies are solved to have a refined DU. From a refined DU, User and RE develop the validation and debugging of ST and CT for the purpose of purging inconsistencies in elements of the DU. Then, User and RE validates EU diagrams to obtain the EUR diagrams. Finally, User and ER develop final review of the EUR, if both granted pursuant to the obtained EUR the application of the technique finishes, if not, return to step 1 to begin to apply again. The diagrams corresponding to the EUR are the output product that provides this technique, which is summarized in Table V.

TABLE V. TECHNIQUE FOR REFINEMENT OF USER´S SCENARIOS DIAGRAM (TRD-EU)

Technique:	Refinement User´s Scenarios Diagram (TRD – EU)
Input:	User Speech (DU) and the UE Diagram
Output:	Refined User Scenarios (EUR)
Step 1.	Consistency Analysis of DU (In this first step, User and RE develop consistency analysis of DU based on the identification of incompleteness and inconsistencies to obtain a refined DU. This step is performed by means of the following three substeps) 1.1. Validation and Debuging of DU Incompleteness 1.2. Validation and Debuging of DU contradictions 1.3. Validation and Debuging of DU
Step 2.	Validation and Debuging of ST and TC (In this second step, user and RE develop validation and subsequent debugging of the ST and CT, since the inconsistencies identified in the DU in the substeps 1.1 and 1.2, are propagated to the ST and CT. Therefore, the refined ST and TC (STR and TCR) is the output product of this step).
Step 3.	Validation and Debuging of EU (In this third step, using DUR, STR and TCR, User and RE develop a validation and subsequent debugging the EU. In this way, it may be a case of having to add actors, change attributes, include interactions among actors; obtaining refined EU diagrams (EUR). Therefore, these EUR diagrams are the output product of this step).
Step 4.	Final Revision of EUR (In this fourth step, User and RE develop a final review of the EUR diagrams contrasting with EU diagrams that served as input to this technique jointly with the original DU. In case User and RE agree with the obtained EUR, these are the output product of this technique and the aplication of the technique finish, otherwise it returns to Step 1 and begin to apply the technique again)

Through this technique, the third and final task to carry out is implemented of the Product Oriented Analysis phase, called Construction of User´s Scenarios Unified Map Diagram (CMUEU). For the development of the TCD-MUEU, RE uses as input products each of ST associated to EU and EUR obtained by the application of the previous technique. As output result is obtained the User´s Scenarios Unified Map Diagram (MUEU). The MUEU diagram represents a space-time sequence about how the user understands the problem to be solved and the reality that fits the problem. The application of TCD sub step - MUEU includes a transitional analysis of User Scenarios (EU) through which it may be identified the triggers of User Scenarios (EU), which allow to identify the corresponding precedence relations among EU. From these triggers the RE s able to establish appropriate links among EU that lead to MUEU diagram. MUEU diagram is to the output product provided by this technique.

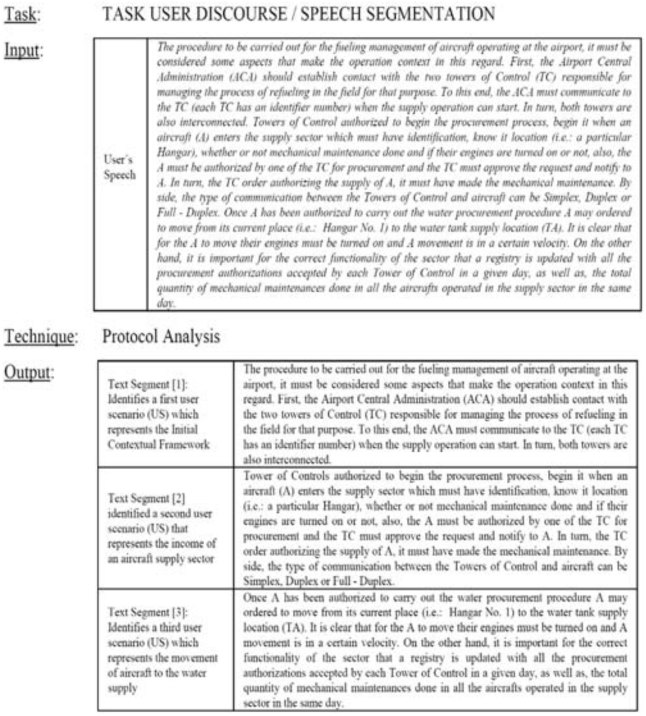
TABLE VI. TÉCHNIQUE OF CONSTRUCTION OF USER´S SCENARIOS UNIFIED MAP DIAGRAM (TCD-MUEU)

Technique:	Construction of User´s Scenarios Unified Map Diagram (TCD-MUEU)
Input:	Text Segments Associated to EU and EUR Diagrams
Output:	MUEU Diagram
Step 1.	Transition Analysis of EU (The RE identifies EU triggers present in ST associated to EU and reflected in the EUR. These triggers produce changes in EU occur in the body of the EU leading precedence relations among EU. The completion of this step is carried out through the following three substeps according to EU triggers types identified by RE) 1.1. Context Change Identification 1.2. Actors State Change Identification 1.3. New Actors Identification
Step 2.	Construction of MUEU Diagram (The RE proceeds to build MUEU diagram using EU wihc identifies Base Context Framework (Trigger type I). With triggers type II and III identified in step 1, build the chain of EU which will then lead to MUEU. MUEU Diagram with their respective EUR properly linked are the output product of this technique, and output of the process of requirements conceptualization)

VI. A CONCEPT PROOF OF THE PROPOSED PROCESS

This section presents the example of an Aircraft's Fuel Supply System as concept proof of the phase "Problem-Oriented Analysis". For each task is described inputs and outputs and the used techniques. There are described: the Task User Discourse / Speech Segmentation (Figure 5) the Task Cognitive Analysis of Text Segments (Figures 6. a, b, c) and the Task Construction of Problem Space based on User Scenarios (Figures 7. a, b, c), the Task Construction of Users Scenarios (Figure 8), the Task Refinement of Users Scenarios (Figure 9), and the Task Construction of Unified-Map of User Scenarios (Figure 10).

The results of cognitive techniques that have been applied to identify factual knowledge, procedural knowledge, and contextual knowledge and association knowledge with the Text Segments are shown in Figures 6.a, 6.b and 6.c. The results of having applied the technique of construction of diagram of problem-space based on user scenarios from templates of factual knowledge, procedural knowledge, contextual knowledge and association knowledge obtained are shown in Figures 7.a, 7.b and 7.c. In the Task Construction of Users Scenarios, shown in Figure 8, the requirement engineer proceeds to the building the User Scenario with the building of the blocks corresponding to the Product Space for those scenarios in which the functionalities associated to the space problem are identified. In order to perform this task, the requirement engineer has as input products two elements: the Problem Space of the Users Scenarios and the Text Segments with Knowledge Types of Association. The result output are the Users Scenarios which are represented by the diagrams that have two blocks corresponding to the Problem Space y al Product Space; which are linked by the arrows between the element and the functionality.



Task: TASK USER DISCOURSE / SPEECH SEGMENTATION

Input:

User´s Speech	*The procedure to be carried out for the fueling management of aircraft operating at the airport, it must be considered some aspects that make the operation context in this regard. First, the Airport Central Administration (ACA) should establish contact with the two towers of Control (TC) responsible for managing the process of refueling in the field for that purpose. To this end, the ACA must communicate to the TC (each TC has an identifier number) when the supply operation can start. In turn, both towers are also interconnected. Towers of Control authorized to begin the procurement process, begin it when an aircraft (A) enters the supply sector which must have identification, know it location (i.e.: a particular Hangar), whether or not mechanical maintenance done and if their engines are turned on or not, also, the A must be authorized by one of the TC for procurement and the TC must approve the request and notify to A. In turn, the TC order authorizing the supply of A, it must have made the mechanical maintenance. By side, the type of communication between the Towers of Control and aircraft can be Simplex, Duplex or Full - Duplex. Once A has been authorized to carry out the water procurement procedure A may ordered to move from its current place (i.e.: Hangar No. 1) to the water tank supply location (TA). It is clear that for the A to move their engines must be turned on and A movement is in a certain velocity. On the other hand, it is important for the correct functionality of the sector that a registry is updated with all the procurement authorizations accepted by each Tower of Control in a given day, as well as, the total quantity of mechanical maintenances done in all the aircrafts operated in the supply sector in the same day.*

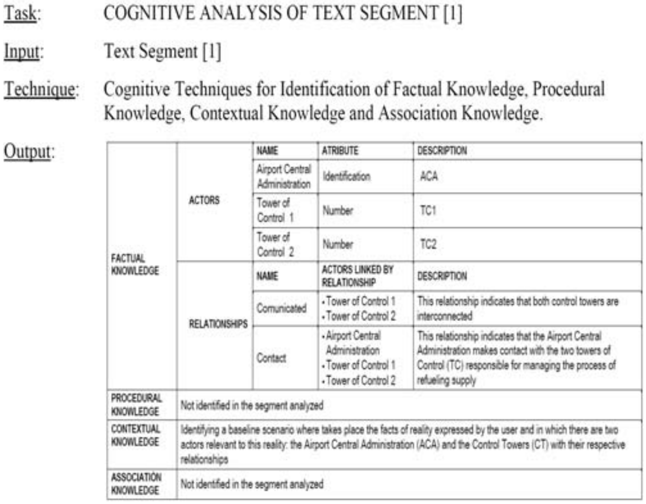
Technique: Protocol Analysis

Output:

Text Segment [1]: Identifies a first user scenario (US) which represents the Initial Contextual Framework	The procedure to be carried out for the fueling management of aircraft operating at the airport, it must be considered some aspects that make the operation context in this regard. First, the Airport Central Administration (ACA) should establish contact with the two towers of Control (TC) responsible for managing the process of refueling in the field for that purpose. To this end, the ACA must communicate to the TC (each TC has an identifier number) when the supply operation can start. In turn, both towers are also interconnected.
Text Segment [2] identified a second user scenario (US) that represents the income of an aircraft supply sector	Tower of Controls authorized to begin the procurement process, begin it when an aircraft (A) enters the supply sector which must have identification, know it location (i.e.: a particular Hangar), whether or not mechanical maintenance done and if their engines are turned on or not, also, the A must be authorized by one of the TC for procurement and the TC must approve the request and notify to A. In turn, the TC order authorizing the supply of A, it must have made the mechanical maintenance. By side, the type of communication between the Towers of Control and aircraft can be Simplex, Duplex or Full - Duplex.
Text Segment [3]: Identifies a third user scenario (US) which represents the movement of aircraft to the water supply	Once A has been authorized to carry out the water procurement procedure A may ordered to move from its current place (i.e.: Hangar No. 1) to the water tank supply location (TA). It is clear that for the A to move their engines must be turned on and A movement is in a certain velocity. On the other hand, it is important for the correct functionality of the sector that a registry is updated with all the procurement authorizations accepted by each Tower of Control in a given day, as well as, the total quantity of mechanical maintenances done in all the aircrafts operated in the supply sector in the same day.

Figure 5. Task: User Discourse / Speech Segmentation

Thus, this representation shows the "existing linkage" between the required functionalities for the software product and the el ements of the problem space that are necessary to process the fu nctionality. In this case, the Text Segment [3] is the only one identified as knowledge of association. This knowledge of association allows defining two functionalities which include all the p roduct space (Registry of the procurement authorizations accepted by the Tower of Control in one given day and T otal quantity of mechanical maintenances performed in all the aircrafts in one given day).



Task: COGNITIVE ANALYSIS OF TEXT SEGMENT [1]

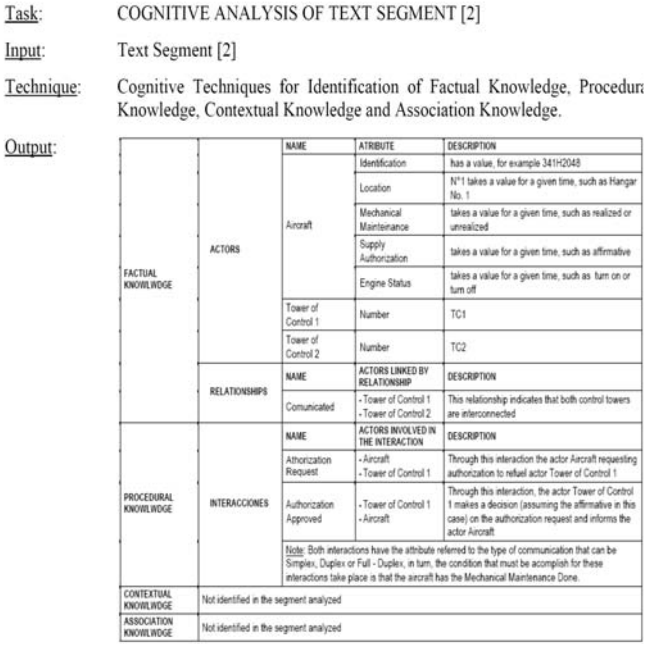
Input: Text Segment [1]

Technique: Cognitive Techniques for Identification of Factual Knowledge, Procedural Knowledge, Contextual Knowledge and Association Knowledge.

Output:

FACTUAL KNOWLEDGE	ACTORS	NAME	ATRIBUTE	DESCRIPTION
		Airport Central Administration	Identification	ACA
		Tower of Control 1	Number	TC1
		Tower of Control 2	Number	TC2
	RELATIONSHIPS	NAME	ACTORS LINKED BY RELATIONSHIP	DESCRIPTION
		Comunicated	• Tower of Control 1 • Tower of Control 2	This relationship indicates that both control towers are interconnected
		Contact	• Airport Central Administration • Tower of Control 1 • Tower of Control 2	This relationship indicates that the Airport Central Administration makes contact with the two towers of Control (TC) responsible for managing the process of refueling supply
PROCEDURAL KNOWLEDGE	Not identified in the segment analyzed			
CONTEXTUAL KNOWLEDGE	Identifying a baseline scenario where takes place the facts of reality expressed by the user and in which there are two actors relevant to this reality: the Airport Central Administration (ACA) and the Control Towers (CT) with their respective relationships			
ASSOCIATION KNOWLEDGE	Not identified in the segment analyzed			

Figure 6.a. Task: Cognitive Analysis of Text Segment 1.



Task: COGNITIVE ANALYSIS OF TEXT SEGMENT [2]

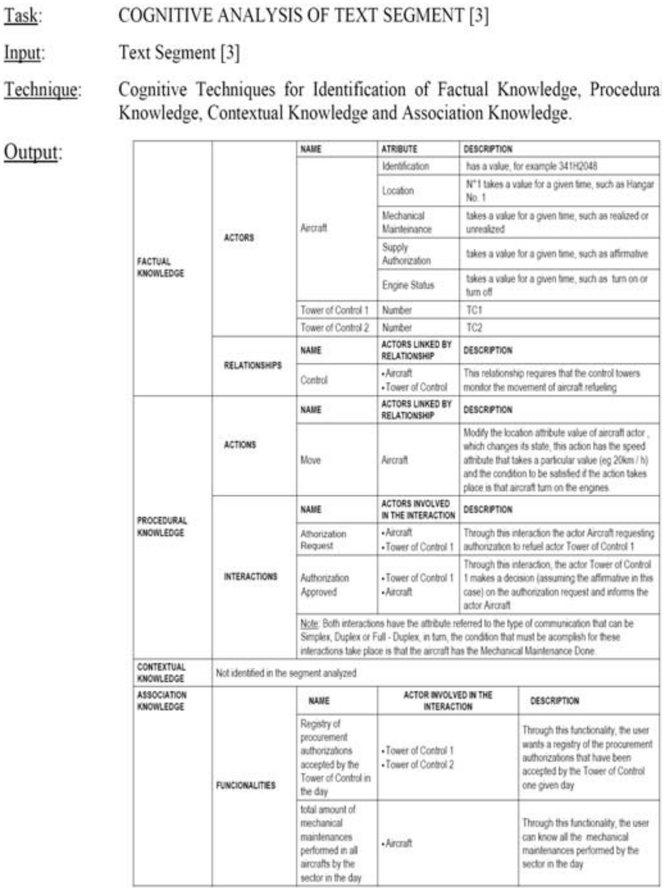
Input: Text Segment [2]

Technique: Cognitive Techniques for Identification of Factual Knowledge, Procedura Knowledge, Contextual Knowledge and Association Knowledge.

Output:

FACTUAL KNOWLWDGE	ACTORS	NAME	ATRIBUTE	DESCRIPTION
		Aircraft	Identification	has a value, for example 341H2048
			Location	N°1 takes a value for a given time, such as Hangar No. 1
			Mechanical Mainteinance	takes a value for a given time, such as realized or unrealized
			Supply Authorization	takes a value for a given time, such as affirmative
			Engine Status	takes a value for a given time, such as turn on or turn off
		Tower of Control 1	Number	TC1
		Tower of Control 2	Number	TC2
	RELATIONSHIPS	NAME	ACTORS LINKED BY RELATIONSHIP	DESCRIPTION
		Comunicated	- Tower of Control 1 - Tower of Control 2	This relationship indicates that both control towers are interconnected
PROCEDURAL KNOWLWDGE	INTERACCIONES	NAME	ACTORS INVOLVED IN THE INTERACTION	DESCRIPTION
		Athorization Request	- Aircraft - Tower of Control 1	Through this interaction the actor Aircraft requesting authorization to refuel actor Tower of Control 1
		Authorization Approved	- Tower of Control 1 - Aircraft	Through this interaction, the actor Tower of Control 1 makes a decision (assuming the affirmative in this case) on the authorization request and informs the actor Aircraft
		Note: Both interactions have the attribute referred to the type of communication that can be Simplex, Duplex or Full - Duplex, in turn, the condition that must be acomplish for these interactions take place is that the aircraft has the Mechanical Maintenance Done.		
CONTEXTUAL KNOWLWDGE	Not identified in the segment analyzed			
ASSOCIATION KNOWLWDGE	Not identified in the segment analyzed			

Figure 6.b. Task: Cognitive Analysis of Text Segment 2.



Task: COGNITIVE ANALYSIS OF TEXT SEGMENT [3]

Input: Text Segment [3]

Technique: Cognitive Techniques for Identification of Factual Knowledge, Procedura Knowledge, Contextual Knowledge and Association Knowledge.

Output:

FACTUAL KNOWLEDGE	ACTORS	NAME	ATRIBUTE	DESCRIPTION
		Aircraft	Identification	has a value, for example 341H2048
			Location	N°1 takes a value for a given time, such as Hangar No. 1
			Mechanical Mainteinance	takes a value for a given time, such as realized or unrealized
			Supply Authorization	takes a value for a given time, such as affirmative
			Engine Status	takes a value for a given time, such as turn on or turn off
		Tower of Control 1	Number	TC1
		Tower of Control 2	Number	TC2
	RELATIONSHIPS	NAME	ACTORS LINKED BY RELATIONSHIP	DESCRIPTION
		Control	• Aircraft • Tower of Control	This relationship requires that the control towers monitor the movement of aircraft refueling
PROCEDURAL KNOWLEDGE	ACTIONS	NAME	ACTORS LINKED BY RELATIONSHIP	DESCRIPTION
		Move	Aircraft	Modify the location attribute value of aircraft actor , which changes its state, this action has the speed attribute that takes a particular value (eg 20km / h) and the condition to be satisfied if the action takes place is that aircraft turn on the engines
	INTERACTIONS	NAME	ACTORS INVOLVED IN THE INTERACTION	DESCRIPTION
		Athorization Request	• Aircraft • Tower of Control 1	Through this interaction the actor Aircraft requesting authorization to refuel actor Tower of Control 1
		Authorization Approved	• Tower of Control 1 • Aircraft	Through this interaction, the actor Tower of Control 1 makes a decision (assuming the affirmative in this case) on the authorization request and informs the actor Aircraft
		Note: Both interactions have the attribute referred to the type of communication that can be Simplex, Duplex or Full - Duplex, in turn, the condition that must be acomplish for these interactions take place is that the aircraft has the Mechanical Maintenance Done.		
CONTEXTUAL KNOWLEDGE	Not identified in the segment analyzed			
ASSOCIATION KNOWLEDGE	FUNCIONALITIES	NAME	ACTOR INVOLVED IN THE INTERACTION	DESCRIPTION
		Registry of procurement authorizations accepted by the Tower of Control in the day	• Tower of Control 1 • Tower of Control 2	Through this functionality, the user wants a registry of the procurement authorizations that have been accepted by the Tower of Control one given day
		total amount of mechanical maintenances performed in all aircrafts by the sector in the day	• Aircraft	Through this functionality, the user can know all the mechanical maintenances performed by the sector in the day

Figure 6.c. Task: Cognitive Analysis of Text Segment 3.

Task: CONSTRUCTION OF PROBLEM SPACE BASED ON USER SCENARIOS (Text Segment [1])

Input:
- Text Segment [1]
- Template of Factual Knowledge, Procedural Knowledge, Contextual Knowledge and Association Knowledge obtained from Text Segment [1]

Technique: Technique of Construction of Diagram of Problem-Space Based on User Scenarios

Output:

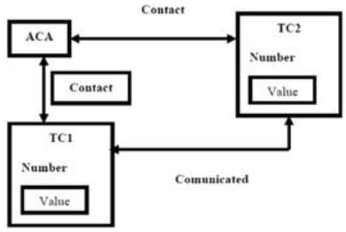


Figure 7.a. Task: Construction of Diagram of Problem-Space Based on User Scenarios (Text Segment 1)

Task: CONSTRUCTION OF PROBLEM SPACE BASED ON USER SCENARIOS (Text Segment [2])

Input:
- Text Segment [2]
- Template of Factual Knowledge, Procedural Knowledge, Contextual Knowledge and Association Knowledge obtained from Text Segment [2]

Technique: Technique of Construction of Diagram of Problem-Space Based on User Scenarios

Output:

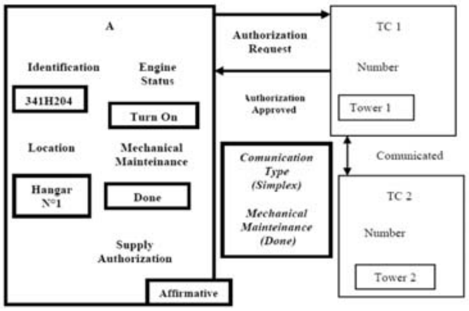


Figure 7.b. Task: Construction of Diagram of Problem-Space Based on User Scenarios (Text Segment 2)

Task: CONSTRUCTION OF PROBLEM SPACE BASED ON USER SCENARIOS (Text Segment [3])

Input:
- Text Segment [3]
- Template of Factual Knowledge, Procedural Knowledge, Contextual Knowledge and Association Knowledge obtained from Text Segment [3]

Technique: Technique of Construction of Diagram of Problem-Space Based on User Scenarios

Output:

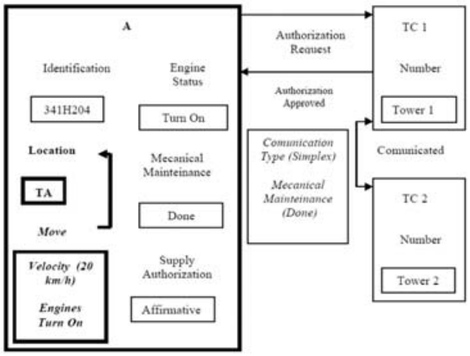


Figure 7.c. Task: Construction of Diagram of Problem-Space Based on User Scenarios (Text Segment 3)

Task: CONSTRUCTION OF USER SCENARIOS (Text Segment [3])

Input:
- Text Segment [3]
- Problem Space of User Scenario corresponding to Text Segment [3]

Technique: Technique of Construction of User Scenarios

Output:

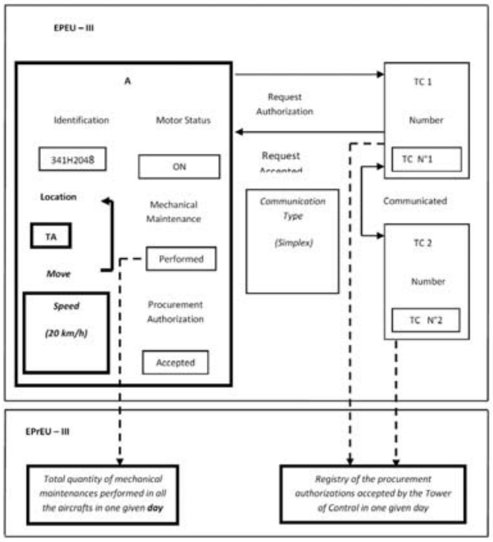

EPEU – III
A
Identification
341H2048
Motor Status
ON
Location
TA
Move
Speed (20 km/h)
Mechanical Maintenance
Performed
Procurement Authorization
Accepted
Request Authorization
Request Accepted
Communication Type (Simplex)
TC 1
Number
TC N°1
Communicated
TC 2
Number
TC N°2
EPrEU – III
Total quantity of mechanical maintenances performed in all the aircrafts in one given day
Registry of the procurement authorizations accepted by the Tower of Control in one given day

Figure 8. Task: Construction of Users Scenarios

In the task Refinement of Users Scenarios, shown in Figure 9, the user an d the requirement engineer interact together in order to acquire scenarios free of errors and inconsistent. These “debugged” scenarios are called Refined Users Scenarios.

Task: REFINEMENT OF USER SCENARIOS (Text Segment [3])

Input:
- User’s Discourse / Speech
- User Scenario III (US – III)

Technique: Technique of Construction of Refined User Scenarios

Output:

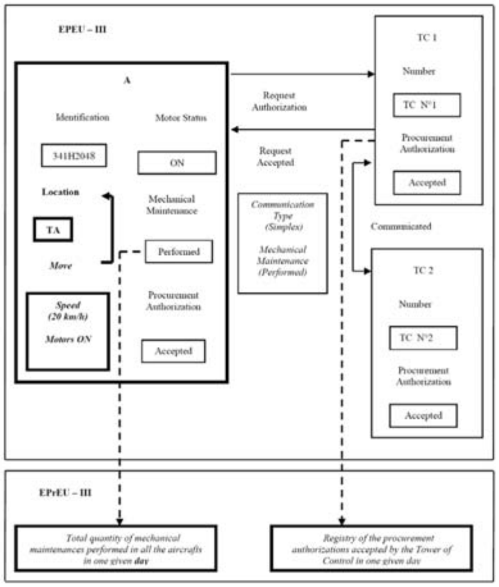
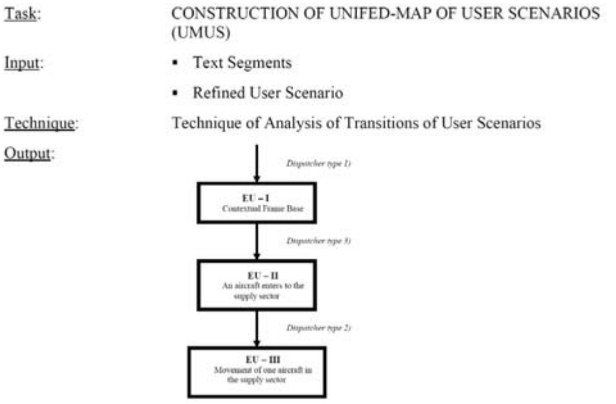

EPEU – III
A
Identification
341H2048
Motor Status
ON
Location
TA
Move
Speed (20 km/h)
Motors ON
Mechanical Maintenance
Performed
Procurement Authorization
Accepted
Request Authorization
Request Accepted
Communication Type (Simplex)
Mechanical Maintenance (Performed)
TC 1
Number
TC N°1
Procurement Authorization
Accepted
Communicated
TC 2
Number
TC N°2
Procurement Authorization
Accepted
EPrEU – III
Total quantity of mechanical maintenances performed in all the aircrafts in one given day
Registry of the procurement authorizations accepted by the Tower of Control in one given day

Figure 9. Task: Refinement of Users Scenarios.

In order to perform this task, the requirement engineer has as input products two elements: the Users Scenarios and the original User’s Discourse / Speech, getting as output the RUS complying the user’s requirements. For this study case, it is reviewed the User’s Discourse / Sp eech and the Users Scenarios in the paragraph of Text Segment [3]: “that a registry is updated with all the procurement authorizations accepted by each Tower of Control in a given day”. that defines the functionality “Registry of the procurement authorizations accepted by the Tower of Control in one given day”. A problem is detected in the User Scenario and it is necessary to add in each Tower of Control actor the attribute Procurement Authorization with the value Accepted, because these type of authorizations are the only one interested to be r egistered by this functionality. As a result, the User Scenario that is refined is the third one. In the t ask Construction of Unified-Map of User Scenarios, shown in Figure 10, the requirement engineer works on the construction of the Uni fied-Map of User Scenarios, which allows documenting the “temporal order” in which the scenarios are performed. For this task, the requirement engineer has as input pr oducts the Text Segments and the Refined Users Scenarios, getting as a result the UMUS. The used technique is “Analysis of Transitions of Users Scenarios”, which allows identifying in the T ext Segments the elements called “scenario dispatchers” from which the transition is performed. The dispatcher can have three types: 1) context changes in the user discourse, 2) state change in the actors of the scenario (modifying the attribute values) and 3) adding actors to the scenario. In this case, the analysis of Text Segment [1] indica tes that the US-1 is performed by a dispatcher type 1), because it is related to a con textual base frame; the analysis of Text Segment [2] indicates that the US – 2 is performed by a dispatcher type 3) , because the aircraft actor is added; and finally the analysis of Text Segment [3] indicates that the US-3 is performed by a dispa tcher type 2), because a change in the state actor air craft is done when the location attribute is set from Hangar N°1) to tank supply location. As a result.



Task: CONSTRUCTION OF UNIFED-MAP OF USER SCENARIOS (UMUS)

Input:
- Text Segments
- Refined User Scenario

Technique: Technique of Analysis of Transitions of User Scenarios

Output:

Figure 10. Task: Construction of Unified-Map of User Scenarios.

VII. CONCLUSIONS

The main contribution of this paper is to present a methodical process called Requirements Conceptualization, which is divided into two phases, called the Problem Oriented Analysis and Product -Oriented Analysis, and whose main objective is to structure and characterize the mass of information from elicitation activity within the discourse (speech) of the user. This paper presents two proofs of concept for a specific case about two phases of this process. The first phase has as input the text associated to the User´s Speech and as an output the Dia gram of Problem-Space Based on User Scenarios. To carry out the tasks it has been ada pted some techniques and developed another ones; they are: Protocol Analysis, Cognitive Techniques for Ident ification of Factual Knowledge, Procedural Knowledge, Contextual Knowledge and Association Knowledge, and Technique of Construction of Diagram of Problem-Space Based on User Scenarios. The structuration of the Phase of Problem Oriented Analysis into the tasks: U ser Discourse / Speech Segmentation, Cognitive Analysis of Text Segments and Construction of Problem Space based on U ser Scenarios; allows the requirements engineer to carry out a systematic analysis of user's speech to reach gradually an integrated representation of the fundamental elements of it. The nex t research steps are: [a] develop and execute an experiment to validate empirically the process of requirements conceptualization introduced and [b] to focus on implementing of high quality conceptual models.

ACKNOWLEDGMENT

The research results reported in this article have been partially funded by the Research Project 33A105, Department of Production and Technology, National University of La nus (Argentine) and by the grants TIN2008-00555 and HD2008-0048 of the Spanish Ministry of Science and Innovation.

REFERENCES

[1] Sommerville, I. 2005. Ingeniería de Software, Addison-Wesley.

[2] Chatzoglou P., S oteriou A. 1999. A DEA framework to assess the efficiency of the software requirements capture and analysis process. Decision-Sciences. 30(2): 503-31.

[3] Van der Vos, B., Gulla, J., Van de Riet, R., 1995. Verification of Conceptual Models based n Linguistic Knowledge. NLDB 1995

[4] Loucopoulos, P., Karakostas, V. 1995. System Req uirements Engineering; McGraw-Hill,

[5] Davis, A. 1993. Software Requirements: Objects, Functions and States; Prentice-Hall International.

[6] Faulk, S. 1997. Software Requirements: A Tutorial; *In Software Engineering, IEEE Computer Society Press*, pp 82-101.

[7] Sutcliffe, A., Maiden, N. 1992. Analysing the Novice Analyst: Cognitive Models in Software Engineering; Intl. Jl of Man-Machine Studies, 36(5).

[8] Wieringa, R. 1995. Requirements Engineering: Fram eworks for Understanding; John Wiley.

[9] Jalote, P. 1997. An Integrated Approach to So ftware Engineering**;** Springer-Verlag.

[10] Juristo, N., Moreno, A. 20 00. Introductory paper: Reflections on Conceptual Modeling; Data and Knowledge Engineering, vol 33.

[11] Moreno, A., 1999. Tesis Doctoral Universidad Politécnica de Madrid,

[12] Chen, P. 1990. Entity-relationship Approach to Data Modeling. In Systemand Software Requirements Engineering, Thayer RH, Dorfman M (eds). IEEE. Computer Society Press.

[13] García Martínez, R. y Britos, P. 2004. Ingeniería de Sistemas Expertos. Editorial Nueva Librería. ISBN 987-1104-15-4.

[14] Carroll, J. 1995. Introdu ction: The Scenario Perspective on Sys tem Development, en "Scenario-Based Design: Envisioning Work and Technology in System Development", John Wiley & Sons.

[15] Anderson, J. 2006. Cognitive Psychology and Its Implications. Watson Guptill Publications.

[16] Robertson S. 2002. Project Sociology: Identifying and involving the stakeholders, ICFAI University Press

Using Empirical Studies to evaluate the REMO Requirement Elicitation Technique

Sérgio Roberto Costa Vieira[1,2]
[1]Fundação Centro de Análise, Pesquisa e Inovação Tecnológica (FUCAPI)
Av. Danilo Areosa, 381 – Distrito Industrial
Manaus-AM, Brasil
sergio.vieira@fucapi.br

Davi Viana, Rogério do Nascimento, Tayana Conte
[2]Grupo de Pesquisa Usabilidade e Engenharia de Software (USES) – Universidade Federal do Amazonas (UFAM)
Av. Rodrigo Otávio, 3000 – Coroado I
Manaus-AM, Brasil
{davi.viana,rogerio,tayana}@icomp.ufam.edu.br

Abstract — **One of the most common problems regarding software quality is the software's incapability of offering effective and efficient support to business operations. A possible motive for this lack of support is the inconsistency of the requirements related to the business needs. In order to avoid this problem, the use of business process modeling during the requirements elicitation must be considered. Therefore, this work proposes a technique to support extracting the requirements from business process diagrams: the REMO (Requirements Elicitation oriented by business process MOdeling) technique. Furthermore, we present the results of two controlled experiments aimed at measuring the effectiveness and adequacy of the produced requirements in comparison to a traditional approach. The results indicated that compared to the traditional approach our technique was better regarding adequacy indicator, and as good as regarding the effectiveness indicator.**

Keywords requirements elicitation; business process modeling; empirical study

I. Introduction

One of the first activities to be executed in software development is the requirements elicitation activity. This activity focuses on using mechanisms in order to understand which needs must be attended by the developed software [11]. According to Monsalve et al. [3], the software development depends on the quality of the requirements elicitation activities. Often, the elicitation is criticized by obtaining incomplete and inconsistent requirements, since the users don't have a concrete idea of what they want [1].

According to Martins and Daltrini [11], in order to understand the users' needs, it is necessary to understand the activities these users execute considering their work context. Martinez et al. [10] state that the focus of business processes can increase the conformity of the software regarding its users' needs. This approach must consider the comprehension of the organizational environment, therefore, focusing on the business processes to carry out the software development.

The observation of the business processes during the software development as a relevant factor for the requirements elicitation, allows to clearly understand the business domain in which the software will work [5]. According to Cardoso et al. [4], business process modeling is a mechanism that can facilitate understanding of how business processes work in a company. Business process models are instruments to identify problems and improvement opportunities within a company. They help to understand the structure and behavior of a company. Furthermore, these models are very useful for increasing the comprehension about business environment by system analysts [8].

The non adoption of business processes modeling, as a mechanism to elicit software requirements, can end up generating software with inconsistent and incomplete requirements regarding the real needs of the business. These types of requirements can make the developed software unavailable to meet the business needs for which it is created [15].

We have developed a requirement elicitation technique that is oriented by business processes modeling called REMO (Requirements Elicitation oriented by business process MOdeling). This technique aims to a id system analysts, who are responsible for the software development, in identifying the functional and non-functional requirements and business rules using business processes diagrams.

The goal of this paper is to describe both the REMO technique and the results from two empirical studies evaluating its feasibility. These studies were conducted in o rder to measure the effectiveness regarding requirements identification using business processes models, with or without the use of the technique, compared with a traditional approach [4]. Furthermore, we have also observed the adequacy of the evaluated requirements by classifying them into: (a) appropriate requirements, and (b) inappropriate requirements ("false positives"), regarding the context of the business processes.

This paper is organized as follows. Section II describes the related works within the context of the proposed technique In Section III we present an initial version of the R EMO technique. Section IV describes how we carried out the first empirical study and its results. Section V discusses how we evolved the technique based on the results of this first study. Section VI presents the second empirical study in order to evaluate the new version of the technique. Finally, the conclusions and future steps of this research are presented in Section VII.

II. Related Work

Business processes modeling consists of the formalization of an organizations' processes activities, capturing the context in which these are executed. According to Carvalho *et al.* [1], the knowledge and understanding of business processes are extremely important to ensure that requirements are appropriate to the real needs of the organization. In order to obtain a complete comprehension of the business process, it is recommended to use strategies aiming conformity of the software requirements with the business needs. These strategies are known as requirements elicitation approaches oriented by business processes models. We will now describe the approaches in which the initial proposal of the technique was based.

Santander and Castro [12] presented guidelines that allow the development of use cases from organizational business models described through the i* framework [16]. These guidelines are used to identify the scenarios that will be described by the use cases, by using the organizational modeling as starting point through the use of the i* framework.

An approach that uses goal oriented analysis to identify use cases through the i* framework is pr esented in [7]. This approach identifies use cases from business processes modeling. Initially, a goal tree must be used to refine the business model in order to identify the types of goals and their actors. After that, the approach suggests to start modeling use cases in UML (Unified Modeling Language).

The approach presented in [6] uses user case diagrams and domain classes diagrams to t ransform business processes models into requirements models. This approach is supported by the RAPIDS (Rules And Process for the Development of Information Systems) tool. This approach has two stages in order to generate use cases from business processes models: (a) transformation of the terms of a determined business into domain classes; and (b) transformation of processes into use cases by using heuristics.

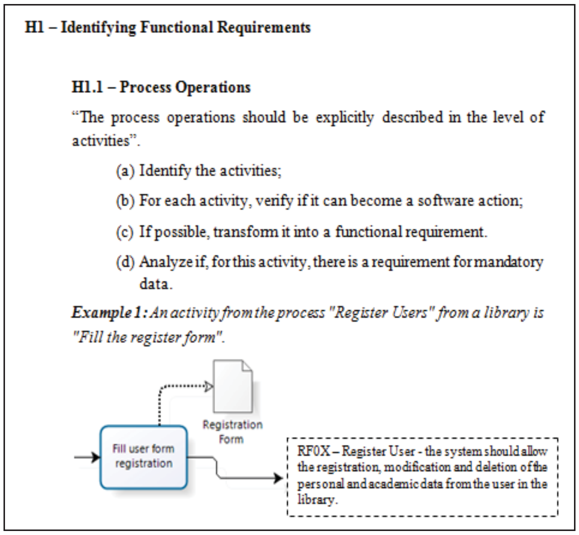
The presented approaches extract user cases from business processes models. However, before modeling use cases, it is important to produce a document with functional and non-functional requirements and business rules. Quality models (e.g. CMMI [13]) point out that it is necessary to specify requirements before describing components and operational scenarios (through use cases or other forms of specification). This scenario has motivated one of our research's goals: to define a technique tailored to support extracting requirements from business process models. In the next section we will describe the first version of the proposed technique.

III. The REMO Requirements Elicitation Technique

The REMO (Requirements Elicitation oriented by business process MOdeling) is a requirements elicitation technique that guides the elicitation process of the developed software by using business processes modeling.

The REMO technique uses a set of heuristics to extract the software requirements from the business processes diagrams modeled in BPMN (Business Process Modeling Notation). Its main goal is to aid the analysts in the identification of the system requirements from business processes models. We based this technique on the set of core elements of the BPMN notation. This notation allows the identification of: the activities, the dependencies controls, and the task flows. The BPMN notation is a pattern to guide process modeling suggested by the OMG (*Object Management Group*) [9].

Business process modeling is used as an entrance subsidy for the technique's application. The analyst applies the REMO heuristics in order to obtain the software requirements. Following the heuristics, the analyst identifies the actions represented by the elements of the BPMN notation and transforms them into a software requirement. In the initial version of the REMO technique, we elaborated 8 h euristics, divided in heuristics to identify functional requirements and heuristics to extract non-functional requirements. Fig. 1 shows an extract from the initial version. A detailed description of the first version of the REMO technique can be found in [14].



H1 – Identifying Functional Requirements

H1.1 – Process Operations

"The process operations should be explicitly described in the level of activities".

(a) Identify the activities;

(b) For each activity, verify if it can become a software action;

(c) If possible, transform it into a functional requirement.

(d) Analyze if, for this activity, there is a requirement for mandatory data.

Example 1: *An activity from the process "Register Users" from a library is "Fill the register form".*

RF0X – Register User - the system should allow the registration, modification and deletion of the personal and academic data from the user in the library.

Figure 1. Extract of the first version of the REMO technique.

The heuristics were based in actions contained in processes diagrams. They aid the identification of requirements from: processes operations; documents used within the processes activities; decisions conditions within the processes flow; dependencies between activities, activities with interruption flow; activities deriving messages/communications; activities possessing restrictions; and, activities with quality attributes. In the next section we present the planning and the execution of the first empirical study.

IV. First Empirical Study

The first empirical study was carried out in or der to evaluate the feasibility of the technique. This evaluation was performed from indicators of effectiveness and adequacy, as described below:

- **Effectiveness:** ratio between the number of real identified requirements and the number of total known requirements obtained from the business processes models.
- **Adequacy:** percentage of adequate pointed requirements regarding the business processes context. After identifying the requirements using the business models, we categorized them in real or "false positives". A

requirement was considered “false positive” when it was wrong (e.g. a misunderstanding about the business processes) or in the case that it could not have been extracted through business processes models. This indicator is considered relevant because it prevents the specification of requirements that are not in conformity with the identified needs extracted from the business processes.

The experiment goal using the GQM (Goal/ Question/Metric) paradigm [2] is presented in Table I.

TABLE I. GOAL OF THE 1ST. EMPIRICAL STUDY

Analyze	The REMO technique
For the purpose of	Characterizing
With respect to	Effectiveness and adequacy of the identified requirements using the business processes
From the point of view	Software Engineering Researchers
In the context of	A requirements elicitation from a w eb based system having as basis business processes models.

Hypotheses: the study was planned and conducted in order to test the following hypotheses (null and alternative, respectively):

H01: There is no difference in t erms of effectiveness in using the REMO technique to elicit requirements regarding business processes.

HA1: The REMO technique presented a difference in the effectiveness indicator, when compared to a traditional approach.

H02: There is no difference in terms of adequacy of the identified requirements when using the REMO technique and a traditional approach.

HA2: The REMO technique presented different results regarding the adequacy of the identified requirements, when compared to a traditional approach.

Context: the study was carried out in M ay 2011 with undergraduate students from senior-level undergraduate Computer Science and Information System courses. These students were attending Analysis and Design class at the Federal University of Amazonas (UFAM). All the subjects had tutorials about Business Process Modeling and Requirements Elicitation. As the object study we used the business processes modeling of part of the processes. This part described the management activities of the discipline “Final Project”.

Preparation: 30 students played the role of system analysts and signed a consent form. The subjects also filled in a characterization questionnaire, with questions regarding their practical experience. This questionnaire allowed us to identify that, despite being undergraduate students; many of them had experience in system analysis in the industry. An ordinal scale was used to measure their experience: Low, Medium and High. Three subjects were classified with High experience (having participated in f ive or more development projects), sixteen subjects were classified with medium experience, and the remaining eleven with low experience (with theoretical knowledge and no previous experience in industrial system analysis). The categorization results are shown in Table II.

Execution: the subjects received: (a) an execution guide with tasks, (b) the business context document containing the BPMN processes modeling, and (c) requirements register spreadsheet. Furthermore, the group that used the REMO technique received the additional documentation: the technique’s heuristics document. The group that did not use the REMO technique used a traditional approach to identify the requirements, using the knowledge used in the previous trainings and the processes modeling, as suggested by [4].

The study was carried out by each group in different days. The first day Group 1 used the traditional approach and the second day Group 2 used REMO technique. Each group had 120 minutes to carry out the requirements elicitation based on the business processes models. We collected 13 and 15 spreadsheets from the traditional approach and the REMO technique approach respectively. Readers must take note that 2 students did not show up the first day, leaving the groups unbalanced. We carried out an outlier analysis in order to balance the groups. After finishing the study, we created a unique requirements list, in order to discriminate the requirements. The analyst responsible for the development of a system for automating final projects control carried out the discrimination process. The discrimination meeting was carried out to identify which requirements were considered inadequate (e.g. misunderstandings about the business processes).

Results and Quantitative Analysis: Table II p resents the results of the requirements registers per group of subjects. Readers must take note that we used a total of 29 known requirements from the processes models, in order to calculate the effectiveness degree of the requirements.

TABLE II. REQUIREMENS RESULTS PER GROUP OF SUBJECTS

Group 1 (TRADITIONAL)						Group 2 (REMO)					
Exp	Sub	IIR	FP	EI (%)	AI (%)	Exp	Sub	IIR\	FP	EI (%)	AI (%)
H	A09	14	5	31.03	64.29	H	R03	21	12	31.03	42.86
	A13	21	10	37.93	52.38	M	R04	21	9	41.38	57.14
M	A03	19	8	37.93	57.89		R06	24	9	51.72	62.50
	A04	32	17	51.72	46.88		R07	14	4	34.48	71.43
	A08	32	19	44.83	40.63		R08	21	11	34.48	47.62
	A10	26	13	44.83	50.00		R10	18	5	44.83	72.22
	A14	21	6	51.72	71.43		R11	19	8	37.93	57.89
	A15	28	10	62.07	64.29		R12	21	8	44.83	61.90
L	A01	28	18	34.48	35.71	L	R02	12	4	27.59	66.67
	A05	7	6	3.45	14.29		R05	19	5	48.28	73.68
	A07	12	6	20.69	50.00		R09	23	8	51.72	65.22
	A11	29	17	41.38	41.38		R13	38	20	62.07	47.37
	A12	23	11	41.38	52.17		R14	24	17	24.14	29.17

Legend: H – High; M – Medium; L – Low; Exp – Experience; Sub – Subjects; IIR –Initially Identified Requirements; FP – False Positives; AI – Adequacy Indicator; EI –Effectiveness Indicator.

In order to validate these data we used the Mann-Whitney statistic method, which is supported by the SPSS Sta tistics v17.0 [1] tool, and the boxplots analysis. Fig.2 shows the distribution of effectiveness per subject, per technique. The boxplots graph shows only a slight difference between the subjects who used the technique and those who did not use it. When we compared the two samples using the Mann-Whitney

[1] http://www-01.ibm.com/software/analytics/spss/

test, we did not find any significant differences between the two groups (p=0.857 and α = 0.05). This result suggests that both techniques are similar regarding the effectiveness of the requirements. These results support the null hypothesis H01, that there are no differences in effectiveness between techniques.

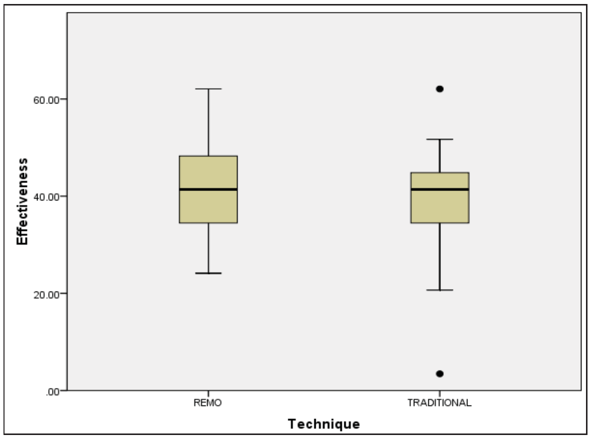


Figure 2. Boxplots for the Effectiveness indicator of the requirements.

For the indicator "requirements adequacy" we also performed a statistical analysis through the Mann-Whitney test and boxplots. Fig. 3 shows the boxplot graph regarding the adequacy of the requirements that were identified by the approaches.

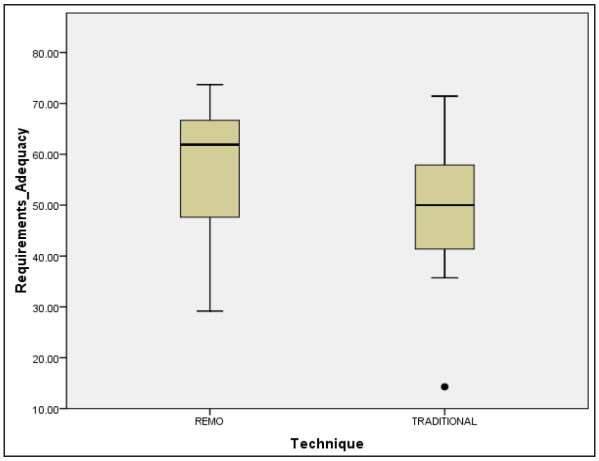


Figure 3. Boxplots for the Adequacy indicator of the requirements.

A further analysis of Fig. 3 indicates that there is a significant difference between the two groups. This result was confirmed by the Mann-Whitney test, which presented a significance level of p=0.106 regarding the degree of adequate requirements. These results support the alternative hypothesis HA2, and, conversely, rejecting the null hypothesis H02. These results indicate that the REMO technique provides more adequate requirements and less inadequate requirements (false positives) than the traditional approach.

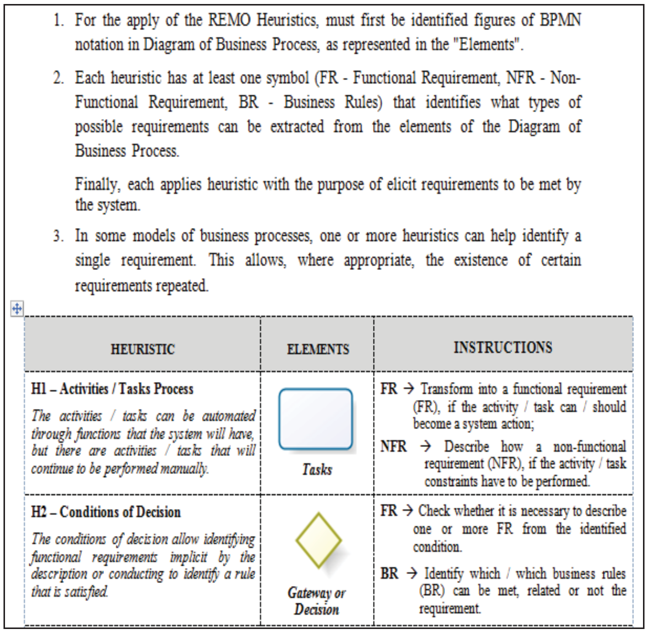
V. EVOLVING THE REMO TECHNIQUE

We noticed that the results from the first empirical study indicated a need for improving the technique. We used data obtained from follow-up questionnaires to acquire both perceived difficulties in the application of the technique and improvement suggestions. We analyzed these data in order to suggest changes in the first version of the proposed technique.

In the second version of the technique we modified the adopted approach by changing the order of the activities. First the analysts identified the set of elements from the BPMN set and afterwards they applied the heuristics to obtain the software requirements. Furthermore, we modified the way in which the heuristics were presented. We separate the examples from the heuristics' descriptions. Moreover, we added a procedure describing the steps for applying the heuristics in order to minimize the subjects' effort in understanding and remembering how to apply the technique.

The REMO technique adopted these modifications in its second version, which possesses 9 heuristics. Moreover, the REMO technique involves the following BPMN elements: task, gateway or decision, message event, conditional event, time event, intermediate event, data object, notes and swimlane.

Each heuristic has instructions to be applied in order to extract the requirements within the business processes diagrams. These instructions are classified according to the type of requirement it expects to identify. Fig. 4 shows an extract of the second version of the REMO technique. A detailed description of the second version of the technique is available in [14].



1. For the apply of the REMO Heuristics, must first be identified figures of BPMN notation in Diagram of Business Process, as represented in the "Elements".
2. Each heuristic has at least one symbol (FR - Functional Requirement, NFR - Non-Functional Requirement, BR - Business Rules) that identifies what types of possible requirements can be extracted from the elements of the Diagram of Business Process.

   Finally, each applies heuristic with the purpose of elicit requirements to be met by the system.
3. In some models of business processes, one or more heuristics can help identify a single requirement. This allows, where appropriate, the existence of certain requirements repeated.

HEURISTIC	ELEMENTS	INSTRUCTIONS
H1 – Activities / Tasks Process *The activities / tasks can be automated through functions that the system will have, but there are activities / tasks that will continue to be performed manually.*	▢ *Tasks*	**FR** → Transform into a functional requirement (FR), if the activity / task can / should become a system action; **NFR** → Describe how a non-functional requirement (NFR), if the activity / task constraints have to be performed.
H2 – Conditions of Decision *The conditions of decision allow identifying functional requirements implicit by the description or conducting to identify a rule that is satisfied.*	◇ *Gateway or Decision*	**FR** → Check whether it is necessary to describe one or more FR from the identified condition. **BR** → Identify which / which business rules (BR) can be met, related or not the requirement.

Figure 4. Extract example of the second version of the REMO technique.

VI. SECOND EMPIRICAL STUDY

We planned and evaluated the second version of the REMO technique through a new feasibility study in order to achieve the following goal, presented in Table III using the principals of the GQM [2] paradigm.

TABLE III. GOAL OF THE 2ND. EMPIRICAL STUDY

Analyze	The 2nd version of the REMO technique
For the purpose of	Characterizing
With respect to	Effectiveness and adequacy of the identified requirements using the business processes in the BPMN, when compared to a traditional approach.
From the point of view	Software Engineering Researchers
In the context of	A requirements elicitation carried out by graduation students and based on business processes models

Hypotheses: the study was planned and conducted in order to test the following hypotheses (null and alternative, respectively) regarding the effectiveness and adequacy indicators:

H01: There is no difference in t erms of effectiveness in using the REMO technique to elicit requirements regarding business processes.

HA1: The REMO technique presented a difference in the effectiveness indicator, when compared to a traditional approach.

H02: There is no difference in terms of adequacy of the identified requirements when using the REMO technique and a traditional approach.

HA2: The REMO technique presented different results regarding the adequacy of the identified requirements (not considering false positives), when compared to a traditional approach.

Context: We carried out the 2nd empirical study in November 2011, with others senior-level undergraduate students from Computer Science course (Federal University of Amazonas). In this experiment, the students also had tutorials regarding requirements elicitation and business processes modeling. The business process model chosen as object of study was part of the processes from the Academic Secretary of the University. We selected four academic processes: (a) to do enrollment adjustment; (b) to request recovery of academic credits; (c) request grades correction; and (d) monitoring application.

Subjects: 20 subjects agreed to participate in this study and played the role of system analysts. These subjects were classified and divided into two groups of 10 subjects. We balanced each group regarding the subjects experience with respect to their experience in software development and requirements elicitation.

Instrumentation: the main instruments in this study were the same type used in the first study: (a) an execution guide with tasks, (b) the business context document containing the BPMN processes modeling, and (c) requirements register spreadsheet. Furthermore, the group that used the REMO technique received the technique's heuristics document.

Execution: the subjects carried out the study the same day in different rooms. There was a moderator in each room to guarantee that there was no communication among the subjects. The subjects had 180 minutes to carry out the requirements elicitation. At the end of the study, all subjects returned the requirements lists and the follow up questionnaire filled out.

We created a unique requirements list, in order to discriminate the requirements. A specialist in requirements elicitation carried out the discrimination process. The discrimination meeting was carried out to ide ntify which requirements were considered inadequate (false-positives).

Results and Quantitative Analysis: Table IV shows the results per group of subjects. We calculated the effectiveness indicator using 152 known requirements from the business process models used as object of study.

TABLE IV. QUANTITATIVE RESULTS PER SUBJECTS

Technique / Subject		**Experience**	**IIR**	**FP**	**TRR**	**EI**	**AI**
TRADITIONAL	**01**	High	63	21	42	27.63%	66.67%
	02	Medium	46	07	39	25.66%	84.78%
	03	Medium	59	09	50	32.89%	84.75%
	04	Medium	47	03	44	28.95%	93.62%
	05	Medium	54	09	45	29.61%	83.33%
	06	Low	31	07	24	15.79%	77.42%
	07	Low	47	15	32	21.05%	68.09%
	08	Low	37	10	27	17.76%	72.97%
	09	Low	21	09	12	7.89%	57.14%
	10	Low	29	05	24	15.79%	82.76%
REMO	**11**	High	52	05	47	30.92%	90.38%
	12	Medium	48	16	32	21.05%	66.67%
	13	Medium	25	10	15	9.87%	60.00%
	14	Medium	26	02	24	15.79%	92.31%
	15	Medium	26	05	21	13.82%	80.77%
	16	Low	32	06	26	17.11%	81.25%
	17	Low	43	06	37	24.34%	86.05%
	18	Low	45	00	45	29.61%	100.0%
	19	Low	48	03	45	29.61%	93.75%
	20	Low	41	03	38	25.00%	92.68%

Legend: IIR – Initially Identified Requirements; FP – False Positives; TRR – Total of Real Requirements; EI – Effectiveness Indicator; AI – Adequacy Indicator.

In order to validate these data we used the Mann-Whitney statistic method (supported by the SPSS Statistics v17.0 tool) and the boxplots analysis. As we can see in Fig. 5, the boxplots graph shows that the effectiveness indicator was similar among the subjects who used the technique and those who used the traditional approach. When we compared the two samples using the Mann-Whitney test, we did not find any significant differences between the two groups (p = 0.850 and α = 0.05) therefore, supporting the null hypothesis H01.

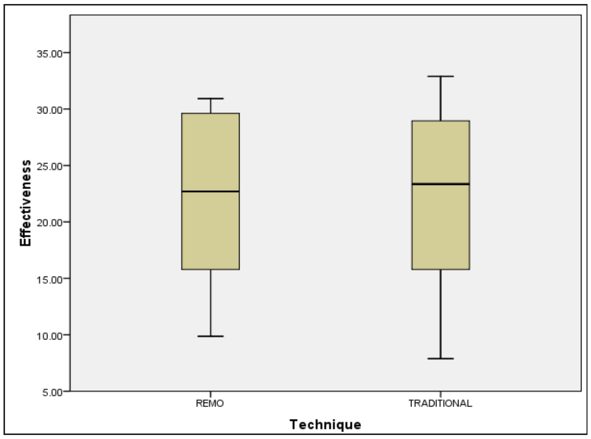


Figure 5. Boxplots for the Effectiveness indicator of the requirements.

For the indicator “requirements adequacy” we also did a boxplots graph, which is shown in Fig. 6. A further analysis of Fig. 6 indicates that there is a significant difference between the two groups. The Mann-Whitney (p=0.162) also suggests this conclusion, therefore rejecting the null hypothesis H02 and supporting the alternative hypothesis HA2. We need to execute more empirical studies to improve our results.

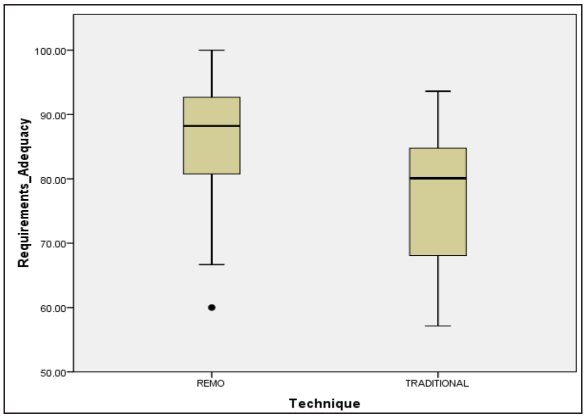


Figure 6. Boxplots for the Adequacy indicator of the requirements.

These results corroborate the results of the first empirical study. Both studies showed the REMO technique provides the same results regarding the effectiveness indicator compared to a traditional approach. Regarding the adequacy indicator, the first feasibility study of the technique showed a mean of 58.13%. The second feasibility study showed that the adequacy indicator increased to 84.39%. This increase indicates that the changes in the second version of REMO were positive.

VII. CONCLUSIONS

In this paper we motivate, propose and validate a new technique to aid extracting the requirements from business process diagrams: the REMO (Requirements Elicitation oriented by business process MOdeling) technique. Additionally, we discuss the results of two controlled experiments aimed at measuring the effectiveness and adequacy of the produced requirements in comparison to a traditional approach. These empirical studies allowed us to evaluate and evolve the technique.

Results showed that REMO had performed better than the traditional approach regarding the adequacy indicator and was also as effective as. This indicates that the REMO technique supports the extraction of requirements from business process diagrams, minimizing the number of inadequate requirements. This prevents the specification of requirements that are not in conformity with the identified needs may be extracted from the business processes.

We acknowledge that the small number of data points is not ideal from the statistical point of view. Small sample sizes are a known problem difficult to overcome. Even considering the limitation of the results due to the size of the sample used for the studies, the results obtained from the empirical studies seem to indicate the REMO feasibility.

As future work, we plan to (a) improve the technique using the results from the second study; (b) execute further studies to obtain more data points and strengthen the conclusion validity; (c) replicate the experiment involving professionals.

ACKNOWLEDGEMENTS

We would like to thank all students from the UFAM and members of the USES group who participated in the empirical studies. Furthermore, we would like to acknowledge the support granted by CAPES process AEX 4982/12-6. The first author also thanks FUCAPI for the financial support.

REFERENCES

[1] E. A. Carvalho, T. Escovedo, R. N. Melo, "Using Business Processes in System Requirements Definition," In: SEW - 33rd Annual IEEE Software Engineering Workshop, 2009, pp.125-130.

[2] V. Basili, H. Rombach, “The TAME Project: Towards Improvement-Oriented Software Environments”, IEEE Transactions on Software Engineering.,Maryland University,College Park, 1988, v.14 pp. 758-773.

[3] C. Monsalve, A. April, A. Abran. “Requirements Elicitation Using BPM Notations: Focusing on the Strategic Level Representation”. In: 10th WSEAS International Conference on Applied Computer and Applied Computational Science (ACACOS), 2011, Venice, Italy, pp. 235-241.

[4] E. C. S. Cardoso, J. P. A. Almeida, G. Guizzardi, “Requirements Engineering Based on Business Process Models: A Case Study”, Enterprise Distributed Object Computing Conference Workshops 13th, EDOCW/IEEE, Auckland, New Zealand, 2009, pp. 320-327.

[5] O. Demirors, C. Gencel, A. Tarhan, “Utilizing Business Process Models for Requirements Elicitation”, 29th EUROMICRO Conference, Ankara, Turkey, 2003, pp. 409-412.

[6] F. Dias, G. Morgado, P. Oscar, D. Silveira, A. J. Alencar, P. Lima, E. Schmitz, “An Approach for Automatic Transformation of Business Models into Requirement Models”, Proceedings of the Workshop on Requirements Engineering (WER’06), Rio de Janeiro-RJ, Brazil, 2006, pp. 51-60 (In Portuguese).

[7] H. Estrada, A. Martinez, O. Pastor, J. Ortiz, O. Rios, “Automatic generation of Oo Conseptual Schemes from Workflow Models”, Proceedings of the Workshop on Requirements Engineering (WER’02), Valencia, Spain, 2002, pp. 177-193. (In Spanish)

[8] J. L. de la Vara, J. Sánchez, O. Pastor. “Business Process Modelling and Purpose Analysis for Requirements Analysis of Information Systems”. In: CAiSE '08 of the 20th international conference on Advanced Information Systems Engineering. 2008, pp. 213 – 227.

[9] OMG (Object Management Group) – Business Process Model and Notation (BPMN), version 2.0., Available at: http://www.omg.org/spec/BPMN/2.0/PDF, 2011.

[10] A. Martinez, J. Castro, O. Pastor, H. Estrada, “Closing the gap between Organizational Modeling and Information System Modeling”. Proceedings of the Workshop on Requirements Engineering (WER’03), Piracicaba, São Paulo-SP, Brazil, 2003, pp. 93-108.

[11] L. E. G. Martins, B. M. Daltrini, “Organizing the Process of Requirements Elicitation Using the Concept of Activity”. Proceedings of the Workshop on Requirements Engineering (WER’01), Buenos Aires, Argetina, 2001, pp. 297-317 (In Portuguese).

[12] V. F. A. Santander, J. F. B. Castro, “Deriving Use Cases from Organizational Modeling”. In: IEEE Joint International Conference on Requirements Engineering, Washington, DC, USA, 2002, pp. 32-42.

[13] SEI-Software Engineering Institute. CMMI® for Development, V1.2, CMU/SEI-2010-TR-033. Carnegie Mellon University, 2010.

[14] S. R. C. Vieira, R. P. C. do Nascimento, T. Conte, “Technical Report: Heuristics of the REMO Requirements Elicitation Technique”, Report Number 007, 2011. Available at: http://www.dcc.ufam.edu.br/uses

[15] L. Xavier, F. Alencar, J. Castro, J. Pimentel, “Integrating Non-Functional Requirements to Business Processes: Integrating BPMN and NFR”, Proceedings of the Workshop on Requirements Engineering (WER’10), Cuenca, Ecuador, 2010, pp. 29-40 (In Portuguese).

[16] E. Yu: “Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering”. Proceedings of IEEE International Symposium on Requirements Engineering - RE97, 1997, pp. 226-235.

Consistency Checks of System Properties Using LTL and Büchi Automata

Salamah Salamah, Matthew Engskow
Department of Electrical, Computer, Software, and Systems Engineering
Embry Riddle Aeronautical University, Daytona Beach, Florida, USA
{salamahs, engskowm}@erau.edu

Omar Ochoa
Department of Computer Science
The University of Texas at El Paso
El Paso, Texas, USA
omar@miners.utep.edu

Abstract

Although formal approaches to software assurance such as model checking and theorem proving improve system dependability, software development professionals have yet to adopt these approaches. A major reason for the hesitance is the lack of maturity of tools that can support use of formal specification approaches. These techniques verify system correctness against formal specifications. Tools such as the Specification Pattern System (SPS) and the Property Specification tool (Prospec) assist users in translating system properties (requirements) into formal specifications in multiple languages such as Linear Temporal Logic (LTL). The goal of such tools is to aid in the generation of a large set of formal specifications of systems or subsystems. A major advantage of formal specifications is their ability to discover inconsistencies among the generated properties. This paper provides an approach to extend the work of the aforementioned Prospec tool to allow for consistency checks of automatically generated system properties. The work will allow for the discovery of discrepancies among system requirements at early stages of system development, providing a return on investment for specifying formal properties using the Prospec tool.

1 Introduction

Today more than ever, society depends on complex software systems to fulfill personal needs and to conduct business. Software is an integral part of numerous mission and safety critical systems. Because of society's dependence on computers, it is vital to assure that software systems behave as intended. It is alarming to consider that software errors cost U.S. economy $59.5 billion annually [13]. The same studies also show that a significant amount of resources can be saved if software defects are discovered at the early stages of development such as requirements and design, rather than the later stages such as implementation and testing. Because of that, it is imperative that the software industry continue to invest in software assurance approaches, techniques, and tools, especially ones that ensure early detection of defects.

The use of formal methods in software engineering can be of great value to increase the quality of developed systems. Formal approaches to software assurance require the description of behavioral properties of the software system, generation of formal specifications for the identified properties, validation of the generated specifications, and verification of system's adherence to the formal specification. The effectiveness of the assurance approach depends on the quality of the formal specifications. A major impediment to the adoption of formal approaches in software development remains the difficulty associated with the development of $correct$ formal specifications (i.e, ones that match the specifier's original intent) [6, 7]. It is also important that the generated formal properties are consistent. A major advantage of using formalism in describing software properties, is that it becomes easier to discover conflicts between properties as early as these properties are generated, which is typically during the requirement and design phases.

Currently, there exist multiple formal specification languages that can be used in a variety of verification techniques and tools. Linear Temporal Logic (LTL) [10], and Computational Tree Logic (CTL) [9]are two of these languages. The aforementioned languages can be used in a variety of verification techniques and tools. For example, the model checkers SPIN [8] and NuSMV [1] use LTL to specify properties of software and hardware systems. On the other hand, the SMV [2] model checker verifies system behaviors against formal properties in CTL.

In this work we propose an approach and a tool to check the consistency between system properties specified in LTL which is a prominent formal specification language in software engineering. The importance of the work stems from the fact that formal properties can be generated early in the development cycle, and detecting discrepancies among properties at that early stage can lead to significant savings in time and resources. The approach to detect inconsistencies makes use of translating LTL specifications into a special type of state machines called a Büchi automata and then checking for the intersection of the state machines for these specifications. This is the same approach used by model checkers to check the correctness of models against formal specifications of properties. The developed tool allows users to examine LTL specifications against other specifications and report any conflicts. The tool is intended to be part of the Property Specification (Prospec) tool [11, 12].

The rest of the paper is organized as follows; Section 2 provides the necessary background for the rest of the work including a description of LTL and Büchi Automata. Section 3 provides the motivation for the work. Sections 4 and 5 introduce the new approach for consistency check and a special tool developed for that purpose respectively. The paper concludes with a summary and the references.

2 Background

This section discusses the background work necessary for the rest of the paper. Specifically we describe the LTL language, the Prospec tool, the notion of Büchi automata, and the model checking approach for consistency checks.

2.1 Linear Temporal Logic

Linear Temporal Logic (LTL) is a prominent formal specification language that is highly expressive and widely used in formal verification tools such as the model checkers SPIN[8] and NuSMV [1]. LTL is also used in the runtime verification of Java programs [16].

Formulas in LTL are constructed from elementary propositions and the usual Boolean operators *not*, *and*, *or*, *imply* (neg, $\wedge$, $\vee$, $\rightarrow$, respectively). In addition, LTL allows for the use of the temporal operators $next$ (X), *eventually* (F), $always$ (G), $until$, (U), $weak\ until$ (W), and $release$ (R).

Formulas in LTL assume discrete time, i.e., states $s = 0, 1, 2, \ldots$ The meanings of the temporal operators are straightforward. The formula XP holds at state s if P holds at the next state $s + 1$. $P\,U\,Q$ is true at state s, if there is a state $s' \geq s$ at which Q is true and, if s' is such a state, then P is true at all states s_i for which $s \leq s_i < s'$. The formula FP is true at state s if P is true at some state $s' \geq s$. Finally, the formula GP holds at state s if P is true at all moments of time $s' \geq s$. Detailed description of LTL is provided by Manna et al. [10].

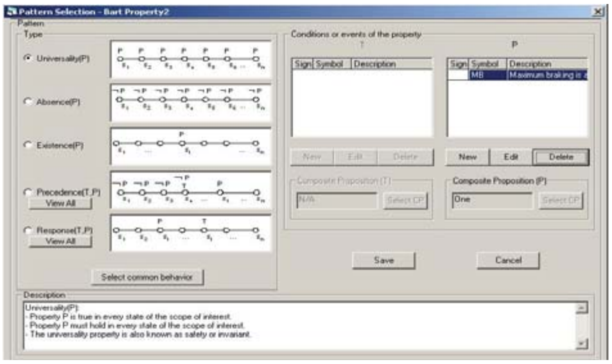
A major factor for the hesitance in using temporal logics in general is that they are hard to write. In addition, once specifications are written in LTL or CTL, it is hard to read and validate the meaning of the generated statement. For example, it is not immediately obvious that the LTL specification $G(a \rightarrow F(p \wedge F(\neg p \wedge \neg a)))$ represents the English requirement "If a train is approaching(a), then it will be passing(p), and later it will be done passing with no train approaching". The proposed tool aims at providing the means by which developers and users can validate the meaning of such specifications as the one above.

2.2 Specification Pattern System and Prospec

Because of the difficulties associated with developing formal specifications, the Specification Pattern System (SPS) [3] developed a set of patterns to assist users in witting formal specifications in multiple formal languages. Patterns are high-level abstractions that provide descriptions of common properties that hold on a sequence of conditions or events in a finite state model. SPS patterns are grouped occurrence and order. Occurrence patterns are $universality$, $absence$, $existence$, and $bounded\ existence$. Order patterns are $precedence$, $response$, $chain\ of\ precedence$ and $chain\ of\ response$. Chain patterns define a sequencing of events or conditions. Chain-precedence and chain-response patterns permit specifying a sequence of events or conditions as a parameter of precedence or response patterns, respectively. SPS allows the specification of sequences only to precedence and response patterns.

In SPS, a pattern is bounded by the scope of computation over which the pattern applies. The beginning and end of the scope are specified by the conditions or events that define the left (L) and right (R) boundaries, respectively. A study by Dwyer et. al. [3] identified the response pattern as the most commonly used pattern, followed by the universality and absence patterns. These three patterns accounted for 80% of the 580 properties sampled in the study.

In many system properties multiple propositions may be needed to specify pattern or scope parameters. Mondragon et al. [11] introduced Composite Propositions (CPs) to handle pattern and scope parameters that represent multiple conditions or events. This was done as part of the Property Specification (Prospec) tool [12, 5]. The introduction of CPs supports the specification of concurrency, sequences, and non-consecutive sequential behavior on patterns and scopes. Mondragon proposes a taxonomy with



Sign	Symbol	Description
	MB	Maximum braking is a

Composite Proposition (T): N/A

Composite Proposition (P): One

Description

Universality(P):
- Property P is true in every state of the scope of interest.
- Property P must hold in every state of the scope of interest.
- The universality property is also known as safety or invariant.

Figure 1: Prospec's Pattern Screen.

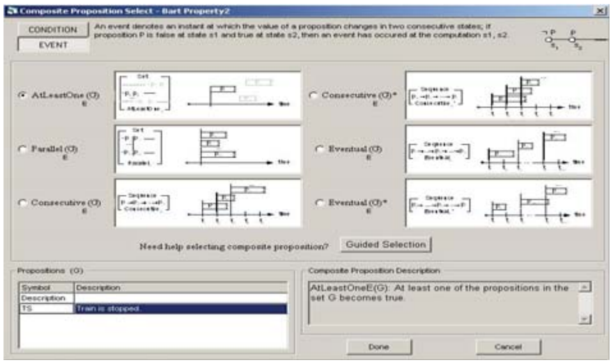
twelve classes of CPs. In this taxonomy, each class defines a detailed structure for either concurrent or sequential behavior based on the types of relations that exist among a set of propositions. The complete list of CP classes and their LTL descriptions is available in Mondragon et. al. [11].

Prospec is an automated tool that guides a user in the development of formal specifications. It includes patterns and scopes, and it uses decision trees to assist users in the selection of appropriate patterns and scopes for a given property. Prospec extends the capability of SPS by supporting the specification of CP classes for each parameter of a pattern or scope that is comprised of multiple conditions or events. By using CPs, a practitioner is directed to clarify requirements, which leads to reduced ambiguity and incompleteness of property specifications.

Prospec uses guided questions to distinguish the types of scope or relations among multiple conditions or events. By answering a series of questions, the practitioner is guided to consider different aspects of the property. A type of scope or CP class is identified at the end of guidance. The soon to be completed Prospec 2.0 generates formal specifications in Future Interval Logic (FIL), Meta-Event Definition Language (MEDL), and LTL. The automatic generation of CTL specification is left as future work. Figures 1 and 2 provide screen shots of the Prospec tool. More detailed description of Prospec 2.0 can be found in [5].

2.3 Model Checking and Büchi Automata

Classical LTL model checking is based on a variation of the classic theory of finite automata [8]. While a finite automaton accepts only terminating executions, model checking requires a different type of machines that can handle executions that might not terminate. Such machines are necessary to model nonterminating systems such as operating systems, traffic lights, or ATMs. One such machine is a Büchi automaton. A Büchi automaton is a tuple $(Q, \sum, \delta, Q_0, F)$ where Q is a finite set of states, $Q_0 \subseteq Q$ is a set of initial states, $\sum$ is an alphabet, δ: $Q \times \sum \to 2^Q$ is a transition function, and $F \subseteq Q$ is set of accepting states.



An event denotes an instant at which the value of a proposition changes in two consecutive states; if proposition P is false at state s1 and true at state s2, then an event has occured at the computation s1, s2.

Propositions (G)

Symbol	Description
Description	
TS	Train is stopped.

Composite Proposition Description

AtLeastOneE(G): At least one of the propositions in the set G becomes true.

Figure 2: Prospec's Composite Proposition Screen

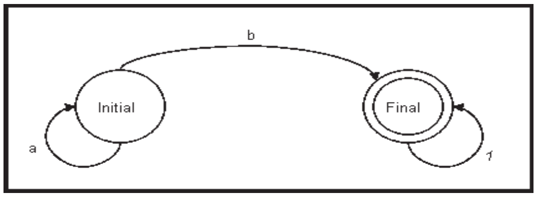


Figure 3: Büchi Automata for "a U b".

An execution is a sequence $s_0, s_1, \ldots$, where for all $s_i \in Q$ and for all $i \geq 0$, $(s_i, s_{i+1}) \in \delta$. A finite execution is an accepting execution if it terminates in a final state $s_f \in F$. An infinite execution, also called an $w - execution$ or $w - run$, is accepting if it passes through a state $s_f \in F$ infinitely often. An empty Büchi Automata (accepts no words) is one that either terminates in a state that is not an accepting state or has no accepting state that is reachable from the initial state and that is visited infinitely often. The set of executions accepted by a Büchi Automata is called the language of the Büchi Automata.

Languages of Büchi Automata represent a superset of those of LTL; every LTL formula can be represented by a Büchi Automata. When a Büchi Automata is generated from an LTL formula, the language of the Büchi Automata represents only the traces accepted by the LTL formula. For example, the Büchi Automata in Figure 3 represents the language accepted by the LTL formula $(a\, U\, b)$. This formula specifies that b holds in the initial state of the computation, or a holds until b holds. The language of the Büchi Automata in Figure 3 accepts the set of traces $\{b \ldots, ab \ldots, aab \ldots, \ldots, aaab\}$. Notice that each of these traces passes through the accepting state Final. This state is both reachable from the initial state and is visited infinitely often (by virtue of the self-transition marked 1).

In Automata-Based model checking, the system model is written in the model checker modeling language (in case of SPIN the language is Promela [8], and in NuSmv it is the SMV language [1]. The languages of these model checkers allow for the presentation of system models as Büchi automata. On the other hand, system properties of interest are provided as temporal logic formulas (SPIN accepts properties in LTL, and NuSMV accepts properties in both LTL and CTL). In the case of LTL formulas, the model checker translates the negation of LTL specification into a Büchi automata and checks the intersection of both Büchi automata for the system model and that of the negated specification. If the intersection is empty, then the model and the original (non-negated specification) are deemed consistent. Otherwise, if the intersection is non-empty (it contains accepted execution traces) then the model accepts some behavior of the negated specification which implies an inconsistency between the model and the specification [2]. In this work, we use the same approach to check consistency between LTL specifications. Specifically, we check for emptiness of the generated Büchi automata that results from "anding" all LTL formulas for the properties in question.

3 Motivation

The goal of automated formal property generation tools like Prospec, is to generate formal specification for multiple system properties. However, Propsec only generates these properties and does not support any consistency checks among these properties. Considering that in a typical software development environment, system properties are elicited by multiple developers for different parts of the system under development, it is essential that these properties are consistent. Lack of overall consistency among system properties will result in problems that will surface during system integration. Discovering defects this late in development will result in significant rework and delays in production.

While formal specifications, by their nature, are better suited for formal analysis and consistency checks, they are hard to manually read or validate and, as such, are difficult to analyze for consistency by manual means. We provide a tool that takes as an input, multiple formal specifications (in LTL) generated by tools such as Prospec, and returns a verdict on the consistency between the set of formal specifications under questions. Moreover, the LTL Consistency checker tool, reports on the group of LTL specifications that cause this inconsistency.

4. Consistency of Formal Properties

Once systems properties are available as LTL formulas it is desirable to check the consistency among the generated properties. In this section we provide an approach to test the consistency of multiple properties written as LTL formulas. In our approach to demonstrate consistency between two LTL formulas we check for emptiness of the Büchi Automata of the conjunction of both formulas. Given two LTL formulas LTL_1 and LTL_2 the two formulas are consistent if the Büchi Automata for $(LTL_1 \wedge LTL_2)$ is not empty (i.e, the combined LTL formula "$LTL_1 \wedge LTL_2$" accepts some behaviors or execution traces).

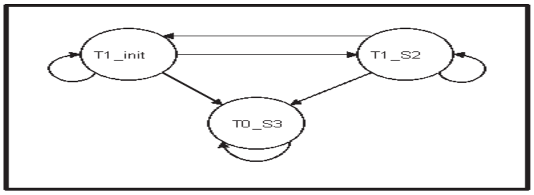


Figure 4: LTL2BA generated BA for $(\neg(G((l \wedge \neg r) \rightarrow ((Gp) \vee (pUr))))) \wedge (G((l \wedge \neg r) \rightarrow (\neg((p \wedge \neg r)U((\neg p) \wedge \neg r)))))$

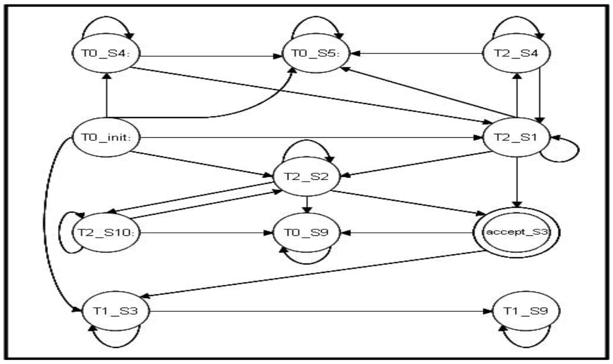


Figure 5: LTL2BA generated BA for $((G((l \wedge \neg r) \rightarrow ((Gp) \vee (pUr))))) \wedge \neg(G((l \wedge \neg r) \rightarrow (\neg((p \wedge \neg r)U((\neg p) \wedge \neg r)))))$

By definition a Büchi Automata is empty if it does not contain a reachable accepting state that is visited infinitely often [2]. The work described here consists of using the LTL2BA[4] tool to generate the Büchi Automata for the formula $(LTL_1 \wedge LTL_2)$ and to check for an absence of acceptance state(s). If acceptance state(s) is/are available then we check that none of them is within a cycle (i.e., visited infinitely often).

An example of a consistent formula is shown in Figure 3 above. The figure shows the Büchi Automata for the formula "a U b". This is clearly not an empty Büchi Automata as there is a final state (the double circle state called Final), and this state is within a cycle by virtue of the self loop (i.e, it is visited infinitely often).

Two examples of empty Büchi Automata are shown in Figures 4 and 5. Figure 4 shows the BA for the formula "$((\neg(G((l \wedge \neg r) \rightarrow ((Gp) \vee (p\,U\,r))))) \wedge (G((l \wedge \neg r) \rightarrow (\neg((p \wedge \neg r)U((\neg p) \wedge \neg r))))))$". This Büchi Automata is empty because it contains no acceptance states (no double circles). The Büchi Automata in Figure 5 is for the LTL formula "$(((G((l \wedge \neg r) \rightarrow ((Gp) \vee (p\,U\,r))))) \wedge \neg(G((l \wedge \neg r) \rightarrow (\neg((p \wedge \neg r)U((\neg p) \wedge \neg r))))))$". Although the Büchi Automata in Figure 5 contains an accepting state, this state is not contained within a cycle and (it cannot be visited infinitely often), as such , this too is an empty automaton.

5 LTL Consistency Check Tool

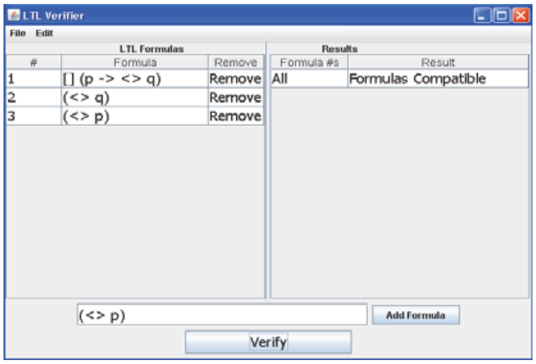
The LTL Consistency Check Tool[1] is designed to ensure that a given LTL formula is valid and is not contradictory (accepts some behaviors and is not empty). It may also be used to check that two separate LTL formulas do not contradict by using the AND operator. In addition, the tool can be used to prove the equivalence of two formulas. Provided we have formula's A and B, checking to see that (A ∧¬ B) is invalid and that (B ∧¬ A) is invalid will prove that the formulas are equivalent [14].

While the above examples in Figures 3-5 can be checked for consistency simply by examining the diagrams (these are manually drawn diagrams based on the textual output by LTL2BA) for the generated Büchi Automata, other more complex Büchi Automata can be hard to visually check for consistency. For this purpose, we have developed the LTL Consistency Check tool. The tool allows the user to input any number of LTL formulas to check for their consistency. The tool makes use of LTL2BA [4] to construct the resulting Büchi Automata for the conjunction of these LTL formulas. The tool interfaces with LTL2BA to pass the "anded" list of LTL formulas (as one formula) and get back the resulting Büchi Automata (in text formats). From here, the tool tests the consistency of the formulas by looking for a reachable acceptance state that falls within a cycle. This is performed by first finding all states reachable from every given state, then by linking these together to form cycles. Should the tool find a single acceptance cycle, that is, an acceptance state which can reach itself, the formulas are declared consistent. Otherwise there is an inconsistency among the formulas. This is the same approach used in automata model checking [2].

In case of inconsistency, the tool will loop through all the permutations of the input formulas and perform the same consistency checks on each permutation group. This procedure will ensure that the user is informed not only of the presence of an inconsistency, but also which set of specifications are inconsistent.

Example1 Figure 6 shows a screen shot of the LTL consistency check tool showing no inconsistencies between the three LTL formulas[2]:

- $G(p \rightarrow Fq)$
- Fq

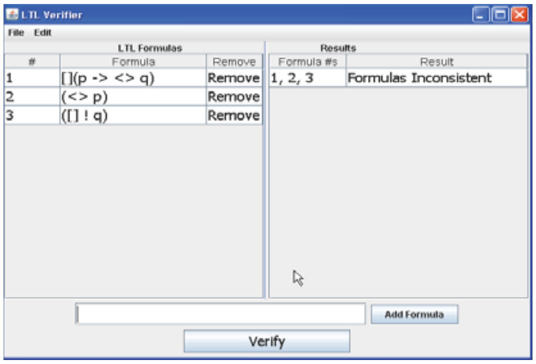


#	Formula	Remove
1	[] (p -> <> q)	Remove
2	(<> q)	Remove
3	(<> p)	Remove

Formula #s	Result
All	Formulas Compatible

(<> p)

Figure 6: Screen shot for Example 1



#	Formula	Remove
1	[](p -> <> q)	Remove
2	(<> p)	Remove
3	([] ! q)	Remove

Formula #s	Result
1, 2, 3	Formulas Inconsistent

Figure 7: Screen shot for Example 2

- Fp

These three formulas are obviously consistent. The first describes a response formula, where a cause p must be followed by a effect q. The second and third formula describe the behavior that at some future point of execution q will hold and p will hold respectively.

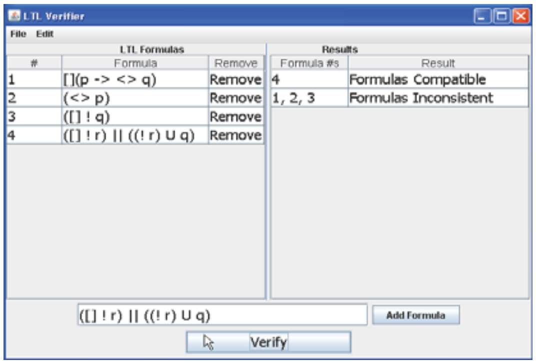
Example2 Figure 7 shows the screen shot of the LTL consistency tool showing an inconsistency between the following three LTL formulas:

- $G(p \rightarrow Fq)$
- Fp
- $G\neg p$

These three formulas are obviously inconsistent. The first formula specifies that every occurrence of p must be followed by a q, and the second formula specifies that there will be an occurrence of p. Combining the two formulas one can conclude that q must happen at some future point. However the third formula states that q will never hold. As a result, the three formulas, combined, are inconsistent.

[1] The current tool can be requested from salamahs@erau.edu. Also a demo of the tool will be performed at SEKE 12.

[2] Note that the consistency check tool makes use of the SPIN syntax for (Global (G): []), (Future (F): <>), and (Logical OR: ||)



LTL Formulas		
#	Formula	Remove
1	[](p -> <> q)	Remove
2	(<> p)	Remove
3	([] ! q)	Remove
4	([] ! r) || ((! r) U q)	Remove

Results	
Formula #s	Result
4	Formulas Compatible
1, 2, 3	Formulas Inconsistent

([] ! r) || ((! r) U q)

Figure 8: Screen shot for Example 3

Example3 Figure 8 shows the screen shot of the LTL consistency tool showing an inconsistency between the following four LTL formulas:

- $G(p \rightarrow Fq)$
- Fp
- $G\neg p$
- $(G\neg r) \vee ((\neg r)\, U\, q)$

This latest example shows that the tool will distinguish between the inconsistent formulas and others that do not pose any inconsistency with other formulas. As such, the tool reports that the first three formulas are inconsistent (as described in Example 2) while the last formulas has no consistency issues with other formulas and itself is a valid formula (does accept certain execution traces).

6 Summary and Future Work

Formal verification techniques such as model checking and theorem proving have been shown to increase system dependability. These techniques verify system models against formal specifications. As such, the success of these techniques depends on the quality of developed systems properties. Tools that assist in the generation of formal specifications in LTL are important to the formal verification community as they relieve the user from the burden of writing specifications in a language that is hard to read and write. Without the help of tools such as Prospec, the user might create faulty specifications. These tools must generate specifications that correspond to the intent of the user. Prospec has demonstrated to provide such support [15]. However, it is also important that tools like Prospec generate specifications that are consistent with each other. Discovering inconsistencies among specifications early in the requirement or design phases should reduce integration and testing times significantly.

This work introduced a new approach, complemented with a tool, to automatically check for consistency among multiple formal specifications in LTL. The tool will help the users discover inconsistencies among their specifications at early stages. While the current tool is a stand-alone tool that has been developed separately from Prospec, it is our goal to integrate this tool with the latest release of Prospec in order to encourage the use of Prospec.

References

[1] Cimatti, A., Clarke, E., Giunchiglia, F., and Roveri, M., "NuSMV: a new Symbolic Model Verifer" International Conference on Computer Aided Verifcation CAV, July 1999.

[2] Clarke, E., Grumberg, O., and D. Peled. "Model Checking". MIT Publishers, 1999.

[3] Dwyer, M. B., Avrunin, G. S., and Corbett, J. C., "Patterns in Property Specification for Finite-State Verification," Proceedings of the 21st Intl. Conference on Software Engineering, Los Angeles, CA, USA, 1999, 411-420.

[4] Fritz, C., "Constructing Büchi Automata from Linear Temporal Logic Using Simulation Relations for Alternating Büuchi Automata," Eighth Conference on Implementation and Application of Automata 2003.

[5] Gallegos, I., Ochoa, O., Gates, A., Roach, S., Salamah, S., and Vela, C., "A Property Specification Tool for Generating Formal Specifications: Prospec 2.0". In the Proceeding of the Conference of Software Engineering and Knowledge Engineering (SEKE), Los Angelos, CA, July 2008.

[6] Hall, A., "Seven Myths of Formal Methods," IEEE Software, September 1990, pp. 11-19.

[7] Holloway, M., and Butler, R., "Impediments to Industrial Use of Formal Methods," IEEE Computer, April 1996, pp. 25-26.

[8] Holzmann, G, J., "The SPIN Model Checker: Primer and Reference Manual", Addison-Wesley Professional, Boston, Mass,USA, 2004.

[9] Laroussinie, F. and Ph. Schnoebelen, "Specification in CTL+Past for verification in CTL," Information and Computation, 2000, 236-263.

[10] Manna, Z. and Pnueli, A., "Completing the Temporal Picture," *Theoretical Computer Science,* 83(1), 1991, 97–130.

[11] Mondragon, O. and Gates, A., "Supporting Elicitation and Specification of Software Properties through Patterns and Composite Propositions," Intl. Journal Software Engineering and Knowledge Engineering, XS 14(1), Feb. 2004.

[12] Mondragon, O., Gates, A., and Roach, S., "Prospec: Support for Elicitation and Formal Specification of Software Properties," in Proceedings of Runtime Verification Workshop, ENTCS, 89(2), 2004.

[13] National Institute of Standards and Technology (NIST), June 02, http://www.nist.gov/public-affairs/releases/n02-10.htm

[14] Salamah, S., Gates, A., Roach, S., and M. Engskow, "Towards Support for Software Model Checking: Improving the Efficiency of Formal Specifications," Advances in Software Engineering, vol. 2011, Article ID 869182, 13 pages, 2011. doi:10.1155/2011/869182.

[15] Salamah, S., Gates, A., Roach , S., and Mondragon, O., "Verifying Pattern-Generated LTL Formulas: A Case Study. Proceedings of the 12th SPIN Workshop on Model Checking Software. San Francisco, California, August, 2005, 200-220

[16] Stolz, V., and Bodden, E., "Temporal Assertions using AspectJ", *Fifth Workshop on Runtime Verification* Jul. 2005.",

Evaluating the Cost-Effectiveness of Inspecting the Requirement Documents: An Empirical Study

Narendar Mandala, Gursimran S. Walia
Department of Computer Science
North Dakota State University
Fargo, ND
{narendar.mandala, gursimran.walia}@ndsu.edu

Abstract—**Inspections and testing are two widely recommended techniques for improving software quality. While inspections are effective at finding and fixing the faults early in the lifecycle, there is a lack of empirical evidence regarding the extent to which the testing costs are reduced by performing the inspections of early software documents. This research applied the *Kusumoto* cost-metric that analyzed the costs and benefits of inspections versus testing using the fault data from four real requirement documents that were inspected twice. Another aspect of our research evaluated the use of *Capture Recapture* (CR) method to estimate the faults remaining post inspection to decide the need for a re-inspection. Our results provide a detailed analysis of the ability of the CR method to accurately estimate the cost savings (within +/- 20% of the actual value) after each inspection cycle.**

Keywords-cost-effectiveness; inspections; requirements.

I. Introduction

Successful software organizations strive to deliver high quality products on time and within budget. To manage software quality, researchers and practitioners have devoted considerable effort to help developers find and fix faults at the early stages of the lifecycle (i.e., in requirements and design documents) and reduce its impact in the later stages [1-3]. In software engineering research, software inspections have been empirically validated to improve quality by helping developers find and fix faults early and avoid costly rework [1-3, 6, 11].

Similar to inspections, testing is also widely recommended technique for improving software quality. While both are effective fault detection techniques, testing cannot be conducted until software has been implemented, whereas inspections can be applied immediately after software documents have been created. Empirical evidence suggests that a majority of the development effort is spent during the testing stage of project [6, 11, 12]. Furthermore, it is estimated that 40-50% of the testing effort is spent on fixing problems that should have been fixed during the early stages of the development [15, 17, 22]. However, there is little empirical evidence regarding the rework cost-savings that can be achieved by performing the inspections of early work products.

Therefore, an empirical evaluation of the costs and benefits of inspections (against the testing cost that would be spent if no inspections are performed) can help manage the software quality. To evaluate the costs and benefits of inspections and testing, this research reports the results from the application of *Kusumoto* cost-metric [13] that analyzed the testing costs that were reduced by performing inspections of software requirement documents immediately after its development.

Furthermore, the *kusumoto* metric calculation requires information regarding the total fault count of the document and the faults remaining post-inspection. However, inspections only provide the information about the faults that are found during an inspection. During software development, project managers need reliable estimates of the remaining faults after an inspection to help decide whether to invest in a re-inspection or to pass the document to the next phase. To that end, our prior research has shown that, among the different approaches that are available for estimating the total number of faults in the artifact (e.g., defect density, subjective assessment, historical data, capture-recapture, curve-fitting), capture-recapture (CR) method is the most appropriate and objective approach [19].

Capture-recapture (CR) is a statistical method originally developed by biologists to estimate the size of wildlife populations. To use CR, a biologist captures a fixed number of animals, marks them, and releases them back into the population. Then another trapping occasion occurs. If an animal that was 'marked' during the first trapping is caught again, it is said to have been recaptured. The process of trapping and marking can be repeated multiple times. The size of the population is then estimated using: 1) the total number of unique animals captured across all trappings, and 2) the number of animals that were re-captured [7-8, 22]. Using the same principle, the CR method can be used during the inspection process to estimate the number of faults in an artifact. During an inspection, each inspector finds (or captures) some faults. If the same fault is found by more than one inspector it has been re-captured [5, 10, 16, 19]. The total number of faults is estimated using the same process as in wildlife research, except that the *animals* are replaced by *faults* and the *trappings* are replaced by *inspectors*. The inspection team can use the estimate of the total fault count along with the number of faults already detected to estimate the number of faults remaining in the artifact post inspection.

This paper reports a comprehensive evaluation of *Kusumoto* metric in conjunction with the CR method using inspection data from real artifacts that contain natural faults made during the development. The artifacts were inspected two times, and we compared the cost-effectiveness after each inspection cycle using the CR estimates against the actual fault count. We also analyzed the ability of the CR method to make a cost-effective decision post-inspection. The rest of the paper is structured as follows: Section II describes the inspection cost model and metrics, the basic principles of CR models in software inspections. Section III describes the literature review used for the evaluation study. Section IV describes the design of the evaluation study. Section V reports the results. Section VI discusses the threats to validity. Section VII summarizes the results. Section VIII contains the conclusions and future work

II. Background

This section provides information regarding the inspection costs and savings, different cost-metrics, the basic principles of the use of CR models in inspections, and a summary of the empirical studies related to cost-effectiveness of inspections.

A. *Inspection Cost Model*

The traditional software inspection cost model [13] consists of the following components. The process of calculating these components after the inspection is discussed as follows:

- D_r *- number of unique faults found during the inspection*; is determined by comparing the faults found by all the inspectors.
- D_{total} *- total number of faults present in the product*; is not available during the development. In this research, the overlap in the faults found by multiple inspectors during the inspection is used to estimate the total fault count using the CR estimators.
- C_r *– cost spent on an inspection*; is measure of the time taken (in hours) to review the software artifact. In this research, we only consider the cost invested during the "*individual review/preparation*" stage of the inspection process and is calculated by adding the time taken by each inspector during the inspection.
- C_t *– cost spent to detect remaining faults in testing*; is the cost required to detect the faults remaining post inspection. If we consider c_t as the average cost to detect a fault in the testing stage, then C_t can be measured as the product of total number of faults remaining post-inspection *(D_{total} – D_r)* and the average cost to detect a fault during testing (c_t).
    - c_t, *– Average cost to detect a fault in testing*, is not available after the inspection. Only the average cost to detect a fault in inspection is available. Therefore, a cost ratio of 1:6 (i.e., time spent to find a fault during the testing is six times the time spent to find a fault during the inspection) was used to calculate the average cost to detect a fault in testing. This cost ratio is derived from literature and described in Section III.
- $\Delta\ C_t$ *– testing cost saved by the inspection*. By spending cost C_r during the inspection, cost ΔC_t is being saved during the testing. It is calculated as the product of number of faults found during an inspection (D_r) and the average cost to detect a fault in testing (c_t). That is, ΔC_t = Dr * c_t
- C_{vt} *– Virtual testing cost*, (i.e., testing cost expended if no inspections are performed) is the sum of the testing cost required to detect the faults remaining post-inspection (C_t) and the testing cost saved by the inspection (ΔC_t). That is, $C_{vt} = (C_t + \Delta C_t)$.

B. *Software Inspection Cost-Metrics*

Meyer [15] and Fagan [11] have used a subset of the components listed in II.A to propose metrics for evaluating the effectiveness (*number of faults*) and efficiency (*faults per hour*) of inspections. Their research has neglected the cost spent and cost saved by the inspections. In this paper, only the inspection metrics that considered cost factors are discussed.

Collofello's Metric (M_c): Collofello et al., [9] defined a cost-effectiveness metric as the ratio of the *cost saved by inspections* (ΔC_t) to the *cost consumed by inspections* (C_r). Although M_c considers the cost factors, it does not take into account the total cost to detect all the faults in the software work product by inspections and testing. As such, we cannot compare the M_c values across different projects as shown using an example: Suppose two projects involve inspections and testing and that in both projects, if inspections had not been performed, the cost of testing would be 1000 units. The first project consumes 10 units for their inspections and saves 100 units. Thus, the total cost for fault detection is 910 units. In the second project, inspections cost 60 units and save 600 units. Thus, the total cost for fault detection is 460 units, which is far smaller than the cost in the first project of 910 units. However, the value of M_c in both projects is 10, which doesn't recognize the benefit of inspections in the second project. The *Kusumoto* metric [13] overcame this problem as discussed below.

Kusumoto Metric (M_k): Kusumoto et al., [13] defined the *cost effectiveness* of an inspection in terms of the reduction of cost to detect and remove all faults from the software product. M_k is calculated as a ratio of the reduction of the total costs to detect and remove all faults from the software product using inspections to the virtual testing cost (i.e., testing cost if no inspection is executed). The M_k normalizes the savings by the potential fault cost. Hence, it can be compared across different projects. M_k is intuitive and appropriate for this research as it can be interpreted in terms of fault rework savings due to inspections. Formally stated, the testing cost is reduced (by $\Delta C_t - C_r$) by performing inspections as compared to the testing cost ($Ct + \Delta Ct$) if no inspection is executed. Therefore, the M_k can be derived as: $M_k = (\Delta C_t - C_r) / (C_t + \Delta C_t)$ *(1)*

C. *Using Capture Recapture (CR) in Software Inspections*

As mentioned in Section II.A, this research uses the CR method to estimate the total fault count, which is then used to estimate the faults remaining post-inspection. Using these fault estimates, the M_k value is then computed using (1).

The use of CR method in biology makes certain assumptions that do not always hold for software inspections. The assumptions made by CR in biology include: 1) closed population (i.e. no animal can enter or leave), 2) equal capture probability (i.e. all animals have an equal chance of being captured), and 3) marks are not lost (i.e. an animal that has been captured can be identified) [8, 22]. When using the CR in inspections, the closed population assumption is met (i.e., all inspectors review the same artifact independently and it is not modified) and the assumption that marks are not lost is met (i.e.

TABLE I. CAPTURE-RECAPTURE MODELS AND ESTIMATORS [8, 10, 16, 18, 19, 22]

Model	Variation Source	Estimators Belonging to Each CR Model
M_o	All inspectors have the same ability, and all defects are equally likely of being detected.	Unconditional Likelihood (M_o-UMLE); Conditional Likelihood (M_o-CMLE); Estimating Equations (M_o-EE)
M_t	Inspectors differ in their abilities, but all defects are equally likely of being found.	M_o-UMLE; M_o-CMLE; M_o-EE
M_h	Inspectors have the same ability, but defects differ in their probability of being found.	Jackknife (M_h-JK); Sample Coverage (M_h-SC); M_h-EE
M_{th}	Inspectors differ in their ability, and defects differ in their probability of being found.	M_h-SC; M_h-EE

it can be determined if two people report the same fault). However, because some faults are easier to find than others and because inspectors have different fault detection abilities, the equal capture probability assumption is not met [18].

To accommodate these assumptions, 4 different CR models are built around the 2 sources of variation: *Inspector Capability* and *Fault Detection Probability*. Table I shows the four CR models along with their source(s) of variation. Each CR model in Table I has a set of estimators, which use different statistical approaches to produce the estimates. The estimators for each CR model are also shown in Table I. The mathematical details of CR estimators are beyond the scope of this paper [18, 22]. The input data used by all the CR estimators is organized as a matrix with rows that represent faults and columns that represent inspectors. A matrix entry is 1 if the fault is found by the inspector and 0 otherwise.

CR was introduced to software inspections by Eick, et al. by applying it to real fault data from AT&T. A major result from this study was the recommendation that an artifact should be re-inspected if more than 20% of the total faults remain undetected [5, 10]. Following this study, various empirical studies in SE have evaluated the use of CR models to accurately estimate the total fault count using artifacts with a known number of seeded defects [18]. In addition, our prior research evaluated the effect of inspection team size on the quality of the CR estimates [19]. A common finding across all the studies is that the estimators underestimate the actual fault count with small number of inspectors, but improve with more faults and inspectors. While there is evidence on the ability of the CR estimators to accurately estimate the total fault count [16], the CR research has neglected the cost spent and cost saved by an inspection. This research extends our prior work by evaluating the cost-effectiveness of software inspections using the CR estimates on real artifacts. These results will provide guidance on whether the CR estimators can be used to evaluate the cost-effectiveness of an inspection process in real software project where the fault count is unknown beforehand.

III. LITERATURE ON INSPECTION VS. TESTING COST RATIO

As mentioned earlier (section II.A), calculating the cost saved by the inspection requires a count of faults remaining post-inspection, and the average cost to detect a fault in the testing stage. While the CR method can be used to estimate the remaining faults, the average cost to detect a fault in the testing stage is not available at the end of inspection cycle. In this research, the average cost (time spent in hours) to find a fault during the testing, is calculated as a factor of the average cost (time spent in hours) to find a fault during the inspection. This section presents a summary of findings from different software organizations that reported the data on the cost (staff-hours) spent to find a fault during the inspections versus testing. This cost ratio is then used to calculate the cost-savings, the virtual testing cost, and the M_k value of the inspections using (1).

Major results regarding the cost spent (in staff hours) to find a fault during inspections versus the cost spent to find a fault during testing show a cost ratio of 1:6 [2, 4, 12, 17, 22]. These values are based on actual reported data. Also, Briand [6], based on the published data has provided probability distribution parameters for the average effort using different fault detection techniques according to which the most likely value for design inspections are 1.58 hours per fault and, for testing are 6 hours per fault. These values were derived from various studies on the cost of finding faults in design, code reviews and testing. Different studies in the literature give different estimates because of the differences in study settings, software processes, severity of the faults, review techniques and other factors. In order to find the most appropriate value, we computed the median of the reported cost ratio values resulting from precise data collection. We did not consider approximations, estimates, or data whose origins were unclear. As a result, the median cost ratio is 1:5.93. Therefore, for this research, the inspection to testing cost ratio of 1:6 was used to calculate the cost-effectiveness of the inspection process.

IV. STUDY DESIGN

Prior software engineering research has validated the fault detection effectiveness of inspections at the early stages of development [1,11]. However, there is a lack on empirical research on the benefits of inspections in terms of the extent to which the testing costs can be reduced by performing the inspections of early software artifacts. Furthermore, while inspections are effective; they cannot provide insights into the remaining faults which are required to calculate the fault rework savings. On that end, the CR method has been evaluated to provide a reliable estimate of the faults remaining post-inspection using artifacts with seeded faults [16]. However, there is a lack of research on the CR estimator's ability to provide a reliable estimate of the remaining faults when the actual fault count of the software artifact in not known beforehand. Therefore, this paper evaluates the ability of the CR estimator's to accurately predict the *cost-effectiveness* of the inspection process using the fault data from 4 real software artifacts that contained naturally occurring faults. In addition, each artifact was inspected twice, which allowed the analysis of the CR estimator's ability to accurately estimate the cost-effectiveness after each inspection cycle.

A. *Research Goal*

The main goal of this study is to evaluate the ability of CR estimators to provide an accurate cost-effectiveness value of an inspection process by comparing the M_k values based on the CR estimates against the actual M_k value after each inspection.

TABLE II. DATA SETS

Data Set	Artifact Name	Description	# of Inspectors	1st inspection Faults	2nd Inspection Faults	Total Faults
1	Starkville Theatre System	Management of ticket sales and seat assignments for the theatre	8	30	25	55
2	Management of Apartment and Town properties	Managing apartment and town property, assignment of tenants, rent collection, and locating property by potential renters	8	41	64	105
3	Conference Management	Helping the conference chair to manage paper submission, notification to authors, and other related responsibilities	6	52	42	94
4	Conference Management		6	64	54	118

B. *Data Set*

The data was drawn from earlier inspection studies conducted at Mississippi State University (MSU). The original goal of these studies was to investigate how the use of error information impacted requirements inspections [20]. Only the information relevant to our research analysis is presented here:

1) Software Artifacts and Software Inspectors: Inspection data from 4 artifacts used in this study were developed by senior-level students enrolled in the Software Engineering Design Course at MSU during two different years. The course required student teams to interact with real customers to elicit and document requirements that they would later implement. So, even though the developers were students, the artifacts are realistic for a small project. The subjects were divided into 4 teams (with 8, 8, 6, and 6 students respectively) that developed the requirement documents for their respective systems as shown in Table II.

2) Software Inspection Process: Each requirement document was inspected twice by the same subjects who created it. During the first inspection, the subjects received training on the use of a fault checklist. Then, each inspector individually inspected the requirements using the fault checklist and logged any faults identified. After the first inspection, the participants were trained on how to abstract errors from faults, how to classify the errors, and how to use the errors to re-inspect the requirements document. Then, each inspector re-inspected the requirements using the errors to find the additional faults. The artifacts were not modified or corrected between inspections (i.e., the same document was re-inspected). Therefore, we collected inspection data from the first inspection, the second inspection, and the total for each artifact. Note that the last three columns in Table II show faults found during the first inspection, during the second inspection, and total for all the 4 artifacts. Also, we collected the time spent (in hours) by each subject during the first and second inspection cycle and the total time for each artifact.

C. *Evaluation Procedure*

The following costs and savings for each artifact were computed after the first and second inspection cycle:

- *Average cost to detect a fault in inspection (c_r)*: Adding all the faults found by the inspectors, the average number of faults found by an inspector is calculated. From the available values of time taken by each inspector and the average number of faults found by an inspector, c_r is calculated as: $c_r = C_r / D_r$
  - The "C_r - *Inspection cost*", is calculated by adding the time spent by all the inspectors during the inspection.
  - The "D_r", is the total number of unique faults found by all the inspectors during an inspection cycle.
- *Virtual Testing Cost (C_{vt})*: is calculated as the product of the average cost to defect a fault in testing (i.e., c_t) and the total number of faults present in the product (i.e., D_{total}).
  - The "*c_t - Average cost to detect a fault in testing",* is calculated as 6 times of average cost to detect a fault during the inspection (c_r).
  - The "D_{total} - *Total fault count*", is the total number of faults present in the document.
- *Cost saved from inspection (ΔC_t)*: The testing cost saved from inspection is the product of the number of unique faults found during the inspection (D_r) and the average cost to find a fault during the testing (c_t). i.e., $\Delta C_t = Dr * c_t$.

The difference in the testing cost saved by the inspection and the cost spent on the inspection provides the reduction of the total costs. M_k value is then obtained as follows:

$$M_k = (\Delta C_t - C_r) / C_{vt}$$

This procedure was executed to calculate: (1) M_k using the actual fault count and (2) M_k using the CR estimates after each inspection for all the 4 artifacts. After the first inspection, the actual and estimated M_k values were obtained as follows:

1) M_k based on the actual fault count: Because we do not know the actual number of faults, the total number of exclusive faults found after both inspections is assumed to be the actual fault count for the purposes of this study. Therefore, for each artifact, the faults found during the first inspection (column 5 of Table II) by all the inspectors and the actual fault count (column 7 of Table II) were used to calculate the actual M_k value of the first inspection cycle.

2) M_k based on the CR estimates: For each artifact, the faults found during the first inspection by all the inspectors (column 5) are inserted into a matrix (as described in Section II.C). The matrix is then fed to the automated tool (CARE-2 [8]) to produce the estimates from all the CR estimators. Using the estimated fault count and the number of faults found during the first inspection, the M_k values were calculated.

Because the artifacts were unchanged between inspections, to calculate the cost-effectiveness after second inspection, we use the total faults found from both inspections (Column 7). Since we are calculating the testing cost saved by performing the inspections, it did not make sense to use only the data from second inspection. Furthermore, using only the data from inspection 2 would have excluded some information that is otherwise available while producing the CR estimates. Therefore, for each artifact, the faults found at the first and second inspection are inserted into a matrix which is then fed to the automated tool to produce the estimates from all the estimators. Using the estimated fault count, and the actual fault count (the number of faults found after both inspections), the estimated M_k values and the actual M_k value was calculated after the second inspection respectively.

D. Evaluation Criterion

The CR estimators are evaluated based on the "*relative error (R.E.)*" in the estimate of the cost-effectiveness after each inspection. For each artifact, and for each inspection cycle, the R.E in the M_k values based on the estimated fault count against the M_k values based on actual fault count was computed as:

Relative error = (M_k using the estimated fault count – M_k using the actual fault count) / M_k using the actual fault count

A R.E of zero means absolute accuracy (i.e., M_k value based on the estimated fault count is same as the M_k value based on the actual fault count), a positive R.E. means an underestimation, and a negative R.E means an overestimation of actual fault count. The accuracy of an estimate is satisfactory when the R.E. is within +/- 20% of the actual value [5, 10, 18].

V. RESULTS AND ANALYSIS

This section compared the actual and the estimated M_k values after each inspection cycle for each artifact in order to analyze whether the CR estimators can be used to make an objective and cost-effective post-inspection decisions.

To provide an overview of the results from Data Set 1, Fig. 2 compares the M_k values obtained using the actual fault count (of 55 faults) against the M_k values using the estimates from all the CR estimators after the first and second inspection. The results show an increase in the fault rework savings achieved by performing a re-inspection of software document (i.e., M_k value of 0.49 after second inspection is higher than M_k value of 0.38 after first inspection). This increase in the cost savings was consistent across all the four artifacts. These results show that the cost invested during the inspections of requirement documents help save costly rework later in the lifecycle.

Additionally, the result also shows a visible difference in the estimated M_k values and the actual M_k value after the first inspection. This is because the CR estimators underestimated the actual fault count after the first inspection which reduces the true virtual testing cost (i.e., C_{vt}, the demoniator of *Kusumoto* metric formula), and thereby returns a higher M_k value. After the second inspection, the estimated M_k values are closer to the actual M_k value because the quality of the CR estimates improved after the second inspection (i.e., the CR estimates of the fault count are closer to the actual fault count).

To quantify the results in Fig. 2, we calculated the R.E. in the M_k values produced by each CR estimator after each inspection cycle as shown in Fig. 3. The dashed lines in Fig.3 show the region of +/- 20% within which the estimation results are considered satisfactory. A major observation from Fig. 3 is that out of all the CR estimators, only the M_{th}-SC estimator produced a satisfactory estimate (i.e., M_k value within +/- 20% R.E. range) after each inspection. Based on the results from Data set 1, we recommend the M_{th}-SC for estimating the total fault count to decide the need for a re-inspection, and for evaluating the cost-effectiveness of the inspections.

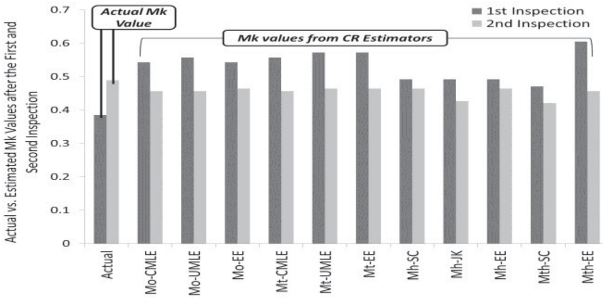


Figure 2. Estimated vs. Actual M_k Values for Data Set 1

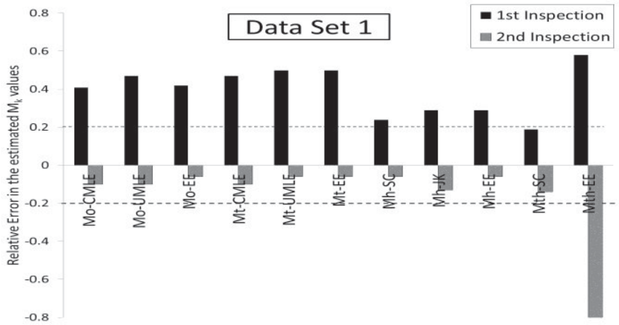


Figure 3. R.E. in the Estimated M_k Values after Each Inspection Cycle

Similar process was replicated to compute the R.E. in the M_k values from all the CR estimators for Data Sets 2, 3, and 4 as shown in Fig. 4. Similar trends were observed across all the four data sets and are summarized as follows:

- For each artifact, the actual M_k value after second inspection is always higher than the actual M_k value after the first inspection. This confirms that re-inspection of software artifacts was a cost-effective decision;
- Across all the four data sets, only the M_{th}-SC estimator produced an accurate estimate of the M_k value (within +/- 20% of the actual) after each inspection cycle. The M_h-SC estimator was the second best estimator;
- The JK estimator is not recommended because of its inability to provide satisfactory estimate for Data Sets 3 and 4. Furthermore, M_{th}-EE is not recommended because of its failure to produce an estimate for Data Set 3.

Therefore, based on these results, SC are the best estimators for evaluating the cost-effectiveness of software inspections.

VI. THREATS TO VALIDITIY

There were some threats to validity that were not completely addressed. The actual number of faults present in each document might actually be higher than the assumed fault count (i.e., the total faults found after two inspections). This threat is somewhat reduced by the fact that the subjective opinion of inspectors regarding the remaining faults after each inspection cycle (which was all that was available during the original study) supported the recommendation that they had

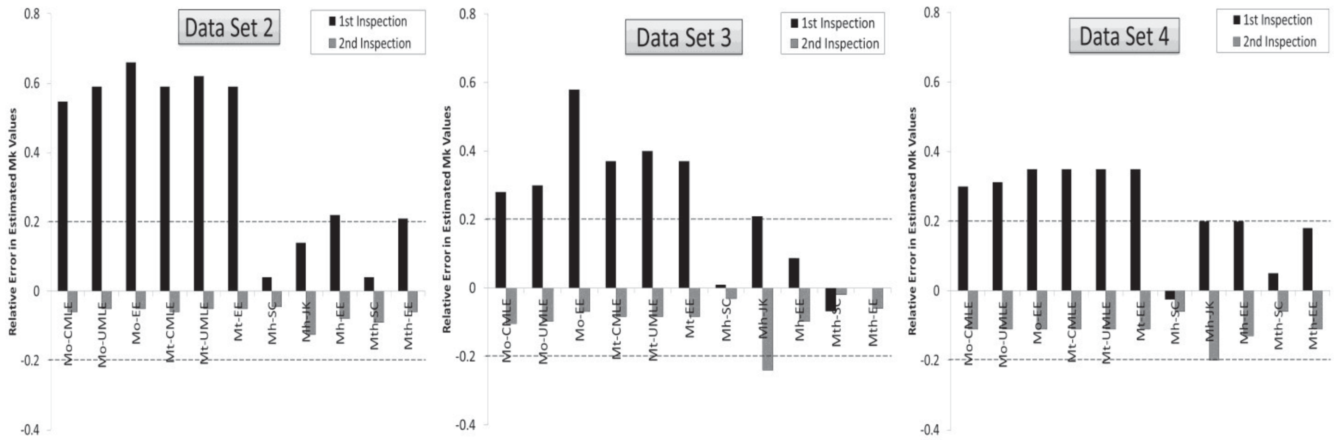


Figure 4. Relative Error in the Estimated M_k Values after the First and Second Inspection for Data Seta 2, 3, and 4

located all the faults present in the documents during the second inspection. So, the inspection process was stopped. A second threat was the artifacts were developed by students and may not be representative of the industrial strength documents.

VII. Discussion Of Results

The results show that some of the CR estimators underestimate the actual fault count, and hence overestimate the actual M_k value after the first inspection. Only the SC estimator for M_{th} model provided an accurate estimate (i.e., +/- 20%) of the total number of faults and hence the remaining faults after the first and second inspection. Consequently, only the M_{th}-SC estimator provide an accurate estimate of the cost-effectiveness (i.e., +/- 20% of the actual M_k) after each inspection cycle. Therefore, project managers can use the M_{th}-SC estimator to estimate the faults remaining after the inspection and decide a need for re-inspection. Additionally, the M_{th}-SC estimator should be used to analyze if the significant fault rework savings are being achieved by performing the inspections to decide when to stop the inspection process. Project managers can also use these research results to motivate the use of the inspections of early software documents in their companies and thereby avoid the costly rework during the later stages of software development.

VIII. Conclusion and Future Work

This paper demonstrated that the *Kusumoto* metric can be used in conjunction with the CR estimate of the remaining faults after an inspection to manage the quality of software product being developed. Our future work in this area will investigate the effect of other factors (e.g., inspection team size, time duration etc) to improve the cost-effectiveness of software inspections. Also, further research will investigate the use of Kusumoto metric and the CR method using industrial strength document (other than the requirement documents), that have been developed and inspected by software professional (as opposed to the students) in real environment.

References

[1] Ackerman, A., Buchwald, L., and Lewski, F., “Software Inspections: An Effective Verification Process.*” IEEE Software*, 1989. **6**(3): 31-36.

[2] Boehm, B.: *Software Engineering Economics*. Prentice-Hall, 1981.

[3] Boehm, B. and Basili, V.R., "Software Defect Reduction Top 10 List." IEEE Computer, 2001. 34(1): 135-13.

[4] Biffl, B. Freimut, O. Laitenberger, “Investigating the Cost-Effectiveness of Reinspection in Software Development”, Proceedings of the 23rd International Conference on Software Engineering (2001) 155-164.

[5] Briand, L.C, Emam, K.E, and B.G.Freimut. “A Comparison and Integration of Capture-Recapture*“. International Symposium on Software Reliability Engineering*. 1998. Paderborn, Germany: 32-41.

[6] Briand, L.C, Emam, K.E., Laitenberger, Fussbroich, Using Simulation to Build Inspection Efficiency Benchmarks for Development Projects, International Conference on Software Engineering, pp. 340-349, 1998.

[7] Burnham, K.P. and Overtom, W.S., “Estimation of the Size of a Closed Population When Capture Probabilities Vary Among Animals.*” Biometrica*, 1978. **65**: 625-633.

[8] Chao, A. and Yeng, H.C., Program CARE-2 (for Capture-Recapture Part.2), http://chao.stat.nthu.edu.tw, 2003.

[9] Collofello, J.S., Woodfield, S.N., Evaluating the Effectiveness of Reliability-Assurance Techniques, Journal of Systems and Software. Vol. 9 (3) (1989) 191-195.

[10] Eick, S., Loader, C., Long, M., Votta, L., and Weil, S.V. “Estimating Software Fault Content Before Coding*”*. International Conference on Software Engineering. 1992. Australia: ACM Press: 59-65

[11] Fagan, M. E., “Advances in Software Inspections,” *IEEE Transactions on Software Engineering*, Vol. SE-12, No. 7, July 1986, pp. 744-751.

[12] Gilb, T., Graham, D. Software Inspection, Addison-Wesley, 1993.

[13] Kusumoto, T, Matsumoto, K., Kikuno, T., Torii, K., A New Metrics for Cost Effectiveness of Software Reviews, IEICE Transactions on Information and Systems E75-D (5) (1992) 674-680.

[14] Madachy, R., “Process Improvement Analysis of a Corporate Inspection Program,” *Software Engineering Process Group Conference,* MA, 1995.

[15] Meyer, G., “A Controlled Experiment in Program Testing and Code Walkthroughs/Inspections”. In *Communications of the ACM*, 21(9):760-768, September 1978.

[16] Miller, J., 1999. “Estimating the Number of Remaining Defects after Inspection”. Software Testing, Verification, Reliability, 9(3), 167-189.

[17] Olson, T., “Piloting Software Inspections to Demonstrate Early ROI,” Notes from Presentation given at the 1995 SEPG Conference

[18] Petersson, H., Thelin, T., Runeson, P., and Wohlin, C., “Capture-Recapture in Software Inspections after 10 Years Research - Theory, Evaluation and Application.*” Journal of Systems and Software*, 2003.

[19] Walia, G., Carver, J. and Nagappan, N. “The Effect of the Number of Inspectors on the Defect Estimates Produced by Capture-Recapture Models.” Proceedings of the 30th International Conference on Software Engineering. May 10-18, 2008. Leipzig, Germany. p. 331-340

[20] Walia, G., Carver, J. "Using Error Abstraction and Classification to Improve Requirement Quality: Conclusions from a Family of Four Empirical Studies." *Empirical Software Engineering: An International Journal*. 2012. DOI: 10.1007/s10664-012-9202-32012.

[21] Weller, E. “Lessons from Three Years of Inspection Data”. In *IEEE Software*, 10(5):38–45, September 1993.

[22] White, G.C., Anderson, D.R., Burnham, K.p., and Otis, D.l., Capture-Recapture and Removal Methods for Sampling Closed Populations, Los Alomos National Laboratory, 1982.

Requirement Analysis and Automated Verification: A semantic approach

Animesh Dutta
Department of Information technology
National Institute of Technology
Durgapur
India
Email:animeshrec@gmail.com

Prajna Devi upadhyay
Department of Information technology
National Institute of Technology
Durgapur
India
Email: kirtu26@gmail.com

Sudipta Acharya
Department of Information technology
National Institute of Technology
Durgapur
India
Email: sonaacharya.2009@gmail.com

***Abstract*—In this paper, we propose an automated software development methodology. The methodology is conceptualized with the notion of agents, which are autonomous goal-driven software entities. Initially, the requirements of the newly proposed system are captured from stakeholders which are then analyzed in goal oriented model and represented as goal graph. The leaf level atomic sub goals of the goal graph are taken as input to our automated system. These atomic sub goals are mapped to concepts in Domain Ontology and are verified with the help of three verification metrics- Completeness, Correctness, and Conflict. We also develop a software tool for the proposed automated system.**

I. Introduction

Requirements engineering is an important phase of the software development methodology. Requiremets should be documented as the SRS(Software Requirement Specification), which is the input to the design phase. To ensure that the requirements are in correct form, requirement analysis and verfication process should detect incomplete, incorrect and ambiguous requirements. Nowadays, requirement verfication techniques supported by domain knowledge are in use which check the specified requirements against the domain knowledge for inconsistencies and help the system analyst to remove them. Ontology [1] is the most suitable representation of domain knowledge because concepts, relationships and their categorizations in a real world can be represented with them. It also allows for semantic processing of requirements. The automation of the requirement verification process complements transformation systems utilised for Agent Oriented Software Engineering(AOSE) methodologies [2], [3], [4]. Transformation systems generate system architecture from the specified requirements automatically by consulting with Domain Ontology.

II. Related Work

There have been a significant number of contributions in the area of Ontology based requirement verification. In [5], [6], [7] authors have used Domain Ontology to express domain knowledge and have used it for the semantic verification of requirements using metrics. The tool development for the proposed methodologies is not shown. [8] This paper proposes a novel user-driven requirements elicitation process UDREP. Based on users individualities and context, support for users to present requirements is provided, including the recommendation of domain assets and advice about multi-user cooperation. It lacks of verification metrics to measure the quality of requirement specification. In [9], authors have proposed AGORA, a method for requirements elicitation which is an extended version of a goal-oriented method by attaching preference and contribution attributes to an AND-OR graph. They have also defined some metrics to measure the quality of requirement specification. In [10], authors propose a method called GOORE where a domain ontology is utilized to support goal decomposition as domain knowledge. A tool has also been developed to facilitate the same. Some more tools to facilitate verification have been developed in [11], [12]. These tools are not the part of any automated system.

III. Scope of Work

There have been a lot of contributions in the area of Ontology based requirements verification techniques. But, most of them are a part of traditional software development methodologies. Also, very few contributions have elicited requirements verification approach for automated systems. Although automated software development methodologies reduce the gap between requirements specification and system design, there is a need to incorporate a requirements verification module to determine whether user requirements have been fully expressed or not in such systems. The lack of verification in traditional as well as automated software development methodologies lead to inconsistent systems which may not adhere completely to user's requirements.

This paper presents requirements verification approach for the automated software development methodology using agents which we proposed in [13]. The proposed methodology takes the requirement represented as goal graph, the domain knowledge represented as Ontology and determines whether the requirements have been expressed properly. The verification is performed against domain Ontology and the result is expressed in the form of three metrics-Completeness, Correctness, and Conflict. A software tool is also developed to facilitate automation of the verification procedure of the input requirements.

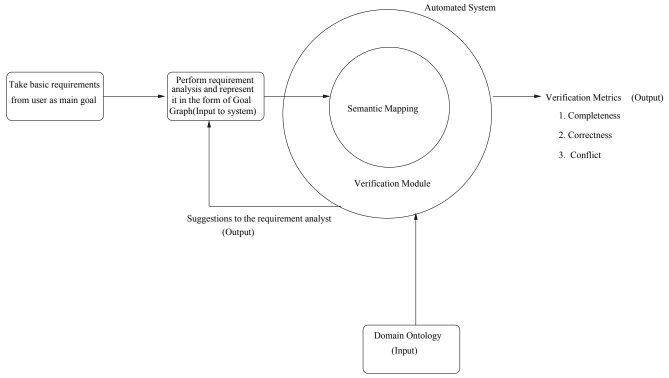

Automated System
Take basic requirements from user as main goal
Perform requirement analysis and represent it in the form of Goal Graph(Input to system)
Semantic Mapping
Verification Module
Verification Metrics (Output)
1. Completeness
2. Correctness
3. Conflict
Suggestions to the requirement analyst (Output)
Domain Ontology (Input)

Fig. 1: Architecture of the proposed Automated System

IV. Proposed Methodology

Figure 1 represents the architecture of our proposed automated system. In [13], we have proposed the architecture of an automated system which generates the system design from the requirement specification by consulting the Domain Ontology of the system. But, this architecture lacks to verify input requirements. Designing the architecture of the system without ensuring that the requirements are in correct form may lead to the development of inconsistent systems. So, the verification module has also been included to be a part of our automated system. Thus, the automated system takes the requirement and the Domain Ontology as input, verifies these requirements against Domain Ontology, and suggests the system analyst to change the specification accordingly. In our automated system, we have peformed goal oriented requirement analysis and represented the requirements in the form of Goal Graph. The Domain knowledge is represented as Domain Ontology. The leaf level sub goals are taken as input to the automated system and mapped to concepts in Domain Ontology and verified against them.

A. Requirements represented by Goal Graph and Domain Knowledge represented by Ontology

Agents in MAS are goal oriented i.e. all agents perform collaboratively to achieve some goal. Goals have been used in Software Engineering to model requirements and non-functional requirements for a software system. Goal Graph and Domain Ontology have been defined in [13].

B. Semantic Mapping from requirements to Ontology

The process by which the basic keywords of the leaf node sub goal are mapped into concepts in the Ontology is called Semantic Mapping. In this paper, the aim of semantic mapping is to find out tasks from Domain Ontology, required to be performed to achieve a sub-goal given as input from the Goal Graph. Let a user requirment R come to our proposed system. After goal oriented requirement analysis of requirment R, we represent it as a goal graph and get a set of leaf level indivisible subgoals. These leaf level sub goals are represented as:

G_0= $\{G_1$, G_2,$\ldots$,G_p $\}$.

Let there be a set denoted as G=$\{g_1$,g_2, $\ldots$,g_q $\}$. Each $g_i \in$ G is another set which consists of set of goal-concepts which are associated with a consists-of relationship in ontology. Let us denote each $g_i \in$ G as ”concept-set”.

Let Ky be a function that maps a subgoal to its basic keyword set. The set of keywords for subgoal $G_i \in G_0$ can be represented by Ky(G_i) = $\{Ky_{i1}$,$Ky_{i2} \ldots Ky_{ij}\}$. Let f be a mapping which maps each Ky(G_i) or some Ky(G_i) $\cup$ Ky(G_j) $\cup \ldots \cup$ Ky(G_k) to a concept-set in ontology, $G_i \in G_0$ where $1 \leq i \leq k$ or $\{G_i$, G_j ,$\ldots$, G_k $\}\subseteq G_0$.

i.e. f(Ky(G_i)) = g_i, $g_i \in$ G, or there exists a subset of G_0, $\{G_1$, G_2, $\ldots$, G_k $\}\subseteq$G$_0$ such that f(Ky(G_1) $\cup$ Ky(G_2) $\cup \ldots \cup$ Ky(G_k)) = g_i.

Each subgoal $G_i \in G_0$ or some set $\{G_i$, G_j ,$\ldots$, G_k $\}\subseteq G_0$ is mapped on some concept-set $g_i \in$ G.

C. Verification metrics of requirement analysis

In this paper we have defined three metrics for verification of requirment analysis.

- Completeness
- Correctness
- Conflict

1) Completeness:

- Case 1: Each keyword in the keyword set KY= Ky(G_1) $\cup$ Ky(G_2) $\cup \ldots \cup$ Ky(G_p) is mapped into some concept $\in$ concept-set g_i of ontology. If a keyword $\in$ KY can be mapped to a concept in concept set g_i of ontology, then add the keyword to set S and add the concept set g_i to ST. Thus, ST=ST $\cup$ g_i, where initially ST is the empty set.
  Repeat this step for every keyword in KY. Finally, the Measure of Completeness, MCOM = |S|/|T|
- Case 2: Let G=$\{g_1$,$\ldots$, g_q $\}$be the set of concept-sets on which G_0 is mapped. Let T_i be the set of tasks associated with $g_i \in$ G with consists_of relation. So, T_0= T_1 $\cup$ T_2 $\cup \ldots \cup$ T_q is the set of all tasks that are required to be performed to achieve the goal set G_0.
  Let pred(t_i) represent the set of tasks that should happen before t_i and succ(t_i) represent the set of tasks that happen after t_i. So, for each task $t_i \in T_0$, check the following condition
  $\{$pred(t_i)$\subset T_0 \bigwedge$ succ(t_i) $\subset T_0$ $\}$= true.
  For all t_i where the above condition is false , let pred(t_i) $\in$T’$_i$ or succ(t_i) $\in$T’$_i$. Let T'_i be associated with $g'_i \in$ G by the consist_of relation. The sub goal formed by the concept-set g_i should be included in the suggestion list.

2) Correctness::

- Case 1: Each keyword in the keyword set KY= Ky(G_1) $\cup$ Ky(G_2) $\cup \ldots \cup$ Ky(G_p) should be mapped into some concept $\in$ concept-set g_i of ontology. So, total number of keywords in requirement analysis is m2= $|$ (Ky(G_i) $\cup$ Ky(G_j) $\cup \ldots \cup$ Ky(G_k))$|$. If a keyword $\in$ KY cannot be mapped to a concept in concept set g_i of ontology, then add the keyword to set US ,where initially US is the empty set.

Repeat this step for every keyword in KY.

- Case 2: Let $G_i \in G_0$ be mapped on concept-set g_i,and to achieve G_i set of tasks T_i will be performed.i.e. g_i concept is associated with T_i by consists-of relationship. Let G_i have some happened before relationship with other subgoals of set G_0. Let $G_j \rightarrow G_i \rightarrow G_k$ i.e. G_j is predecessor and G_k is successor of G_i. Then requirement analysis will be considered as correct if same happened-before relationship exists between corresponding task sets in ontology. i.e. there exists $T_j \rightarrow T_i \rightarrow T_k$. If $\exists$ G_i for which uppper condition is not satisfied then it can be considered as redundant subgoal in goal graph after requirment analysis and can be discarded from goal graph. In this case, add the keywords of goal G_i to the set US.
- Case 3: Let there is a goal $G_i \in G_0$ which has no happened before relationshiop with other goals of G_0. Similarly in ontology, its corresponding task set T_i also has no happened-before relationship with other corresponding task set of goals of G_0 in ontology. Then subgoal G_i can be added to the suggestion-list of our proposed system for requirement analysis stating that it may be incorrect/irrelevant subgoal in goal graph after requirement analysis. So analyst is recommended to verify whether this subgoal should be included in requirement specification.

  Correctness can be measured by,

  n2=total number of keywords in correctly mapped leaf level subgoals.

  m2=total number of keywords in leaf level subgoals in goal graph.

  Mathematically n2=| (Ky(G_i) $\cup$ Ky(G_j) $\cup \ldots \cup$ Ky(G_k))| - | US |

  m2= | (Ky(G_i) $\cup$ Ky(G_j) $\cup \ldots \cup$ Ky(G_k))|

  So, Measure of Correctness, MCOR = n2/m2

3) *Conflict:* Let from user side requirement R come to our proposed system. After requirement analysis of requirement R, we get a goal graph with a set of leaf level indivisible subgoals represented as a set $G_0= \{G_1$, $G_2,\ldots,$ G_p $\}$The necessary condition to occur conflict is,

For any two G_i, $G_j \in G_0$ f(Ky(G_i)) $\cap$ f(Ky(G_j)) $\neq \emptyset$ where i $\neq$ j.

If necessary condition is true then evaluate

A = f(Ky(G_i)) - f(Ky(G_j)) B = f(Ky(G_j)) - f(Ky(G_i))

- Case 1: If both sets A and B each contain single numerical value then both G_i and G_j are conflicting if both numerical values are different.i.e. these two subgoals represent one same fact but by different numerical values.
- Case 2: Now check whether for some (a_i, a_j) $\in A\times$ B, conflict(a_i)=a_j or conflict(a_j)=a_i. If yes, goal G_i and a_j are in conflict, i.e. conflict(G_i, G_j)=true
- Case 3: Let after mapping of subgoals of G_0 to the ontology the set of tasks required to perform to achieve all subgoals of G_0 can be represented by the set T_0 = $\{T1$, $T_2,\ldots,$ T_q $\}$. Now if for any pair (T_i, T_j) $\in$ T_0 X T_0, conflict(T_i)=T or conflict(T) = T_i,then those two subgoals associated to corrosponding two tasks are conflicting. Let n3= total number of (G_i, G_j), $G_i \neq$G$_j$, G_i, G_j $\in$G$_0$ for which conflict(G_i,G_j)=true.

  Let m3= total number of (G_i, G_j), $G_i \neq$G$_j$, G_i, G_j $\in$G$_0$

  So, Measure of Conflict, MCON = n3/m3

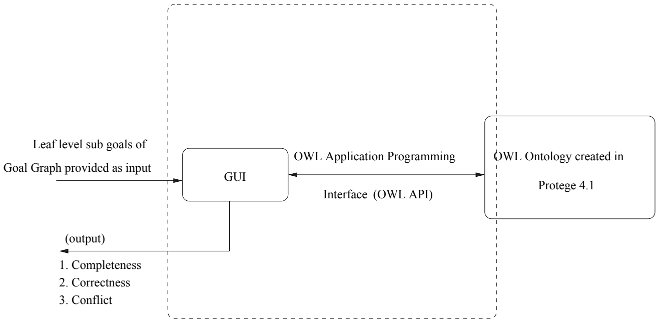


Fig. 2: Architecture of the tool for Automated Requirements System

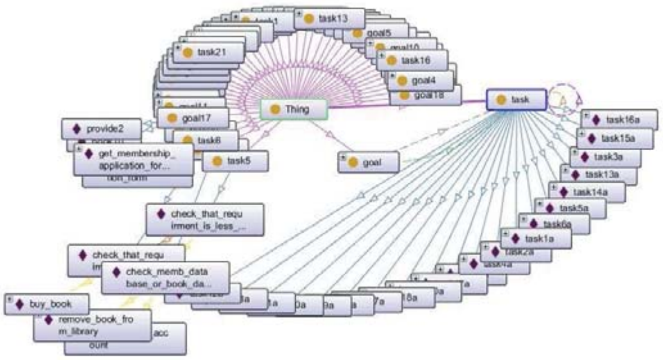


Fig. 3: Snapshot of the Ontology created in Protege 4.1

V. Tool Development

We have shown the architecture of the tool that has been developed in Figure 2.

A. *OWL Ontology of Library Management Systemin Protege 4.1*

The ontology for Library Management System(LMS) is built in Protege 4.1. Protege [15] is an open source tool to build OWL ontologies. An owl ontology consists of individuals, Properties and classes. Individuals represent objects in the domain of interest. Properties are binary relations on individuals i.e. properties link two individuals together. OWL classes are interpreted as sets that contain individuals. They are described formally stating the requirements for the membership of the class. A snapshot of the ontology for LMS created in Protege 4.1 is shown in Figure 3.

B. *OWL Application Programming Interface(OWL API)*

The OWL API [16] is a java Interface and implementation for the W3C Web Ontology Language(OWL), used to represent Semantic Web Ontologies. It consists of the following components.

- An API for OWL 2 and an efficient in-memory reference implementation.
- RDF/XML parser and writer
- OWL/XML parser and writer
- OWL Functional Syntax parser and writer
- Turtle parser and writer
- KRSS parser
- OBO Flat file format parser
- Reasoner interfaces for working with reasoners such as FaCT++, HermiT, Pellet and Racer

We have used the OWL API to connect to the OWL Ontology that we have created. The requirements gathered from the system analyst will be verified against the ontology at the back end using this API. It provides a number of packages to help perform required operations. We have used the org.semanticweb.owlapi.* package to verify the requirements against the ontology.

C. *Graphical User Interface*

To capture the leaf level atomic requirements represented as sub goals, a Graphical User Interface has been built. It takes the leaf level sub goals as input and returns to the user whether these are complete, correct or conflict free. It specifies upto what extent the requirements are complete, correct or conflict free.

VI. Conclusion

In this paper we have proposed a methodology for an automated system to verify the requirment analysis done by the system analyst. The requirement analysis is goal oriented and represented by goal graph.We have considered three metrics, Completeness, Correctness and Conflict to verify the requirement specification and express its correctness in percentage. We have also developed a tool to facilitate the verification procedure. The future prospect of this work is to propose a methodology to verify the system design. Since we are concerned with agent and MAS architecture, we have to develop a verfication methodology for the same. Thus, we will develop a tool that will not only generate the MAS architecture from the requirement specification automatically, but will also verify the requirement specification and the system architecture.

References

[1] K. K. Breitman and J. C. S. do Prado Leite, ”Ontology as a Requirements Engineering Product”, In 11th IEEE International Requirements Engineering Conference (RE03),pages 309319, Sep. 2003.

[2] N.R. Jennings, ”On Agent-Based Software Engineering”, Artificial Intelligence, vol. 177, no. 2, 2000, pp. 277-296.

[3] J. Lind, ”Issues in Agent-Oriented Software Engineering”, In P. Ciancarini , M. Wooldridge (eds.), Agent-Oriented Software Engineering: First International Workshop, AOSE 2000. Lecture Notes in Artificial Intelligence, Vol. 1957. Springer-Verlag, Berlin

[4] M. Wooldridge, P. Ciancarini, ”Agent-Oriented Software Engineering: the State of the Art”, In P. Ciancarini, , M. Wooldridge, (eds.), Agent-Oriented Software Engineering: First International Workshop, AOSE 2000. Lecture Notes in Artificial Intelligence, Vol. 1957. Springer-Verlag, Berlin Heidelberg (2001) 1-28

[5] Haruhiko Kaiya, Motoshi Saeki, ”Using Domain Ontology as Domain Knowledge for Requirements Elicitation”, 14th IEEE International Requirements Engineering Conference (RE’06)

[6] Haruhiko Kaiya and Motoshi Saeki, ”Ontology Based Requirements Analysis: Lightweight Semantic Processing Approach”,In Proc. of QSIC2005, pages 223230, Sep. 2005.

[7] Haibo Hu, Lei Zhang, Chunxiao Ye, ”Semantic-based Requirements Analysis and Verification”, In 2010 International Conference on Electronics and Information Engineering (ICEIE 2010), 2010

[8] Shu Fengdi,etal. ”User-Driven Requirements Elicitation Method with the Support of Personalized Domain Knowledge”, Journal of Computer Research and Development,2007, 44 (6) : pp1044-1052

[9] H. Kaiya, H. Horai, and M. Saeki, ”AGORA: Attributed Goal-Oriented Requirements Analysis Method”, In IEEE Joint International Requirements Engineering Conference,RE02, pages 1322, Sep. 2002.

[10] Masayuki Shibaoka, Haruhiko Kaiya, and Motoshi Saeki, ”GOORE : Goal-Oriented and Ontology Driven Requirements Elicitation Method”, J.-L. Hainaut et al. (Eds.): ER Workshops 2007, LNCS 4802, pp. 225234, 2007.

[11] M. Kitamura, R. Hasegawa , H. Kaiya, M. Saeki, ”An Integrated Tool for Supporting Ontology Driven Requirements Elicitation”, In: ICSOFT 2007. Proc. of 2nd International Conference on Software and Data Technologies (2007)

[12] Kitamura, M., et al., ”A Supporting Tool for Requirements Elicitation Using a Domain Ontology”, In Proceedings Software and Data Technologies (2009)

[13] Prajna Devi Upadhyay, Sudipta Acharya, Animesh Dutta, ”Automated Software Development Methodology: An agent oriented approach”, In The 8th International Conference on Computing and Information Technology, IC2IT 2012, Thailand, 9-10 May, 2012(Accepted for Publication)

[14] P.H.P. Nguyen, D. Corbett, ”A basic mathematical framework for conceptual graphs”, In IEEE Transactions on Knowledge and Data Engineering, Volume 18, Issue 2, 2005.

[15] Matthew Horridge, Simon Jupp, Georgina Moulton, Alan Rector, Robert Stevens, Chris Wroe, ”A Practical Guide To Building OWL Ontologies Using Protege and CO-ODE Tools”, Edition 1.1, The University Of Manchester, October 16, 2007.

[16] http://owlapi.sourceforge.net/

Risk-driven Non-functional Requirement Analysis and Specification

Yi Liu[1,2], Zhiyi Ma[1,2], Hui Liu[1,3], Weizhong Shao[1,2]

[1] Key Laboratory of High Confidence Software Technologies, Ministry of Education,

[2] School of Electronics Engineering and Computer Science, Peking University, Beijing, 100871, China

[3] Beijing Lab of IntelligentInformation Technology, School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100081, China

{liuyi07, mzy} @sei.pku.edu.cn, liuhui08@bit.edu.cn, wzshao@pku.edu.cn

Abstract—**The complexity and usefulness of software systems are determined not only by their functionality, but also by non-functional requirements such as accuracy, security, cost, user-friendliness and performance. However, even with the growing interest in dealing with NFRs from early stages of software development, current technology is not adequate for analyzing and representationally expressing these requirements. They mainly focus on refinement of NFRs themselves, neglecting the analysis of factors or risks in the environment that may have adverse impacts on these NFRs, which may lead to incomplete understanding and prevent the effective processing of NFRs. In this paper, we propose a risk driven approach to analyzing and specifying NFRs, in which NFRs are considered to address business risks. We first elaborate the aspects that should be considered when understanding NFRs. We also propose constrained case diagram to model the analysis result and their impacts on target system. This approach provides us an operating procedure and a complete NFR view. It could complement with existing approaches to more completely analyze and refine NFRs.**

Keywords-non-functional requirements; risk driven; analysis; specification; software development

I. INTRODUCTION

The complexity and usefulness of software systems are determined not only by their functionality, but also by Non-Functional Requirements (NFRs) such as accuracy, security, reliability, user-friendliness, performance [1][2]. Due to the importance of NFRs, researches have been done on how to achieve them in software systems. Generally, two main approaches were defined: curative approach and preventive approach. A curative approach is applied after a software system has been developed; the software system is tested and defects associated with NFRs are identified in order to be fixed. In a preventive approach, the software development tasks (e.g. constructing architectures and design models) are planned to prevent defects associated with NFRs, incorporate NFRs into the design and implementation, and finally obtain a system which satisfies the NFRs [2].

In both approaches, especially the preventive approach, a concise specification of NFRs is an important premise. Traditionally, NFRs are just simply and textually denoted in a distinctly section of the requirement specifications [3], e.g., "The system shall be available 24×7, give users a one-second response time, and process 200 orders per minute". These specifications are remote from the job of the software and it hard to tell that how developers are supposed to react to them and take which steps to achieve them. Later, goal-oriented approaches [4][5] have been proposed to further analyze and operationalized NFRs, in which NFRs are taken as softgoals. Scenario-based approaches [6][7] are also proposed to describe NFRs using scenarios.

However, these approaches mainly focus on refinement of NFRs themselves in the scope of software, neglecting the analysis of the factors/risks in the environment that would have adverse impacts on them, which may lead to incomplete understanding and inappropriate solutions of NFRs. Additionally, corresponding models are just represent NFRs and their relationships with functionalities, the derived requirements which will be directly designed and implemented to support the NFRs are always ignored or denoted implicitly. These incomplete models may prevent effective handling of NFRs.

In this paper, we propose a risk-driven approach to stepwise analyze and refine NFRs. Besides breaking down a specific kind of broad non-functional goal into clear requirements that are as detailed as developer need, we also identify origins of NFRs; analyze the external and internal challenges for achieving them; and recommend tactics based on the previous analysis. A constrained case diagram is also proposed to model the analysis results, so as to effectively deal with them in the following software development lifecycle.

Main contributions of our approach include: (1) Support the capture of NFRs based on risks, identify the threats and challenges, which allows the business and developers to understand the origins of NFR and determine the appropriate solutions; (2) Provide an operational guideline to thoroughly analyze NFRs, follow which we could obtain a more complete understanding NFR origins, challenges need to be tackled and other context; (3) Allow the developers to fully capture the requirements by providing an integrated graphical view about NFRs, the derived requirements, the functional requirements affected by them and relationships among them. (4) Moreover, this approach could complement with existing analysis approaches to better deal with NFRs.

The rest of the paper is organized as follows. Section II introduces related works and our basic ideas. Section III elaborates the risk driven NFR analysis approach and the proposed constrained cased diagram. Section IV introduces a case study. Section V discusses the proposed approach and concludes the paper.

II. Background and Related Works

A. Non-Functional Requirements Analysis Approaches

To effectively deal with NFRs in software development, specifying them in detail and accurately is an important premise. Since initial descriptions of NFRs are often roughly and non-behavioral, they are hard to tell be realized like functional requirements. Therefore, a further analysis to break down a specific kind of broad non-functional goal into clear requirements that are as detailed as developer need, and accurately record and manage these requirements throughout the life cycle of systems development is needed.

Goal-oriented methods, such as the NFR Framework proposed by Chung et al [4], the i*/Tropos approach [5], are typical representatives of these works. NFRs are taken as softgoals, gradually refined to lower, more-precise level of detail either by the type or functional topic, and finally refined to operationalizations. However, these approaches mainly focus on the refinement of NFR themselves in the scope of target software, lack of adequate analysis of the environment entities. Scenario based approaches are also proposed to deal with NFRs [6][7][8]. However, most of them focused on the representation of NFRs not the analysis process. Moreover, nearly all the approaches are based on the premise that NFRs are already identified, lack of the considerations about the source and origin of NFRs, which makes it hard to determine their priorities and may lead to improper solution recommendations. Thus, besides refining NFRs themselves, a more thorough analysis about the motivations of NFR and the factors in the environment that would have adverse impacts on NFRs are also needed, in order to effectively dealing with NFRs in software development lifecycle.

In security engineering [9], risk assessment is an important task. It is defined as a process of identifying the risks to system security and determining the probability of occurrence, the resulting impact and additional safeguards that would mitigate this impact [10]. This process conducts analysis about security requirement from various aspects, and provides solutions to achieve them. It is also used in Software Performance Engineering (SPE) to help analysis of performance [11]. Through analysis in practice and surveying existing literatures, we found the idea of risk assessment could be also applied for analyzing other NFRs such as availability, reliability. NFRs could be considered as a set of quality statements that the system must meet in order to manage the risks exposed to system stakeholders, where these business risks may originate from low quality of services under different operating conditions [12]. For example, the business may be at risk of: customer retention if the system is running under poor performance or response times; loss of revenue if the system does not provide sufficient availability of its services, and to mitigate these risks, corresponding performance requirement, availability requirement should put on the table and find solutions to deal with them. Based on this idea, we could more fundamentally know much about the NFRs, thus, conducting more thorough analysis of them.

In this paper, we propose a risk driven approach to analyzing and refining NFRs. It draws lessons from the idea of risk assessment and management [9][10], and will complement with existing approaches to more adequately analyze NFRs.

B. Specifying and Modeling NFRs

Traditionally, during the requirement phase, NFRs are collected and documented in some textual format. They are specified in a separate section of the documentation, as an entry of the functionality description, or recorded in an artifact called supplementary specification [3]. When use case models become popular to model functional requirements, approaches have been proposed to extend them to address NFRs. Armour and Miller [13] discuss how one may document the ‘non-behavioral requirements’ using a use case, immediately after the Alternative Flows and Exception Section. Supakkul and Chung [8] represent NFRs as softgoals and associate them with appropriate use case model elements. Abuse case and Misus e case is proposed which extends the use case to define a security requirement [6].

However, these approaches mainly focus on NFRs themselves and their constraint relationships with functionalities. The derived requirements identified to achieve or support NFRs are often neglected or just implicitly represented in the models. Since NFRs are non-behavioral in nature, corresponding tactics for achieving them should be identified first in order to make them operational. Thus, just graphically representing FRs and NFRs together is not enough and relatively incomplete. In this paper, based on the proposed risk driven NFR analysis approach, we propose a constrained case diagram to specify NFRs. Besides associating NFRs with related use case, it also represents the derived requirements for achieving the NFRs, and attaches them to affected functional requirements. This diagram could be considered as a complement to the use case, and provide an integrated view for NFRs.

III. The Proposed Appraoch

The proposed approach in this paper for analyzing and refining NFR is risk-driven which we got inspiration from risk assessment and management used in security engineering and the SPE process [9][11]. NFRs are identified based on possible business risks for potential low quality of service of the target system. External threats and internal vulnerabilities are then analyzed to find the possible reasons for low quality of service. Furthermore, tactics for achieving NFRs are recommended based on the risks, threats and vulnerabilities. In the following, we first introduce the aspects that needed to be considered when analyzing and refining NFRs. Then, based on these aspects, the risk driven analysis process is elaborated. Lastly, we present the constrained case diagram which used to model the analysis results.

A. Understanding Non-Functional Requirements

For best achievement of an NFR, a concise understanding of NFR itself and its environment is very important. Basically, we need to find out answers for the following questions in the analysis process [14]: (1) what exactly the NFR is about; (2) why need the NFR; and (3) how to achieve it. The first question could roughly correspond to a basic description of the NFR, including its category, related business goal (functionality) and the target value. The answer of the second question is risk related information since NFRs are proposed to address business risks. To achieve NFRs, we should identify the factors that may have adverse impact on them, and propose solutions according to the factors. Table I shows the main aspects that should be considered in NFR analysis. It could also be considered as a template to specify NFRs.

TABLE I. ASPECTS FOR UNDERSTANDING NFR

	Aspect	Description
what	Name	Name of the NFR
	NFR Category	Category of the non-functional requirement. E.g., performance, security, availability, reliability.
	Related business goals	Which part of the system does this NFR target apply to? An NFR can apply to a single function, a group of functions, a single interface, and so on, or it can apply to everything (all functions).
	Target (NFP value)	The quality target that the system need to deliver. It might be some quantitative or qualitative measure. E.g., System should operate 24 *7
	Exceptional cases	Is the quality target likely to be unachievable in some cases? E.g., functions that involve intensive processing will be slower, so it' s unfair to judge them against the same response time goal.
why	Motivation (risks to be addressed)	Specify the potential risk and analyze its impact on system's business value. It is the reason for proposing the NFR. . E.g., the motivation for a fast download of software by a Web visitor might be so that they don't give up impatiently halfway through.
	Criticality /Priority	Description of impact if the NFR is not achieved, and specify the impact grade,such as:high, medium or low.
How	Challenges, Threats or Vulnerabilities	Any circumstance or event that can have negative impacts on end-user experience of the quality, or a list of all those pieces relevant to your environment that have a bearing on the non-functional goal of your system.
	Impact rating/ Likelihood / Risk rating	Level of the impact resulting from successful exploitation of a vulnerability; likelihood of the probability that a potential vulnerability might be exploited in the context of the associated threat environment/ the level of risks to the system.
	NFR Tactics (risk mitigation recommendations)	Things to be done to help achieve this non-functional goal. They could be any specific features we want our system to have to help to achieve the goal. They are considered as a set of new derived requirements. It will be better if each of these requirements includes a statement estimating the extent to which it contributes.
	Extra requirements	Explains what sorts of extra requirements should be considered and in what circumstances. It provides guidance on what to do beyond simply specifying the main requirement.E.g.: E.g., Functions for measuring response times and letting administrators see them.
	Acceptance criteria	Could be taken as the criteria for verifying that the NFR has been met (may be used to assist definition of system test cases).

Among above aspects, the knowledge about threat, vulnerabilities, NFR tactics and relationships among them is especially important for finally implementing NFRs in the target system. The threat and vulner abilities explains the possible reasons why the system may provide poor quality of service, and the tactics provide countermeasures for the threats and vulnerabilities, so as to achieve NFRs. Fortunately, these types of knowledge could be accumulated based on previous experience, such as the development experience of similar systems, journal articles and conference papers, technical reports etc.

B. Risk-driven NFR Analysis Process

Formulating a detailed and accurate analysis of NFRs is not a straightforward task, because various aspects have to be considered. In this subsection, we present a process that guides the analyst to analyze and refine NFRs. It starts with characterizing the system and identifying the initial NFRs by recognizing the associated risk to the system stakeholders. Then, threats and vulnerabilities that may do harm to the NFR, and possible tactics to mitigate the risks are identified so as to achieve NFRs in the target system. This stage is accomplished through following steps, which need the participation of both the stakeholders and developers.

1) Preparation:characterizing the target software system

In this step, the analyst defined the boundaries of the IT system, along with the resources that constitute the system, its connectivity, and any other elements necessary to describe the system. The aim is to provide a context for NFR analysis.

2) Risk analysis and NFR identification

Analyze potential business risks, assess their impacts; and neatly specify the concerned non-functional properties for mitigating the risks, including the category, a brief summary on the target value and the reasons why stakeholders concern about them. This step could be assisted by an existing risk assessment process.

3) Related functionality identification

Based on system characterization, identify which functionality is related to the non-functional requirements or what does this NFR apply to. Is it for a specific function, a collection of functions, all functions for a class of users, or something else?

4) Threats identification

Towards the identified NFR and system characterization, draw up a list of the external threats or risks relevant to the environment that have a bearing on the non-functional properties of the system. Record and specify their impacts to corresponding non-functional properties. This knowledge could be collected from trusted and proven journal articles, conference papers, books, software companies' brochures and some websites [9][15].

5) Vulnerability idenification

In this step, the analysts developed a list of internal system vulnerabilities (flaws or weaknesses) that could be exploited by the potential threat vectors or that would have negative impacts on the achievement of NFRs. It often needs a deeper analysis of system characteristics.

6) Risk determination

In this step, the analysts determined the degree of risk to the system, including the likelihood determination, impact rating and the risk level. Existing assessment methods [9][10] could be directly used in this step.

7) Exceptional cases identification

Is the non-functional requirement likely to be unachievable in some cases? For example, functions that involve intensive processing will be slower, so it's unfair to judge them against the same goal.

8) Tactic recommendations

Based on the identified threats and risks, identify the tactics that could be used to improve any of these threats or risks. It would be better if each tactic include a statement estimating the extent to which it contributes. Since tactics for NFRs are relatively fixed, knowledge about the tactics could be collected in advance from previous experiences or dependable documents [6][15].

9) Extra requirement identification

Besides the tactics recommended mitigating the threats and vulnerabilities, some additional requirements may be introduced. For example, functions for measuring response times and letting administrators see them. In this step, we identify and explain what sorts of extra requirements should be considered and in what circumstances.

10) Result documentation

Presented to the stakeholders to check and confirm the derived requirements and to the developers for the following design and implementation. We could use the template derived from table II and constrained case diagrams defined in the following subsection.

Noticing that this is an iterative and incremental process, and not all steps must be performed in a NFR analysis process. Analyzers and users could customize above steps and form more appropriate process.

C. Constrained Case Diagram for NFR Modeling

To provide a precise context to NFR design and implementation, we propose a constrained case diagram to graphical model the NFRs, the derived requirements, the original functional requirements and relationships among them. This diagram could be roughly considered as a complement to the use case diagram since we refer to the use cases to specify the functionality affected by the NFRs.

Figure.1 shows the metamodel of the constrained case diagram. The gray elements are borrowed from UML specification [14], and the white elements are which we proposed to model NFR related elements. This diagram mainly consists of four types of classifiers and four types of relationships. The classifiers include: (1) the *Constrained Case* which is used to represent the refined non-functional requirement; (2) the constrained artifact that constrained by the NFRs which refers to specific *Use Case*; (3) the *NFR Tactic* which recommended to deal with the threats and vulnerabilities so as to achieve NFR; (4) the *Extra Requirement* which identified to support the NFR. The relationships include: (1) the relationship *Constrain* which connect NFR and related functionality together; (2) the relationship *Satisfy* which connect the *NFR tactic* to the NFRs ; (3) the relationship *Support* which connect the *Extra Requirement* to the NFR; (4) the relationship *ActUpon* which specifies that a derived requirement will act upon the constrained artifact.

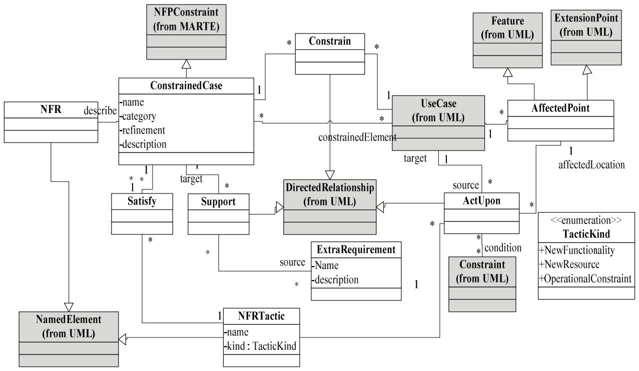


Figure 1. Metamodel of the Constrained Case diagram

For *ConstrainedCase*, the attribute *refinement* denotes the concrete NFRs refined from the high-level NFR. The attribute *constrainedElement* refers to the artifact that constrained by the NFR. In our metamodel, it mainly refers to use cases. For example, associating fast response time NFR to Withdraw Fund use case of an Automated Teller Machine (ATM) system. Moreover, to specify relationships between the NFRs and the constrained functionality, we introduce a relationship *Constraint* in this metamodel.

The element *NFRTactic* in this metamodel represents the countermeasure adopted to make the NFR satisfied. It might be a new functionality, new resource, operational constraint, etc. It will be realized into some software entities and interact with the constrained functionality. The *Extra Requirement* represents the requirement derived to support certain NFR but not to achieve them, such as monitoring related non-functional property and letting administrators see them. The relationship *ActUpon* is proposed to specify the relationship between a NFR derived requirement and the affected use case. Moreover, since sometimes use case is further described by interactions, activities or state machines, we could specify the specific location that the NFR tactics will act upon. Using the element *affectedPoint*. For example, for a use case described by an activity diagram, the affected point may be one of the actions.

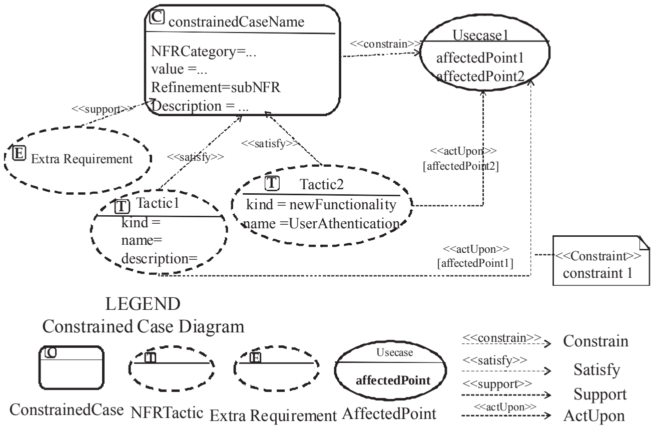


Figure 2. Graphical notation of Constrained Case diagram

Figure 2 shows the graphical notation of the constrained case diagram. Using this diagram, the NFRs, the derived requirements (NFR tactics and extra requirements), the affected use cases, and relationships among them could be modeled explicitly in an integrated view, which provides the designers a precise context for NFR realizations and a source of the transformation from NFRs into architecture.

The constrained case diagram could be considered as a complement to the functionality-focused use case diagram, highlighting in each view that designer needs to consider both the functional and non-functional requirements to present a complete picture for the following software development stages. For example, it could be leveraged in the 4+1 view [15], together with the use case view to assist the software architecture design; or it could be leveraged by the tester to validate the requirements. Additionally, it can also be used in planning project activities, resources, cost and timeframe.

IV. CASE STUDY

In this section, we present a case study on a simplified Online Shopping System.

A. *System Characterization and Identfiied NFRs*

Basic functionalities of an online shopping system include account management which assists customer to maintain their account information and; allowing customers to browse the product; adding, removing items to shopping carts and to reviewing their orders; especially, to attract customer, the system is required to provide a flash sale/spike area function, where in a certain period of time, a product is sold in an incredible price. This may result in a big visit amount on the website. Figure 3 shows the functionality-focused use case diagram of the system.

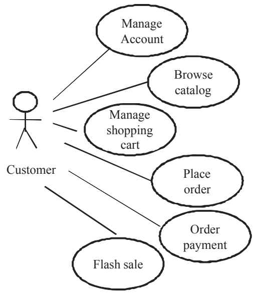


Figure 3. The usecase diagram

By analyzing potential risks from users' perspective, three types of important NFRs are identified, one is the *Security* for account, the other is the *Availability* of the shopping, and the last one is the *Performance* for flash sale. Table II(a),II(b),II(c) shows the initially identified NFRs using our template.

TABLE II. INITIALLY IDENTIFIED NFRS DESCRIPTION

Constrained Case: Security constraint
Description: the system shall have high security.
Motivation: Account information is very important for online shopping system, poor security may injure customers' benefits and affect the customer retention.
NFR Category: Security

(a) Initially identified Security constraint

Constrained Case: Availability constraint
Description: the system shall be available to users 24 hours every day, every day of the year, especially the functionality Place Order.
Motivation: most online shopping systems are available 7*24, if the target system does not provide similar availability, it might lead the risk of losing many customers for the business.
NFR Category: Availability

(b) Initially identified Availability constraint

Constrained Case: Performance constraint
Description: the various functionalities shall have an appropriate and acceptable response time.
Motivation: Under multiple concurrent user load, the system's response time may become unacceptable to the end users which lead to the risk of losing customers.
NFR Category: Performance (Response time)

(c) Initially identified Performance constraint

B. *Risk Driven Analysis and Refinement*

Towards these high-level NFRs, the next to do is refining them, analyzing threats and vulnerabilities, and deriving the tactics and extra requirements. Table III shows the analysis results guided by our risk driven analysis approach.

TABLE III. REFINED NFRS DESCIPTION

Constrained Case: Security constraint		
Related Business Goal: Manage Account, Order payment		
Challenges or threats: unauthorized access		
Impact Rating: High	Likelihood: Medium	Risk Rating: Medium
Recommended tactics: Data Encryption, Password Authentication, select the third-party payment platform		

(a) Refined Security constraint

Constrained Case: Availability constraint		
Related Business Goal: Place Order, Order payment		
Challenges or vulnerabilities: Unreliable network links, hardware failure, service interruptions of the pay platform		
Impact Rating: High	Likelihood: Medium	Risk Rating: Medium
Recommended tactics: Ping/echo, Replicate Server, Multiple payment mechanism		
Extra Requirement: System unavailable page(when the system is unavailable to users, any attempt by a user to access the system shall result in the display of page informing them that it is unavailable),		

(b) Refined Availability constraint

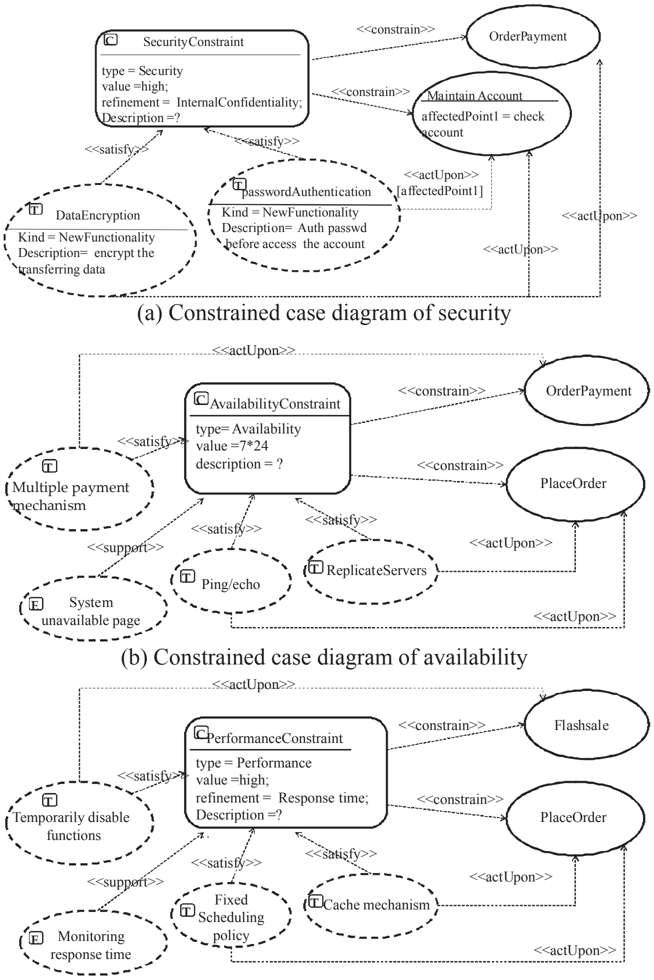
Constrained Case: Performance constraint		
Related Business Goal: Place Order, Flash sale		
Challenges or vulnerabilities: Large amount of concurrent users,		
Impact Rating: High	Likelihood: Medium	Risk Rating: Medium
Exceptional case: at the time of flash sale, system load are inevitable high, allow more response time for flash sale		
Recommended tactics: Fixed Scheduling Policy, Cache mechanism, Temporarily disable functions that cause intensive processing		
Extra Requirement: Response time monitoring (Functions for measuring response times and letting administrators see them)		

(c) Refined Performance constraint

C. *Model NFRs in Constrained Case Diagrams*

To provide a clear graphical input to the following software development stages, we model the NFRs and their analysis results using graphical constrained case diagrams as shown in Figure 4.

From this example, we could see that our risk driven approach could provide an operational guide to effectively analyze various aspects of NFRs, and the constrained case diagram could give developers an integrated view for NFRs, derived requirements and related functionalities.



(a) Constrained case diagram of security

(b) Constrained case diagram of availability

(c) Constrained case diagram of performance

Figure 4. Constrained Case diagrams for NFRs

V. Discussions and Conclusions

Non-functional requirement is important for building user-satisfied software. However, for its non-behavioral nature, it is relatively difficult to directly implement them in the target software systems. A complete analysis of its context, source, related threat and vulnerabilities is very useful and helpful for identifying the tactics to achieve them. In this paper, we propose a risk driven approach to analyzing and refining NFRs. A constrained case diagram is also proposed to provide an integrated graphical view for modeling the analysis results.

The effectiveness of this approach is dependent on the strength and validity of the NFR related knowledge, including the risks, threats, vulnerabilities and corresponding NFR tactics. A deep understanding of the system characteristics and its environment are also important for analyzing NFRs. Another factor is the experience of the software engineer. Although there some NFR related knowledge for reference, it also needs software engineer to understand the target system and make decisions during analysis. In addition, for graphical modeling NFRs, the constrained case views are more appropriate to represent the NFRs that associated with specific functionality that denoted by functional use case. For system-wide NFRs, such as maintainability, and portability that are not normally associated with a particular functionality, we could use the template to describe them, and use the constrained case view to document the policy or tactics that help the project achieve the softgoal. In the future work, we will entail validation of the efficacy of the risk driven NFR analysis approach and collect more specific NFR related knowledge to support the process.

Acknowledgment

The work supported by Beijing Natural Science Foundation (4122036); the National Basic Research Program of China under Grant No.2009CB320703; the Scientific Research Foundation for the Returned Overseas Chinese Scholars, State Education Ministry; the National Natural Science Foundation of China under Grant No.61121063, No.61003065; Specialized Research Fund for the Doctoral Program of Higher Education (No.20101101120027); Excellent Young Scholars Research Fund of Beijing Institute of Technology (No.2010Y0711).

References

[1] L Chung, J do Prado Leite, “On Non-Functional Requirements in Software Engineering”, Conceptual Modeling: Foundations and Applications, 2009, pp.363-379.

[2] A.Matoussi, R.Laleau, “A Survey of Non-Functional Requirements in Software Development Process”, Technical report,TR-LACL-2008-7.

[3] IEEE Std 830-1993. IEEE Recommended Practice for Software Requirements Specifications.

[4] L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos, “Non-Functional Requirements in Software Engineering”.1st ed. Springer 1999.

[5] Y.Eric,“Towards modeling and reasoning support for early-pahse requirements engineering”, Proc.Third IEEE Inl.Symposium on Reqirement Engineering, 1997.

[6] I.Alexander, “Misuse cases help to elicit non-functional requirements”, Computing & Control Engineering Journal, 2009, pp. 40-45.

[7] Joe Zou, Christopher J.Pavlovski, “Control case approach to record and model non-functional requirements”, Information Systems and E-Business Management, 2008, Volume 6, Number 1, p49-67, 2008.

[8] L.Chung, S.Supakkul, “Representing NFRs and FRs: A Goal-Oriented and Use Case Driven Approach”, SERA 2004, LNCS 3647, pp.29-41.

[9] J.Allen,S.Barnum,R.Ellison,G.McGraw,N.Mead, “Software security engineering: a guide for project managers”, Addison-Wesley,2008.

[10] Stoneburner G, Goguen A, Feringa A, “Risk management guide for informaion technology systems”, Natiaonal Institute of Standards and Technology (NIST), U.S.Department of Commerce, Publication 800-300.

[11] C.U.Smith, L.G.Williams, “Performance Solutions: A practical Guide to Creating Responsive, Scalable Software”, 1st E, Addison-Wesley, 2011.

[12] M. Glinz, “On Non-functional Requirements”, Proc.5th IEEE International Conference on Requirements Engineering, 2007, pp.21-26 .

[13] F. Armour and G. Miller, “Advanced Use Case Modelling”, Addison-Wesley, 2001.

[14] C.Harris,M.Davis,M.Pritchard,M.Rabins,“Engineering Ethics: What? Why?How? And When?”, Joual of Engineering Education, April 1996.

[15] M.Barbacci,M.H.klein,T.A.Longstaff,C.B.Weinstock,"Quality Attruibutes",Technical Report,CMU/SEI-95-TR-021,1995

[16] Malik Hneif, Sai Peck Lee, “Using Guidelines to Improve Quality in Software Nonfunctional Attributes”, IEEE Software 28(6), p72-77,2011.

[17] Object Management Group, “Unified modeling language: Superstructure version2.0”, OMG Document,formal/05-07-04,2005.

[18] P.Kruchten, “Architectural Blueprints – The ‘4+1’ View model of Software Architecture”, IEEE Software, 12(6), 1995, pp42-50.

Eliciting Security Requirements in the Commanded Behavior Frame: An Ontology based Approach

Xiaohong Chen*†,Jing Liu*

**Shanghai Key Laboratory of Trustworthy Computing, East China Normal University, CHINA*

†*Key Laboratory of High Confidence Software Technologies (Peking University), Ministry of Education, CHINA*

***Abstract*—The Problem Frames(PF) approach is well known for analysing and structuring requirement problems in requirements engineering. However, currently it lacks of a practical and effective way to obtain security requirements. To solve this problem, this paper proposes to elicit security requirements by constructing an act-effect model which is used to model the environment effects that react to the users' commands. The generation of the act-effect model can be guided by a software environment ontology which models the software environment. Finally, a case study is given for illustrating the security requirements elicitation.**

***Keywords*-requirements engineering; security requirements; software environment ontology; Problem Frames approach;**

I. INTRODUCTION

With the increasing demands for secure softwares, how to get security requirements becomes an important issue. Generally speaking, security requirements are included in a system to ensure [1]:

- unauthorized access to the system and its data is not allowed (the authority problem);
- the integrity of the system from accidental or malicious attack (the integrity problem).

Much work has been done in security requirements aiming at these two problems such as [2][3][4][5][6]. This paper focuses on the integrity problem. There is a famous approach for analysing security requirements called abuse frames[1] which is developed on the basis of the Problem Frames(PF) approach[7]. Security requirements are viewed in terms of "Assets" and "Attackers" where attackers behave like operators to cause state changes in the assets. The security behavior is quantified as an "anti-requirement" which is a behavior of an asset under some inputs from an attacker.

However, at present the abuse frames approach still lacks of a practical and effective way to obtain such security requirements. In this paper, we propose to use the software environment ontology for helping elicit security requirements from the basic problem frames. There are five basic problem frames in the PF approach. They are the required behavior frame, the commanded behavior frame, the information display frame, the simple workpieces frame and the transformation frame, and some variants to them. Aiming at the commanded behavior frame, this paper first presents the act-effect model which tries to describe the environment effects that react to the user commands according to environment properties. The environment properties are provided by a software environment ontology built for modeling the environment of software [8]. Then the undesirable behaviors of "Assets", i.e., security requirements, can be obtained.

The rest of this paper is organized as follows. Section II introduces the commanded behavior frame in the PF approach and the software environment ontology. Section III presents security requirements in the commanded behavior frame. Section IV gives a small example-secure sluice gate control problem to show how to capture security requirements in the commanded behavior problem. At last, section VI concludes this paper and indicates the future work.

II. BACKGROUND

A. The Commanded Behavior Frame

In the PF approach, the commanded behavior frame characterizes problems in which the machine is required to accept the operator's commands and impose the control accordingly. The problem frame diagram is shown in Fig. 1.

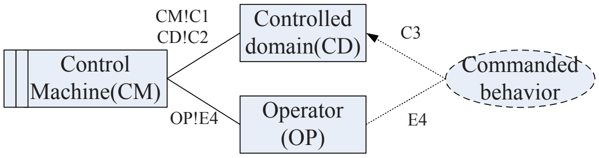


Figure 1. The commanded behavior: problem frame diagram [7]

In the figure, the software problem is to specify a control machine (the solution) to control a controlled domain (the problem context) in accordance with commands issued by the operator so that the commanded behavior (the requirement) is satisfied. In this frame the general form of a software development problem has been elaborated only by more specialised names for the principal parts and by markings on the connecting lines which indicate:

- The requirement 'commanded behavior' is a condition over phenomena of C3 and E4.
- The control machine CM is the machine to be built.
- The controlled domain CD is its problem domain. CD is a kind of environment entity, or an asset to be protected.

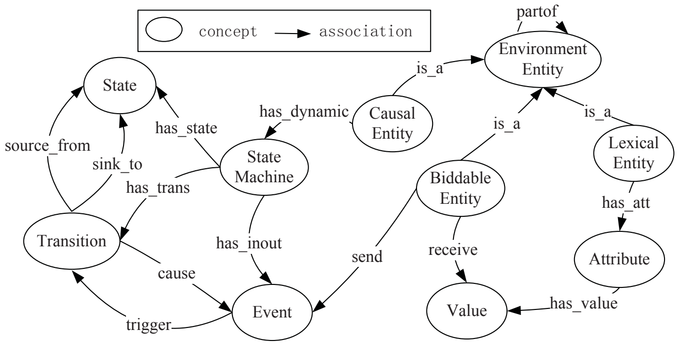


Figure 2. Conceptual model of software environment ontology

- The interface between the control machine CM and controlled domain CD consists of two kinds of shared controllable phenomena: C1, controlled by CM, and C2, controlled by CD. The interface between CM and the Operator OP consists of event E4 issued by OP.
- The shared phenomena C1, C2, C3 and E4 are interactions between CM and CD, and CM and OP.

Accordingly, a requirement problem can be defined.

Definition 1: A requirement problem is a five-tuple:

$$P := < M, ES, PS, IS, RS >$$

Where

- M is the machine to be built
- $ES = \{e_1, e_2, \dots, e_n\}$ is the entity set
- $PS = \{ps_1, ps_2, \dots, ps_n\}$ is the phenomena set
- $IS : M \times ES \times PS \cup ES \times M \times PS$ is the interaction set
- $RS = \{r_1, r_2, \dots, r_n\}$ is the requirement set

B. Software Environment Ontology

We have developed software environment ontology in [8]. It is built for sharing knowledge about software environment. Fig. 2 gives its conceptual model which includes the following basic concepts:

- The environment of a software system is assumed to be a set of entities that will interact with the to-be built software system. So the basic concept is *environment entity*. Environment entities can be classified into three categories: the *biddable entity*, the *lexical entity*, and the *causal entity*.
- A biddable entity usually consists of people, it is physical but lacks positive predictable internal causality. Each biddable entity usually sends events or provides (or receives) *values*. Each biddable entity has a set of *attributes*. Each attribute has a value.
- A lexical entity is the physical representation of data.
- A causal Entity is one whose properties include predictable causal relationship among its shared phenomena. Each causal entity has a set of dynamic properties which can be described by *state machines*.
- Each state machine has a set of *states* and *transitions*. Each transition sources from a state, and sinks to a state. A transition can be triggered by an *event*.

Among these concepts, we'll give a formal definition of the state machine of a casual entity for further use.

Definition 2: State machine SM of a casual entity ce

A SM can be defined as a triple:

$$SM := < S, E, T >$$

where

- S=$\{s_0, s_1, \dots, s_n\}$ is the state set of ce
- E=$\{e_0, e_1, \dots, e_m\}$ is the event set of ce
- $T : S \times E \rightarrow S$ is the transition set of ce

The above software environment ontology is an upper ontology. It employs a core glossary that contains basic terms and associated descriptions of the software environment as they are used in various, relevant domain sets. The instantiation of the upper ontology in a specific domain will result in the domain software environment ontology. It contains the domain specific terms for describing the software environment. There could be many domain environment ontologies upon to the upper software environment ontology.

III. Security requirements in the commanded behavior frame

Some part of the physical world's behavior is to be controlled in accordance with commands issued by an operator in the commanded behavior frame. In other words, these control commands will cause effects to the physical world which could be state changes of the physical world. As functional requirements can be viewed as desired effects, the security requirements can be seen as undesired effects. To model relations between commands and effects, we introduce act-effect model.

Definition 3: Act-effect model of (P,be, ce)

Suppose a problem $P = < M, ES, PS, IS, RS >$, a biddable entity $be \in ES$, a casual entity $ce \in ES$. An act-effect model ae of (P, be, ce) is defined as a triple:

$$ae := < S, E, T >$$

where

- $S = \{s_0, s_1, \dots, s_n\} \subseteq PS$ is the state set of ce
- $E = \{e_0, e_1, \dots, e_m\} \subseteq PS$ is the event set of be
- $T : S \times E \rightarrow S$ is the transition set

An act-effect model dynamically describes how the commands from a biddable entity finally change the states of a casual entity. Then our concern turn to how to get an act-effect model from an existing problem description (i.e., a problem diagram) and a domain software environment ontology. The problem description gives desired effects by commands of the biddable entities. The domain software environment ontology provides the effects of casual entities and the essential reasons for effects by state machines.

Combining these together, we can get an algorithm for generating an act-effect model (see Alg. 1). There are three main steps in Alg. 1: 1) getting the states of the casual entity; 2) obtaining the events sent by the biddable entity; and 3) constructing the transitions.

Algorithm 1: Act-effect model generation algorithm

```
Input: problem description $P =< M, ES, PS, IS, RS >$, a biddable entity $be \in ES$, and a casual entity $ce \in ES$, domain software environment ontology $DO$
OutPut: act-effect model of $(P, be, ce)$ $A_2 =< S_2, E_2, T_2 >$
begin
1. Search the casual entity $ce$ in $DO$;
2. Find the state machine $A_1 =< S_1, E_1, T_1 >$ of $ce$ through 'has_dynamic' in $DO$;
3. Set $S_2 = S_1 \cap PS$;
4. Search the biddable entity $be$ in $DEO$;
5. Find all the events that be initiated, recording them as $E$;
6. Set $E_2 = E \cap PS$;
7. For $t_1 \in T_1$, $t_1 =< s_1, e_1, s'_1 >$, $\forall e_2 \in E_2$
8.          If after $e_2$ happens, the machine will send $e_1$,
9.                    Construct $t_2 \in T_2$, $t_2 =< s_1, e_2, s'_1 >$
end
```

However, the generated act-effect model only describes the desired commands that the biddable entity initiated to impose effects on the causal entity. Nothing is said about the undesired commands and its effects. Therefore we define complete act-effect model for modeling both desired and undesired effects.

Definition 4: Complete act-effect model

For an act-effect model $A =< S, E, T >$ of a problem P, $\forall e \in E$, $s_1 \in S$, if $\forall t \in T$, $\exists s_2 \in S$ such that $t =< s_1, e, s_2 >$, then A is a complete act-effect model of P. The complete act-effect model demands that each state of a causal entity will react to each command that the biddable entity initiates. It includes effects both desired and undesired. The desired effects can be obtained from Alg. 1. The undesired effects need to be determined by the requirement providers. Generally speaking, we often recommend ignoring the commands. According to the above analysis, the complete act-effect model can be generated by following Process 1. There are three main steps in Process 1: 1) setting all the states of the casual entity from a general act-effect model; 2) getting all the events of the biddable entity from domain environment ontology; and 3) constructing all the transitions both desired and undesired.

Process 1: Complete act-effect model generation process

```
Input: Problem description $P =< M, ES, PS, IS, RS >$, biddable entity $be \in ES$, casual entity $ce \in ES$, domain environment ontology $DEO$, act-effect model of $(P, be, ce)$ $A_2 =< S_2, E_2, T_2 >$
OutPut: complete act-model of $(P, be, ce) A_3 =< S_3, T_3, E_3 >$
begin
1. Set $S_3 = S_2$;
2. Search the biddable entity $be$ in $DEO$;
3. Find all the events that $be$ initiated, recording them as $E$;
4. Set $E_3 = E$;
5. For $t_1 \in T_2$
6.          $t_1 \in T_3$
7. For $e \in E_2$ $\forall s \in S$
8.          If $\exists s' \in S$ such that $< s, e, s' > \in T_2$;
9.          else construct $t_3 =< s, e, s > \in T_3$;
10. For $e \in E/E2$ $\forall s \in S$
11.                   Construct $t_3 =< s, e, s > \in T_3$;
12. return $A_3$;
end
```

After getting the complete act-effect model A_3, the security requirements as undesired effects could be generated by deleting the desired effects in A_2. The detailed process is shown in Process 2.

Process 2: Security requirements generation process

```
Input: act-effect model $A_2 =< S_2, E_2, T_2 >$, and complete act-model of $(P, be, ce) A_3 =< S_3, T_3, E_3 >$
OutPut: security requirements
begin
1. For $t \in T_3/T_2$, $t =< s_1, e_1, s_1 >$
2. Output
3. "when $ce$ is in $s_1$, if $be$ initiates $e_1$, then $M$ ignores it"
end
```

IV. Case Study

We will illustrate our method by developing a secure sluice gate control problem (adapted from Jackson's book [7]). The problem statement is as follows. *A small sluice, with a rising and falling gate, is often used in the irrigation system. A computer system is needed to raise and lower the sluice gate in response to the commands of an authorized operator. The gate is opened and closed by rotating vertical screws. The screws are driven by a small motor, which can be controlled by clockw, anti, on and off pulses. There are sensors at the top and bottom of the gate travel; at the top it's fully open, at the bottom it's fully shut. Sometimes the state of the sluice gate could be monitored by a monitor.*

Before we deal with this problem, a domain software environment ontology named sluice gate environment ontology must be built.

A. Sluice Gate Environment Ontology

According to the software environment, at least four steps are needed for constructing a domain software environment ontology:

1) extract environment entities and classify them

All the entities related to the sluice gate are possible environment entities. The environment entities involved include:

- *Operator (Biddable)*: someone who is in charge of controlling the sluice gate.
- *Monitor(Lexical)* : a device for displaying information.
- *Sluice Gate(Causal)*: can be rising and falling.

- *Sensor(Lexical)*: at the top and bottom of the gate, can be used to indicate the state of the gate.
- *Screws (Causal)*: can be rotated to open and close the door.
- *Motor(Causal)*: can be controlled by clockwise, anti-clockwise, on and off pulses, used to drive the screws.

2) find the attributes of the lexical entities and the events sent by the biddable entities

The attributes of the lexical entities are as follows:

- *Sensor* has an attribute *location*, which is used to indicate the location of the sluice gate. It has two values, *top* and *bottom*.
- *Monitor* has an attribute *state*, used to describe the state of the sluice gate. It has two values: *Open* and *Shut*.

Events that Operator sends are:

- *OpenG*: open the door
- *ShutG*: shut the door

3) construct the state machines for the causal entities

For each causal entity, analyze its state machine. For instance, *sluice gate* and *motor* are causal entities. They can be modeled as an entity-*sluice gate and motor(SM)*. Their states are changed from *open* to *shut*, *shut* to *open*, respectively after receiving *Anti* and *Clockw* pulses. The state machine is shown in Fig. 3.

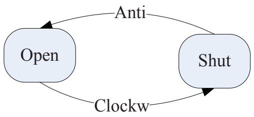


Figure 3. State machine of SM entity

B. Secure Sluice Gate Control Problem

The secure sluice gate control system must conform to the following rough problem statement:

The problem is to build a computer system to obey the authorized operator's command to control the sluice gate.

Step 1: fit the sluice gate control problem diagram to the commanded behavior frame

According to [7], this problem can be fitted to the commanded behavior frame whose problem frame is shown in Fig. 4. Where, the *sluice controller* is the control machine, the *gate & motor* is the controlled domain, the *sluice operator* is the operator, and the *OpenG* and *ShutG* commands issued by the *sluice operator* are the phenomena E4. The required behavior is called *open & shut gate*. The general idea is that the *sluice operator* can position the gate as desired by issuing *OpenG* and *ShutG* commands: the machine should respond to *ShutG* by putting the *gate & motor* into a *shut* state, and so on.

Step 2: identify attacks

1) build an act-effect model

In the sluice gate environment ontology, there are two states of *SM*, *Open* and *Shut*, and two states events issued by *OP* including *OpenG* and *ShutG*. The command *OpenG* corresponds to *Clockw*, and *ShutG* to *Anti*. According to Alg. 1, replace *ClockW* with *OpenG*, and *Anti* with *ShutG* in Fig. 3, then an act-effect model of *SM* can be obtained as shown in Fig. 5.

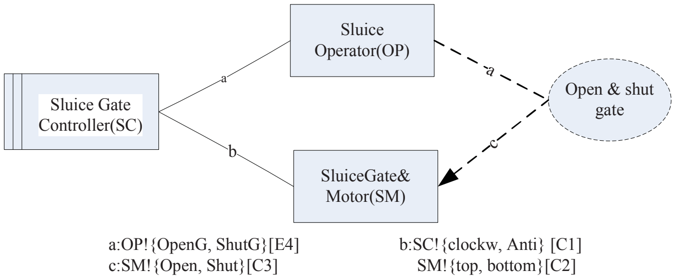


Figure 4. Problem diagram of the sluice gate control problem

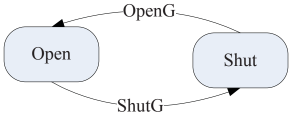


Figure 5. Act-effect model of (SC,OP,SM)

2) get a complete act-effect model

According to Process 1, besides the act-effect model in Figure 5, other state transitions need to be considered. For example, if *SM* is *Open*, the command *OpenG* should never work. Likewise, if *SM* is Shut, the command *ShutG* should be ignored. Thus we get a complete act-effect model shown in Fig. 6.

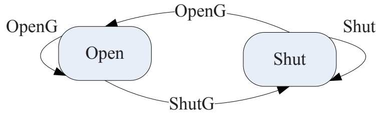


Figure 6. A complete act-effect model of (SC,OP,SM)

3) generate the security requirements

Security requirements of the sluice gate control problem can be obtained from complete act-effect model in Fig. 6. Using Process 2, the security requirements in Fig. 6 can be expressed as:

- When *SM* is in the state of *Open*, if *OP* initiate a *OpenG* command, *SC* ignores it.
- When *SM* is in the state of *Shut*, if *OP* initiate a *ShutG* command, *SC* ignores it.

V. RELATED WORK

A number of efforts have been done in security requirements and engineering. For example, misuse cases [9][2] and abuse cases [3] have been suggested as tools for threat analysis. However, these typically represent instances of attacks that are often described in the language of implementation.

Chung [4] treats security requirements as softgoals, that are identified and refined based on a knowledge catalogue of decomposition methods, security techniques for satisfying softgoals, and correlation rules. This approach focuses on the elicitation process of high level security goals. The i* framework [5] takes an organizational view by modeling trustworthiness as softgoals to be satisfied. Attacks by malicious users are modeled as negative contributions that obstruct these softgoals. Above all, i* focuses on analysis of security threats imposed by internal actors at the organizational level. Attack trees [10] adopt a goal-oriented approach to refining a root goal into a goal tree to derive scenarios. But this approach is best suited to design.

Compared with the above work, our approach is at the analysis phase which treats the solution domain as a black box and focus on the domain. It complements the softgoal view by analysing external threats in software systems problem domains.

The most relevant works are two supplementary approaches based on the PF. The first one is defined by Lin et al. [1]. They define so-called “anti-requirements” and the corresponding “abuse frames” to use the ideas underlying problem frames in security. An anti-requirement expresses the intentions of a malicious user, and an abuse frame represents a security threat. For a threat to be realised, its abuse frame must be composed with the base problem frame in the sense that the asset attacked in the abuse frame must overlap, or be identified with, a domain of the base problem frame. The purpose of anti-requirements and abuse frames is to analyze security threats and derive security requirements.

Hatebur et al. [11][6] take a different approach using problem frames. Security requirements are expressed using a threat model. Security problem frames are used to consider security requirements. The goal is to construct a machine that fulfills the security requirements. Security problem frames strictly refer to the problems concerning security.

In our work, the undesirable under-attack behaviors are actually captured as “anti-requirements”. Following Lin et al. ’s work, we provide a practicable way to obtain these anti-requirements.

VI. Conclusion

Based on the commanded behavior frame and the abuse frames, this paper proposes to elicit security requirements by constructing an act-effect model. This model is generated by referring to the properties of the environment in software environment ontology. The security requirements can be easily obtained from these act-effect models. The case study shows that our approach is feasible.

An important benefit of our approach is the systematic and repeatable of the security requirements elicitation. The software environment ontology is used to guide the elicitation process, which greatly reduces the analysts’workload. Our approach is not a substitute for abuse frames or the other traditional security engineering techniques. We have found them to be useful in complementing such techniques when deployed during requirements analysis. We are currently examining ways of eliciting security requirements for the other basic problem frames.

Acknowledgments

This work was supported by the National Basic Research and Development 973 Program of China (Grant No.2009CB320702), the National Natural Science Foundation of China (Grant No.61170084,No.90818026), and Creative Team of NSFC (Grant No.61021004), the Opening Fund of Top Key Discipline of Computer Software and Theory in Zhejiang Provincial Colleges at Zhejiang Normal University, as well as the National 863 High-tech Project of China (Grant No.2011AA010101).

References

[1] L.Lin, B.Nuseibeh, D.Ince, M.Jackson, and J.Moffett, “Introducing abuse frames for analyzing security requirements,” in *the 11th IEEE International Requirements Engineering Conference (RE’03)*, 2003, pp. 371–372.

[2] I. Alexander, “Misuse cases: use cases with hostile intent,” *IEEE Software*, vol. 20, no. 1, pp. 58–66, 2003.

[3] J. McDermott and C. Fox, “Using abuse case models for security requirements analysis,” in *Annual Computer Security Applications Conference*, 1999, pp. 6–10.

[4] L. Chung, B. Nixon, E. Yu, and J. Mylopoulos, “Non-functional requirements in software engineering,” in *Kluwer*, 2000.

[5] L. Liu, E. Yu, and J. Mylopoulos, “Security and privacy requirements analysis within a social setting,” in *International Conference on Requirements Engineering (RE03)*, 2003, pp. 8–12.

[6] H. Schmidt, D. Hatebur, and M. Heisel, *A Pattern-Based Method to Develop Secure Software*. IGI Global, 2011, ch. 3, pp. 32–74.

[7] M.Jackson, *Problem Frames: Analyzing and Structuring software development problems*. Harlow, England:Addison-Wesley, 2001.

[8] X.Chen, B. Yin, and Z. Jin, “An approach for capturing software requirements from interactive scenarios,” *Chinese Joumal of Computer*, vol. 34, no. 2, pp. 329–341, 2011, in Chinese.

[9] G. Sindre and A. Opdahl, “Eliciting security requirements by misuse cases,” in *37th International Conference on Technology of Object-Oriented Languages and Systems (TOOLS-PACIFIC 2000)*, 2000.

[10] B. Schneier, *Attack Trees*. Dr. Dobb’s Journal, 1999.

[11] D. Hatebur, M. Heisel, and H. Schmidt, “Using problem frames for security engineeing,” Tech. Rep., 2006.

An Overview of the RSLingo Approach

David de Almeida Ferreira, Alberto Rodrigues da Silva
INESC-ID, Instituto Superior Técnico (IST), Lisbon, Portugal
{*david.ferreira, alberto.silva*}*@inesc-id.pt*

***Abstract*—In order to carve out from the open space of possibilities the software system that the business stakeholders need and expect, it is crucial to properly document all observable and desired characteristics of the software system to be built, i.e., its requirements. In this paper we present RSLingo, an information extraction approach based on two domain-specific languages: RSL-PL and RSL-IL. The former allows the definition of linguistic patterns to enable the automatic analysis of requirements specifications written in natural language. While the latter supports the formal specification of requirements--related information that is automatically extracted. Since it enables further processing on the gathered knowledge, the RSLingo approach allows one to automatically verify some quality criteria and generate complementary representations that assist stakeholders during requirements validation.**

***Keywords*-Requirements Specification Language, Information Extraction, Requirements Modeling.**

I. Introduction

Although the body of knowledge provided by literature in Requirements Engineering (RE) is extensive [1], [2], RE activities still require a *significant human effort*. Most RE techniques are human-centric, thus having a multi-disciplinary background. Also, these techniques still lack proper tool support. Several requirements management tools exist [3], yet most RE techniques are manually applied, such as when producing documents, tables, and diagrams [4]. The human-intensive labour that these activities entail makes them time-consuming and error-prone. Thus, despite the RE field's best practices being well documented [1], [5], [6], the quality of these artifacts strongly depends on the experience and skills of the team members developing them.

It would be helpful if some of the manually performed tasks, related with requirements text analysis, could be automated for the following purposes:

- *Domain analysis:* for identifying all the relevant concepts, their relations, and how the software system manipulates them to provide the desired capabilities, while abiding to all stated constraints regarding the functionality that provides such capabilities to its users;
- *Verification:* for performing consistency checking on the extracted domain knowledge, through inference and ambiguity resolution based on glossaries and other lexical resources, to prevent delays and extra costs due to rework arising from belatedly discovered defects [2];
- *Transformations:* for automatically generating alternative requirements representations such as diagrams, tables, reports, or even template-based paraphrases of requirements statements in a controlled natural language, according to specific viewpoints. Also, an initial draft of the domain model and behavior models (e.g., UML class diagrams and UML use case diagrams, respectively) can be produced and provided as an input to software development processes that follow the Model-Driven Engineering paradigm [7].

Recognizing that natural language text is the preferred and recommended medium for documenting requirements [8], and that *ambiguity* is an intrinsic characteristic of natural language [9], we consider that there are two main approaches to deal with this necessary nuisance within the field of RE: (1) following a *controlled natural language approach* [10], in which language ambiguity is removed by forcing users to write according to an unambiguous predefined syntactical structure and a limited vocabulary; or (2) adopting a more flexible and extensible *information extraction approach* [11] based on linguistic patterns where, despite ambiguity not being completely eradicated, only the most plausible interpretation is considered for further processing, according to a best-fit match with the linguistic patterns recognized.

In this paper we overview RSLingo, an Information Extraction [11] approach for automatically analyze and formally specify requirements written in natural language. The novelty of RSLingo arises from its strategy of using two distinct domain-specific languages: RSL-PL and RSL-IL. Each of these languages addresses a specific aspect regarding the concerns that must be taken into consideration while formally documenting requirements. The former addresses the definition of common linguistic patterns of textual requirements representations. The latter was designed to formally convey RE-specific information, thus enabling further processing of this information. Through the mapping process between RSL-PL and RSL-IL, RSLingo aims to improve the quality and rigor of requirements specifications written in *ad hoc* natural language through complementary representations that can be used to validate the specified requirements.

The structure of this paper is as follows. Section II introduces the RSLingo approach, namely the main processes, roles, and the advantages of using a multi-language strategy. Section III discusses the strengths and limitations of the RSLingo approach, by comparing it with related work. Finally, Section IV concludes this paper and lays down some ideas for future work.

II. The RSLingo Approach

Natural language is the most common form of requirements representation [12], [8] used within requirements documents, requirements repositories (e.g., databases), and even diagrams[1]. Thus, we advocate that, in order to further benefit from the effort in developing requirements specifications in natural language [5], [6], suitable languages and supporting toolsets are mandatory to better and properly support requirements authoring and validation activities [1].

The first step towards the automation of requirements analysis is treating requirements' textual representations according to a "white-box" approach in order to extract and analyze their meaning. To this end, while not forcing stakeholders to adopt a new notation, we propose an information extraction approach based on simplified Natural Language Processing (NLP) techniques, such as *chunk parsing* [13]. The aim of chunk parsing is not to enforce grammatical correction, but to exploit the alignment between the structure of sentences and their semantics (i.e., the meaning of requirements). Besides requiring less computational power, chunk parsing is more flexible and robust while allowing one to focus on pieces of information (i.e., the chunks) that carry relevant requirements information. Therefore, despite being less complex than the traditional full parsing approach, these techniques still enable us to obtain the required domain knowledge for a deeper insight on the system to be built. These simplifications exploit the syntactic–semantic alignment imprinted in linguistic patterns [13].

Considering this premise, we named our approach *RSLingo*. The name stems from the paronomasia on "RSL" and "Lingo". On the one hand, "RSL" (Requirements Specification Language) emphasizes the purpose of formally specifying requirements. The language that serves this purpose is *RSL-IL*, in which "IL" stands for *Intermediate Language*. On the other hand, "Lingo" emphasizes that its design has roots in natural language, which are encoded in linguistic patterns used during information extraction from the textual requirements specification [11]. The language designed for encoding these RE-specific linguistic patterns is *RSL-PL*, in which "PL" stands for *Pattern Language*.

The RSLingo approach considers two distinct stages: definition at *process-level* and usage at *project-level*. Process-level, depicted in Figure 1, comprises the definition (or adaptation) of the linguistic patterns encoded in RSL-PL, and also establishing the mapping between these linguistic patterns and the semantically equivalent RSL-IL formal structures. On the other hand, as illustrated in Figure 2, project-level consists in applying its languages and using the RSLingo's toolset during a specific software project.

*1) **RSLingo's Process-Level:*** as illustrated in Figure 1, it considers an unusual role, which we named *Requirements Architect*. As the adopted name suggests, the person performing this role must have a thorough knowledge and vast experience in RE, to be able to tailor the RSLingo approach to the project at hand. Also, the Requirements Architect must be able to identify common linguistic patterns that frequently occur in natural language requirements specifications. The expertise of those playing this role is a crucial factor for the success of the RSLingo approach. However, after this tacit knowledge is encoded into the mapping between RSL-PL linguistic patterns and RSL-IL formal structures (the main asset to be produced), this knowledge can be reused in similar projects multiple times.

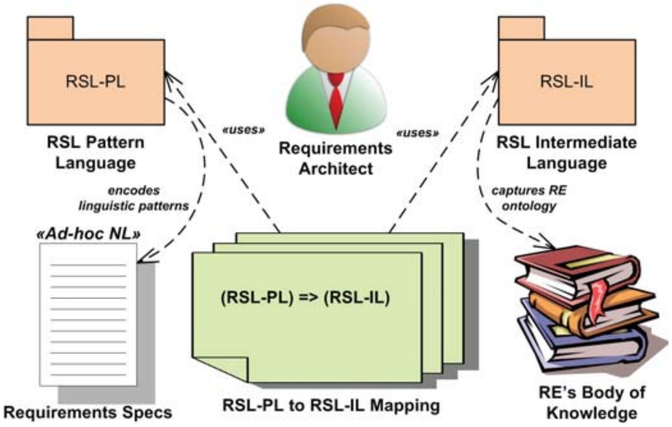


Figure 1. Overview of the RSLingo approach at process-level.

*2) **RSLingo's Project-Level:*** as illustrated in Figure 2, the approach is not disruptive with regard to traditional RE approaches [1]. During the requirements specification activity, we consider two main roles: the *Requirements Engineer* and the *Business Stakeholder*. RE is social and collaborative by nature, thus we value the direct contribution of Business Stakeholders. Those playing the role of Business Stakeholder must be acquainted with the problem-domain, which makes them valuable requirements sources and thus crucial during the requirements validation activity [14]. Therefore, the RSLingo approach encourages Business Stakeholders to directly author requirements themselves. Within this collaborative environment, the purpose of the Requirements Engineer role is to facilitate the requirements specification process and help Business Stakeholders to discover their *real* requirements [2].

According to the information flow represented in Figure 2, both the Requirements Engineer and the Business Stakeholder roles contribute with textual inputs to the requirements specification (written in natural language), which are depicted as `Requirements Specs` with the `ad hoc natural language` stereotype. Additionally, these stakeholders should follow RE's best practices of maintaining a project-specific `Glossary`. This sort of structured dictionary is of paramount importance because it establishes

[1] For a matter of simplicity, we henceforth refer to all of these types of requirements representation media as "*requirements specifications*".

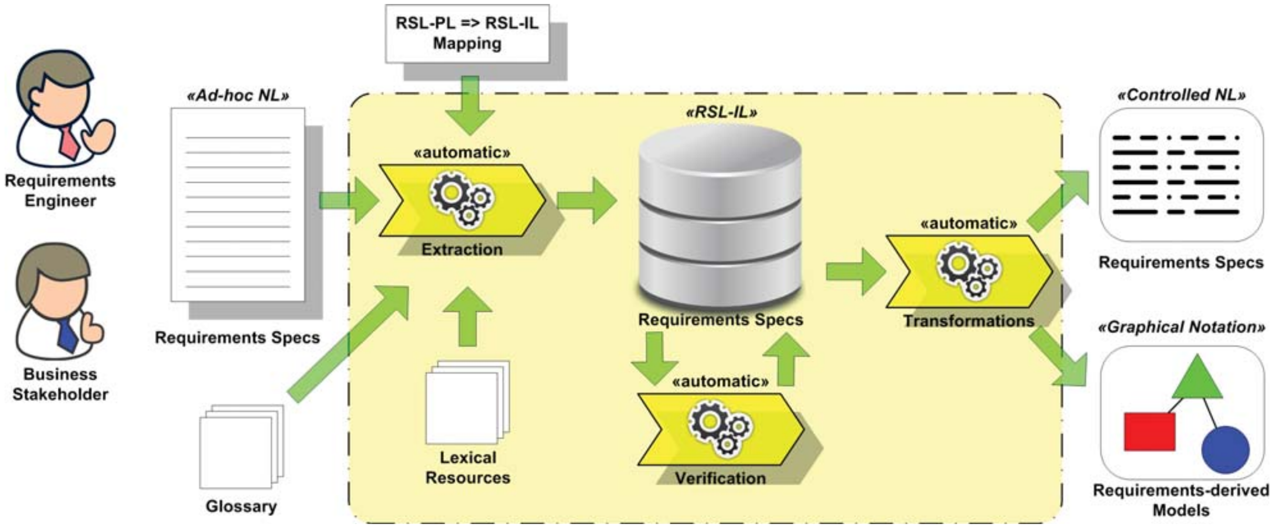


Figure 2. Overview of the RSLingo approach at project-level.

a common vocabulary for key problem-domain terms, which should be consistently used in a crosswise manner throughout the project, since they help all stakeholders to reach a shared understanding of those terms. Finally, the remaining manually created artifact to be considered by the RSLingo approach, required as an input to the RSLingo's toolset, is the `RSL-PL => RSL-IL Mapping` defined by the Requirements Architect at RSLingo's Process-Level.

Besides the sentence-level *syntax–semantic alignment* addressed by chunk parsing techniques, one must also take into consideration the word-level *lexical–semantic alignment*. Thus, to circumvent the lack of world knowledge, and in addition to the three manually made artifacts previously mentioned, the RSLingo toolset requires other `Lexical Resources`, such as WordNet[2] and VerbNet[3] lexical databases. These `Lexical Resources` are needed because RSLingo deals with *ad hoc* natural language. Without these word-level resources it would be hard to "fully understand" requirements, namely to support disambiguation tasks during the automatic extraction process, and provide additional information on the meaning of terms and their lexical relationships. However, in order to enable some flexibility regarding the definition of project-specific terms, the additional information provided by these `Lexical Resources` is overridden by the terms defined in the previously mentioned `Glossary` artifact.

In a nutshell, the RSLingo toolset works as follows: when a best-fit match occurs between a linguistic pattern (defined in RSL-PL) and a textual requirements representation, the captures identified during the match provide relevant information considering the syntactic–semantic alignment of the recognized linguistic patterns. After yielding a best-fit match, a translation from each of the match's captures to the semantically equivalent RSL-IL formal structures takes place, as described in Information Extraction literature [11].

III. Discussion

RSLingo clearly contrasts with the common practice of only dealing with the organization of requirements (e.g., document structure and requirements relations) and metadata (e.g., requirements attributes) in databases or diagrams – the "black-box" approach –, around which academia and industry seem to have a greater concern. We regard documentation best practices as being of the utmost importance [6], yet computers should be able to automatically extract relevant information pertaining to the applicational domain of the system to be built, which is captured through the semantics of words and phrases of their textual representations.

For addressing this matter, an alternative would be using Controlled Natural Languages (CNL), such as Attempto Controlled English (ACE) [10]. CNLs are subsets of natural languages whose grammars and vocabularies have been engineered in order to reduce or eliminate both ambiguity (to improve computational processing) and complexity (to improve readability). By completely avoiding ambiguity, CNLs can be mapped to structures equivalent to First-Order Logic formulas. CNLs are typically designed for knowledge engineering, thus not addressing behavioral aspects as required in RE. Additionally, they are *easy to read*, yet *hard to write*, because writing with CNLs resemble programming with a high-level programming language with strict grammar rules, hence it is easy for untrained users to become frustrated while using it. Therefore, CNLs require significant effort and specialized tools (such as predictive editors) for creating (or adapting) language-compliant specifications.

[2] http://wordnet.princeton.edu/

[3] http://verbs.colorado.edu/verb-index/

While recognizing the importance of controlled natural languages within other fields, the obstacles they pose within RE's collaborative work environments are serious impediments to their adoption by untrained users. These limitations prevent domain experts from directly contributing with their own requirements text, which is why we propose a flexible linguistic approach based on Information Extraction [11].

RSLingo follows an approach similar to the CIRCE project [15]. CIRCE provides a "lightweight formal method", supported by model-checking techniques, to produce a validation of the requirements specifications written in natural language, and its toolset provides feedback to the user with a multi-perspective approach. To this end, CIRCE uses fuzzy-matching parsing to extract knowledge from requirements specifications, which is then used to generate different views to analyze the information captured from requirements text.

However, the RSLingo approach goes further in the conceptual distinction between the two different concerns to be considered: defining linguistic patterns and formally specifying requirements knowledge. This separation of concerns is addressed by two domain-specific languages: RSL-PL and RSL-IL, respectively. The higher abstraction provided by these two languages makes them more user-friendly for those playing the Requirements Architect role. Also, this separation of concerns allows one to evolve the set of recognized linguistic patterns in RSL-PL while maintaining the requirements knowledge already encoded in RSL-IL specifications. Being a fixed language, RSL-IL provides a sound and stable knowledge base of requirements specifications, supporting reuse of specifications independently of RSL-PL extensions. Overall, these two languages (and the mapping between them) provide a requirements specification approach – RSLingo – endowed with extensibility and reusability mechanisms.

IV. Conclusion

In this paper we overview RSLingo, a linguistic approach to extract and formally specify RE-specific information from requirements specifications written in natural language. RSLingo follows a multi-language strategy, since its operationalization relies on two purpose-specific languages, RSL-PL and RSL-IL, and the mapping between them.

Unlike other requirements specification approaches, which treat requirements representations as "opaque entities", the RSLingo approach allow us to gain a deeper understanding of the system to build through NLP techniques.

As future work, we plan to conduct laboratory-controlled case studies to validate the RSLingo approach, because we still need to evaluate it according to the users' perspective. The next step in this research roadmap is to exploit the potential of RSLingo in producing alternative representations from requirements specifications formally encoded in RSL-IL, besides textual requirements paraphrases.

References

[1] K. Pohl, *Requirements Engineering: Fundamentals, Principles, and Techniques*, 1st ed. Springer, July 2010, ISBN-13: 978-3642125775.

[2] R. Young, *The Requirements Engineering Handbook*, 1st ed. Artech Print on Demand, November 2003, ISBN: 978-1580532662.

[3] INCOSE, "Requirements Management Tools Survey," Retrieved Monday 12th March, 2012 from http://www.incose.org/ProductsPubs/products/rmsurvey.aspx.

[4] E. Gottesdiener, *The Software Requirements Memory Jogger: A Desktop Guide to Help Software and Business Teams Develop and Manage Requirements*, 1st ed. Goal / QPC, October 2009, ISBN-13: 978-1576811146.

[5] IEEE Computer Society, "IEEE Recommended Practice for Software Requirements Specifications," *IEEE Std 830-1998*, August 1998.

[6] B. Kovitz, *Practical Software Requirements: Manual of Content and Style*. Manning, July 1998.

[7] D. C. Schmidt, "Guest Editor's Introduction: Model-Driven Engineering," *Computer*, vol. 39, no. 2, pp. 25–31, February 2006.

[8] H. Foster, A. Krolnik, and D. Lacey, *Assertion-based Design*. Springer, 2004, ch. 8 - Specifying Correct Behavior.

[9] D. C. Gause and G. M. Weinberg, *Exploring Requirements: Quality Before Design*. Dorset House Publishing Company, Incorporated, September 1989.

[10] N. Fuchs, U. Schwertel, and R. Schwitter, "Attempto Controlled English – Not Just Another Logic Specification Language," in *Logic-Based Program Synthesis and Transformation*, P. Flener, Ed., Eighth International Workshop LOPSTR'98. Manchester, UK: Springer, June 1999, pp. 1–20.

[11] H. Cunningham, "Information Extraction, Automatic," in *Encyclopedia of Language & Linguistics*, 2nd ed. Elsevier, 2006, vol. 5, pp. 665–677.

[12] A. Davis, *Just Enough Requirements Management: Where Software Development Meets Marketing*, 1st ed. Dorset House Publishing, May 2005.

[13] S. Bird, E. Klein, and E. Loper, *Natural Language Processing with Python*, 1st ed. O'Reilly Media, June 2009, ISBN-13: 978-0596516499.

[14] K. Pohl and C. Rupp, *Requirements Engineering Fundamentals: A Study Guide for the Certified Professional for Requirements Engineering Exam – Foundation Level – IREB compliant*, 1st ed. Rocky Nook, April 2011, ISBN-13: 978-1933952819.

[15] V. Ambriola and V. Gervasi, "On the Systematic Analysis of Natural Language Requirements with CIRCE," *Automated Software Eng.*, vol. 13, no. 1, pp. 107–167, January 2006.

DETECTING EMERGENT BEHAVIOR IN DISTRIBUTED SYSTEMS CAUSED BY OVERGENERALIZATION

Seyedehmehrnaz Mireslami *Mohammad Moshirpour* *Behrouz H. Far*

Department of Electrical and Computer Engineering, University of Calgary, Canada
{smiresla,mmoshirp,far}@ucalgary.ca

Abstract— **Distributed system design faces many challenges due to lack of central control. Emergent behavior is a vital problem in distributed systems and leads to unexpected behaviors and major faults. Emergent behaviors are usually categorized into two groups: emergent behaviors occur due to scenarios incompleteness and emergent behaviors that are produced as a result of synthesis of behavior models. In this paper, the methods for detecting the second group of emergent behaviors are studied and a technique for synthesis of behavior models from scenarios is developed to be employed for detecting emergent behaviors. The proposed technique detects overgeneralization in the behavior model synthesis. Overgeneralization happens as the result of behavior model synthesis and depends on the assumptions of the process. In addition, the proposed technique addresses the issue of the existing ad-hoc methodologies by providing an automated algorithm. This algorithm can be used by a syntax checker to automatically detect the emergent behaviors in the scenarios. The proposed algorithm is validated using a case study of a mine sweeping robot.**

Keywords; Distributed systems; Emergent behavior; Message sequence chart; Overgeneralization.

I. INTRODUCTION

A distributed system is composed of several components without a central controller [1]. Distributed systems specification is usually composed of a set of scenarios that show system components and messages sent between them. Scenarios are generally shown using Message Sequence Charts (MSCs) or sequence diagrams [2, 3]. Behavior of the components can be generated using the scenarios.

A popular approach for analyzing the behavior of system components, is going from scenarios to state machines, i.e. synthesis of behavior models [4-6]. Each component is modeled by a state machine and the messages that are sent and received determine the states and transitions.

In the process of behavior model synthesis, several behaviors not originally intended for system may happen that are called em ergent behaviors [8-10]. They are eith er unexpected situations that may be caused by incompleteness of the scenarios or produced as the results of behavior model synthesis. Detecting emergent behavior in early design stages reduces the deployment costs significantly [11].

There are two types of emergent behaviors: first group is detected by comparing the behavior models and scenario specifications by considering the properties of the system. The second group is result of overgeneralization [7]. Hence, it depends on the rules and assumptions of the behavior model synthesis.

Only a few works have targeted the overgeneralization problem. In [7], synthesis of state machines from scenarios is proposed. However, the relationships between scenarios are rather ambiguously defined. Blending scenarios is necessary as it provides a comprehensive overview of system behavior. Two methods are proposed to combine the scenarios: state identification and scenario composition. In state identification [5, 6, 12], the components are f irst modeled with different states in the state machines and similar component states a re identified and combined to enable the scenarios to merge. In [2], another approach for merging scenarios is proposed where scenarios are split to smaller parts with lower complexity and high-level MSC graphs are used to blen d the smaller sequence of behavior since they are simpler to manage.

In [13], a tech nique to identify the identical states that may cause emergent behaviors is proposed and a method to synthesize emergent behaviors by merging identical states is presented. This leads to detecting emergent behavior due to presence of identical states. However, although identical states are the potential causes of emergent behavior, n ot all identical states may lead to e mergent behavior. The shortcomings of this technique are: first, all identical sta tes are used for merging the state m achines, and overgeneralization is inevitable; and second, it relies on designer to decide which states causes emergent behavior.

As ad-hoc methods are unreliable and in efficient, in this paper, we replace th e ad-hoc methodology in [13] by an automated method. A technique is devised to distinguish the identical states that cause emergent behaviors. To have a fair comparison with [13], the same mine sweeping robot system used in [13] is considered.

The rest of this paper is organized as follows: In Section II, a case study to compare this work and previous works is presented. In Sections III and IV, behavioral modeling and emergent behavior detection are discussed and our new algorithm is presented. Finally, in Section V, conclusions and future work are given.

II. CASE STUDY

To obtain a fair comparison between the proposed technique and the technique presented in [13], the same prototype of mine sweeping robot is used. The robot's objective is to pass 10km maze-based routes while identifying and flagging the existing mines. The robot is not given any map and just takes advantage of the information received from its sensors. The robot has four sensors: a GPS sensor that operates to control the certain distance (about 10km) robot is allowed to pass, an infrared (IR) sensor to identify mines, a battery sensor that checks if the battery can provide the power and an ultra-sonic (ULT) sensor which helps the robot to navigate the routes by finding the obstacles. The messages from these sensors are sent to the client control to be processed and sent to the server control in order to take necessary actions.

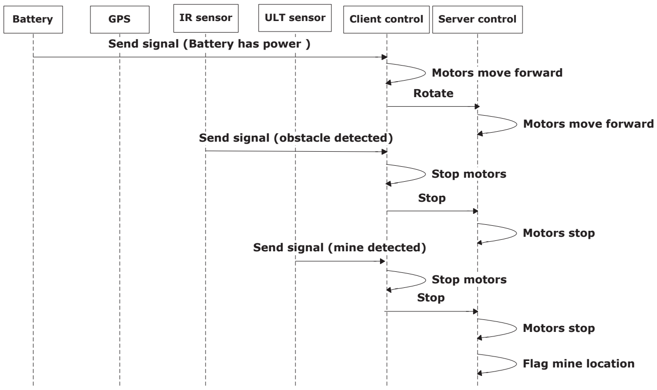


Figure 1. MSC m1

Two scenarios that represent the behavior of the robot are presented. In Figure 1, MSC m1 shows where robot moves since the battery has power. Later, based on the message sent from IR sensor, the motor stops due to detection of an obstacle. Then, ULT sensor sends a message indicating detection of a mine but the robot has already stopped and so the mine is just flagged.

In Figure 2, the robot moves since the ULT sensor, infrared and battery sensors do not cause the robot to stop. As in [13], messages are generalized to represent objective instead of implementation. For example, there are three messages in Figure 1, which have the same purpose of "send signal" where the contents of the messages are different.

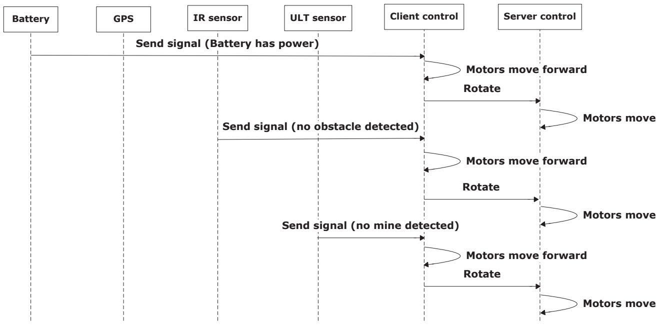


Figure 2. MSC m2

III. BEHAVIORAL MODELING

In this paper, the process of converting the scenarios into the corresponding state machines based on the definitions given in [13] is performed. These definitions are then applied to the case study to validate their effectiveness.

As proposed in [13], in a message sequence chart MSC, Finite State Machines (FSMs) are built for any component *i*. In Figure 3, the FSMs for the client control component in MSC m1 (Figure 1) are shown. $s_0^{m1}, s_i^{m1}, s_f^{m1}$ are the initial state, the i^{th} state and the final state of the component client control in MSC m1, respectively.

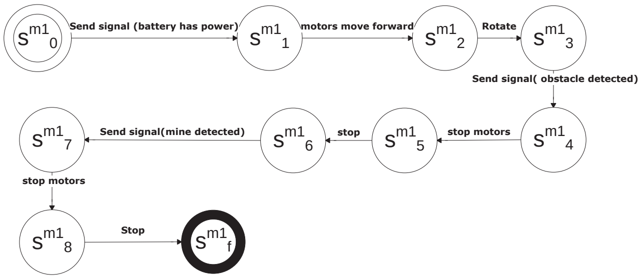


Figure 3. FSM for component client control of MSC m1

Then, a certain value is given to each state of the FSMs of each component. This technique, is based on the definitions initially presented in [7].

Definition 1: If component *i* needs the result of message $m|_i[j]$ for doing $m|_i[k]$, then message $m|_i[j]$ is a semantical cause for message $m|_i[k]$ and is shown by $m|_i[j] \overset{se}{\rightarrow} m|_i[k]$, where $m|_i[j]$ represents the j^{th} message interacted by the component *i* of MSC m.

The semantic cause is an invariant feature of the system. As an example, in Figure 3, the message "send signal (mine detected)" is a semantic cause for performing message "stop". Each state of a component is defined by considering the messages that are semantic causes for messages that come after it. The semantic cause is resulted from the domain knowledge and defined as:

Definition 2: For a set of MSCs M, the domain theory D_i for each component i is defined as: if $m|_i[j] \overset{se}{\rightarrow} m|_i[k]$, then the pair $(m|_i[j], m|_i[k])$ is in the domain theory D_i.

From Definition 2, (" send signal (mine detected)", "stop") from Figure 3 is in the domain theory. This is used in [7] to define a method to quantify the states.

In [6], several variables are defined to differentiate the states of a component. However, it leads to having different behaviors when producing various behavior models for a single component. Choosing between these models is hard as they cannot be compared. In [13], it is proposed to use the invariant properties of the system to find a unique way of calculating the state values as described below:

Definition 3: In the finite state machine shown with tuple $A_i^m = (S^m, \Sigma^m, \delta^m, s_0^m, s_f^m)$, for the final state s_f^m the state value is calculated as: $v_i|(s_f^m) = m\ |_i\ [f - 1]$, and for $0 < k < f$ the state value is defined as follows:

i) $v_i|(s_k^m) = m|_i\ [k - 1]v_i|(s_j^m)$, if there exist some j and l such that j is the maximum index that $m|_i[j - 1] \overset{se}{\rightarrow} m|_i[l], 0 < j < k,\ k \le l < f$.

ii) $v_i|(s_k^m) = m|_i\ [k - 1]$ if case i) does not hold but $m[k - 1] \overset{se}{\rightarrow} m|_i[r]$, for some $k \le r < f$.

iii) $v_i|(s_k^m) = 1$, if none of the above cases hold.

The value of s_k^m is related to message that comes after. For case i), the semantic cause is $m|_i[j - 1], 0 < j < k$. For case ii), $m|_i[k - 1]$ is the only semantic cause. Finally, there is not any semantic cause for case iii). The Order of messages is used to achieve state values.

The state values that are found using Definition 3 are effective for analyzing the system behaviors. After constructing the FSMs from various scenarios or a single scenario, to clearly analyze the system behavior, the FSMs for each component are blended. Therefore, the concept of identical states is defined as:

Definition 4: For each component i, two states s_j^m and s_k^n form the MSCs m and n (m can be equal to n) are identical states if any of following holds:

i) $j = k$ for $0 \le l \prec j: m|_i[l] = n|_i[l]$.

ii) $v_i(s_j^m) = v_i(s_k^n)$.

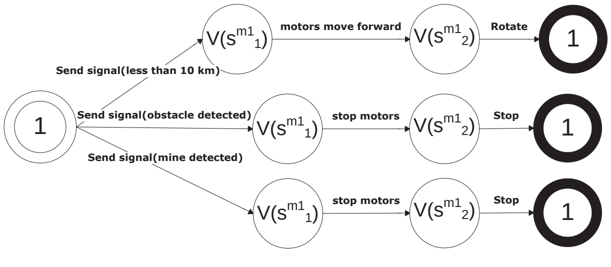


Figure 4. Blending FSMs for client control of MSC m1

Following the example in Figure 3, the FSMs are blended in Figure 4 where states are assigned with appropriate state values. Initial and final state values are 1.

The presence of identical states in the behavior models may result in emergent behavior since the component may be confused when the next message is received. Therefore, dealing with these issues is an important challenge in analyzing the behavior models. Once the identical states are found, the finite state machines are merged by merging the found identical states. In Figure 3, emergent behavior occurs for component client control as a result of identical states $s_0^{m1}, s_3^{m1}$and s_6^{m1} in MSC m1 when the content of "send signal" messages are not considered. These identical states are then merged as shown in Figure 5.

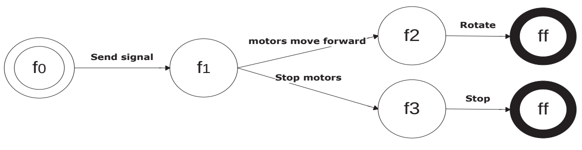


Figure 5. Merging identical states of FSMs for client control of MSC m1

IV. DETECTING EMERGENT BEHAVIOR

The method proposed in [13] merges all the identical states without considering whether they may produce emergent behaviors or not. This is an important issue since merging the identical states that do not produce emergent behaviors results in overgeneralization in the behavior models. Furthermore, merging all the identical states takes too much unnecessary time and resources.

In the FSMs of MSC m1 (Figure 3), identical states leading to emergent behavior are shown. These states are identified and merged in Figures 4 and 5, respectively. In Figure 6, another example is considered where the FSM for MSC m2 (presented in Figure 2) is given. Then, states values are found and the identical states are identified to be s_0^{m2}, s_3^{m2} and s_6^{m2}. However, these states do not lead to emergent behaviors as the component never gets confused in performing the messages. Hence, merging is not necessary.

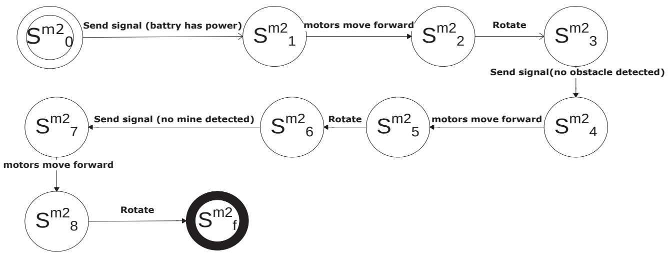


Figure 6. FSM for component client control of MSC m2

Since most of the existing approaches merge all of the identical states, they result in overgeneralized FSMs. In [7], some criteria are proposed to identify identical states that have emergent behavior. This definition is as follows:

Definition 5: The component i in MSC m has emergent behavior in state s_j^m, if there exists MSC n (m and n can be equal) and a state s_k^n, such that s_j^m and s_k^n are identical states and one of the following holds:

i) $m|_i[j] \neq n|_i[k] = i!\,l(c)$ where l is a component and $i!\,l(c)$ represents message with content c sent from i to l.

ii) $m|_i[j] \neq n|_i[k] = i?\,l(c)$ where l is a component and $i?\,l(c)$ represents a message with content c received by i from l. Component l sends a message with content c to component i ($l!\,i(c)$) such that i does not receive this message before event of $m|_i[j]$ in MSC m and by removing this event, still component l can send $l!\,i(c)$.

iii) State s_k^n is the final state of MSC n and $m|_i[j] = i!\,l(c)$ is a send message for component i.

iv) Case ii) holds except that by removing event of $m|_i[j]$ in MSC m, component l cannot send $l!\,i(c)$ anymore. Then, there exist events e and w for l such that the event of $m|_i[j]$ is syntactical cause for $m|_l[e]$. Then, states s_{e-1}^m and s_{w-1}^n are identical and emergent behavior occurs for l.

Only the identical states that result in emergent behaviors based on Definition 5 should be merged and unnecessary merging for the other states should be avoided. Behavior Model algorithm (Figure 7), is proposed to synthesize behavior models while preventing emergent behaviors due to overgeneralization.

```
Algorithm: Behavior Model
Input: Set of message sequence charts (M)
Output: State machines for each component

1.  For each component i:
2.  Make domain theory D_i based on Definition 2.
3.  Make state machines and assign state values based on Definition 3.
4.  Find identical states based on Definition 4.
5.  For each set of identical states:
        a.  If they lead to emergent behavior based on Definition 5:
            Merge the states to one single state.
        b.  Otherwise, no merging is needed.
```

Figure 7. Synthesis of emergent behavior preventing overgeneralization

To show the effectiveness of Definition 5 and the proposed algorithm, the case study is considered. MSC m1 has emergent behavior since case i) of Definition 5 holds for it. For the FSMs of client control in MSC m2 (shown in Figure 6) the proposed algorithm is applied and the identical states are found to be s_0^{m2}, s_3^{m2} and s_6^{m2}. However, these states do not lead to emergent behavior since they do not hold any of four cases in Definition 5. The reason is that in cases ii) and iv) of Definition 5, the event that comes after the identical states should be a receive message while in this example, the event is a send message. In cases i) and iii) , although the event that comes after identical states is sending message which matches the example, one of the identical states should be the final state in the machine or the next event should be different for the identical states in order to produce emergent behavior. Since none of these is valid for the example, the identical states of MSC m2 do not lead to emergent behavior and merging them is unnecessary.

V. CONCLUSIONS AND FUTURE WORK

It has been reported that detecting unwanted behavior during the design phase is 20 times cheaper than finding them during the deployment phase [11]. This paper provides a systematic approach to analyze system requirements for defects, while saving on overhead by replacing ad-hoc methodologies with automated ones. A new algorithm is developed for behavior model synthesis and emergent behavior detection while preventing overgeneralization. The proposed algorithm improves the existing ad-hoc methods [7, 13]. The future work is implementing the proposed algorithm as a syntax checker to provide an automated tool to check and correct the system designs. Moreover this work can be utilized as part of a comprehensive framework to analyze system requirements and design.

REFERENCES

[1] M. Moshirpour, "Model-Based Detection of Emergent Behavior In Distributed and Multi-Agent Systems from Component Level Perspective," in *Dept. of Electrical and Computer Engineering*. Master of Science Thesis. University of Calgary, 2011.

[2] "ITU: Message Sequence Charts. Recommendation, International Telecommunication Union," 1992.

[3] "Recommendation Z.120: Message Sequence Chart," Geneva, 1996.

[4] D. Harel and H. Kugler, "Synthesizing state-based object systems from lsc specifications," *IJFCS,* 2002.

[5] I. Kruger, R. Grosu, P. Scholz, and M. Broy, "From mscs to statecharts," in *Franz j. rammig (ed.): Distributed and parallel embedded systems*: Kluwer Academic Publis, 1999.

[6] J. Whittle and J. Schumann, "Generating statecharts designs from scenarios," in *ICSE* Limerick, Ireland, 2000.

[7] A. Mousavi, "Inference of Emergent Behaviours of Scenario-Based Specifications," in *Dept. of Electrical and Computer Engineering*., PhD Thesis: University of Calgary, 2009.

[8] J. Grabowski, "Test Generation and Test Case Specification with Message Sequence Charts," in *Institute for Informatics and Applied Mathematics*: Universitat Bern, 1994.

[9] H. Muccini, "Detecting implied scenarios analyzing nonlocal branching choices," in *FASE 2003* Warsaw, Poland.

[10] S. Uchitel, J. Kramer, and J. Magee, "Negative scenarios for implied scenario elicitation," in *FSE 2002*.

[11] R.F. Goldsmith, *Discovering Real Business Requirements for Software Project Success*. Norwood MA: Artech House, Inc., 2004.

[12] K. Koskimies, T. Mannisto, T. Systa, and J. Tuonmi, "Automated support for modeling oo software," *IEEE Software,* pp. 15(1):87–94, 1998.

[13] M. Moshirpour, A. Mousavi, and B.H. Far, "Detecting Emergent Behavior in Distributed Systems Using Scenario-Based Specifications," in *Proc. of SEKE,* 2010, pp. 349-354.

Stability of Filter-Based Feature Selection Methods for Imbalanced Software Measurement 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

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

***Abstract*—Feature selection (FS) is necessary for software quality modeling, especially when a large number of software metrics are available in data repositories. Selecting a subset of features (software metrics) that best describe the class attribute (module's quality) can bring many benefits such as reducing the training time of learners, improving the comprehensibility of the resulting classifier models, and facilitating software metrics collection, organization, and management. Another challenge of software measurement data is the presence of skewed or imbalanced distributions between the two types of modules (e.g., many more not-fault-prone modules than fault-prone modules found in those datasets). In this paper, we use data sampling to deal with this problem. Previous research usually evaluates FS techniques by comparing the performance of classifiers before and after the training data is modified. This study assesses FS techniques from a different perspective: stability. Stability is important because FS techniques that reliably produce the same features are more trustworthy. We consider six filter-based feature selection methods and six data sampling approaches. We also vary the number of features selected in the feature subsets. We want to examine the effect of data sampling approaches on the stability of FS when using the sampled data. The experiments were performed on nine datasets from a real-world software project. The results demonstrate that different FS techniques may have quite different stability behaviors. In addition, other factors, such as the sampling technique used and the number of attributes retained in the feature subset, may also greatly influence the stability results.**

***Index Terms*—software defect prediction, software metrics, feature selection, data sampling, stability**

I. Introduction

Software defect prediction is a process of building a classifier by using software metrics and fault data collected during the previous software project and then applying this classifier to predict the quality of new program modules (e.g., classify the program modules as either fault-prone (*fp*) or not-fault-prone (*nfp*)) [1]. The benefit of such prediction is that the project resources can be strategically allocated to the program modules according to the prediction. For instance, intensive inspection and testing can first be applied to the potentially problematic modules, thereby improving the quality of the product.

Two problems that often come with the software measurement data are high-dimensionality and class imbalance. *High-dimensionality* refers to the situation where the number of available software metrics is too large to easily work with. Several problems may arise due to high-dimensionality, including longer learning time of a classification algorithm and a decline in prediction performance of a classification model. *Class imbalance* occurs when instances of one class in a dataset appear more frequently than instances of the other class. This phenomenon is more prevalent in high-assurance and mission-critical software systems, where the type of *nfp* modules is always dominant between the two types (*fp* and *nfp*) of modules in a given dataset. The primary weakness of such imbalanced data is that a traditional classification algorithm tends to classify *fp* modules as *nfp*, resulting in more customer-discovered faults that have serious consequences and high repair cost.

Feature selection and data sampling are often employed to deal with these problems. *Feature selection* (*FS*) is a process of choosing a subset of input variables by eliminating features with little or no predictive information. Although FS techniques have been studied in a variety of domains [2], [3] for many years, research working on improving software defect prediction through metric (feature) selection just started recently [4], [5]. *Data sampling* is a common technique to alter the relative proportion of the different types of modules, therefore achieving a more balanced dataset. Note that in this study, the training dataset is sampled to change the relative proportion of the *nfp* and *fp* modules before FS is performed.

To evaluate a FS technique, most previous research focuses on comparing the performance of classification models before and after a specific FS technique is performed. In this paper, we use a different way to assess FS techniques – *stability*. Stability of a FS technique usually refers to the sensitivity of the technique to variations in the training set. Practitioners may prefer a FS algorithm that can produce consistent results despite such variations. For example, if a FS technique produces the same or similar results when using the entire training dataset or only half of it, a practitioner may save the computation time and use this FS technique on the smaller training dataset to get the same reliable results.

In this study, we are more interested in investigating the stability of FS techniques with respect to various data sampling approaches. The strategy we adopted is that the ranking of features from each sampled dataset is compared to the ranking from the original dataset which it came from. Those FS techniques that are able to produce consistent outputs with respect to the different perturbations (due to sampling) in the input data are considered stable (robust), or say they are insensitive to that particular data sampling technique. Since the purpose of data sampling here is to alter class proportions rather than changing the size of the training dataset, the consistent outputs for feature rankings imply that the data sampling technique has less or no effect on the FS technique. To our knowledge, limited research has been done on studying the impact of data sampling on the stability of feature selection.

The case study of this paper is performed on nine datasets from a real-world software project. We examine six filter-based FS methods, including five threshold-based techniques (mutual information (MI), Kolmogorov-Smirnov statistic (KS), geometric mean (GM), area under the ROC curve (AUC), and area under the precision-recall curve (PRC)), and the signal-to-noise ratio (S2N) approach. We employ three data sampling techniques (random undersampling (RUS), random oversampling (ROS), and synthetic minority oversampling (SMO)), each combined with two post-sampling class ratios. Besides, we vary from 2 to 10 the number of features retained in the feature subsets. The empirical results demonstrate that S2N, AUC, and PRC show higher stability performance than the other filters. Moreover, the filters with the ROS sampling technique have higer stability behavior than they do with the other sampling methods. Finally, the stability of the FS techniques increases as the number of attributes retained in the feature subset increases.

The remainder of the paper is organized as follows: Section II discusses related work. Section III outlines the methods and techniques used in this paper. Section IV describes nine datasets used in the case study. Section V presents the case study including design, results, and analysis. Finally, we summarize our conclusions and provide suggestions for future work in Section VI.

II. Related Work

Feature selection (FS), also known as attribute selection or variable selection, is a process of selecting some subset of the features which are useful in building a classifier. FS techniques can be divided into wrapper and filter categories [3]. *Wrappers* use a search algorithm to search through the space of possible features, evaluate each subset through a learning algorithm, and determine which ones are finally selected in building a classifier. *Filters* use a simpler statistical measure to evaluate each subset or individual feature rather than using a learning algorithm. Feature selection may also be categorized as ranking or subset selection [3]. Feature *ranking* scores the attributes based on their individual predictive power, while *subset selection* selects subset of attributes that collectively have good prediction capability. In this study, the FS techniques used belong to the *filter-based feature ranking* category.

Class imbalance, which appears in various domains, is another significant problem in data mining. One effective method for alleviating the adverse effect of skewed class distribution is sampling [6], [7]. While considerable work has been done for feature selection and data sampling separately, research on investigating both together started recently. Chen et al. [8] 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 specifically for the class imbalance problem, and also the classification models were not for binary problems.

To evaluate FS techniques, most existing research works on comparing the classification behaviors of models built with the selected features to those built with the complete set of features. Instead of using classification performance, the present work assesses FS techniques using stability. The stability of a FS algorithm is normally defined as the degree of consensus between the output of that FS method as it pertains to randomly-selected subsets of the same input data. Lustgarten et al. [9] presented an adjusted stability measure that computes robustness of a FS method with respect to random FS. Saeys et al. [10] assessed the robustness of FS techniques using the Spearman rank correlation coefficient and Jaccard index. Abeel et al. [11] presented a general framework for stability analysis of the FS techniques. They showed that stability could be improved through ensemble FS. Alelyani et al. [12] jointly considered both sample sets' similarity and feature list similarity in stability assessment for FS algorithms.

III. Methodology

A. Filter-based feature ranking techniques

The procedure of filter-based feature ranking is to score each feature (attribute) according to a particular method (metric), allowing the selection of the best set of features. In this study, we use five threshold-based feature selection techniques and the signal-to-noise ratio method.

1) Threshold-based feature selection (TBFS) methods: The TBFS techniques were 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 dataset 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 classifier performance metrics.

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 $\hat{X}^j$. Precision $PRE(t)$ is defined as the fraction of the predicted-positive examples which are actually positive. The feature rankers utilize five metrics: Mutual Information (MI), Kolmogorov-Smirnov Statistic (KS), Geometric Mean (GM), Area Under the ROC Curve (AUC), and Area Under the Precision-Recall Curve (PRC). 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. Five metrics are calculated for each attribute individually, and attributes with higher values for MI, KS, GM, AUC, and PRC 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 five metrics. For more detailed information about these five metrics, please reference the work of Dittman et al. [2].

2) Signal-to-Noise Ratio (S2N) Technique: S2N represents how well a feature separates two classes. The equation for signal-to-noise is:

$$S2N = (\mu_P - \mu_N)/(\sigma_P + \sigma_N) \tag{1}$$

where μ_P and μ_N are the mean values of that particular attribute in all of the instances which belong to a specific class, which is either P or N (the positive and negative classes). σ_P and σ_N are the standard deviations of that particular attribute as it relates to the class. The larger the S2N ratio, the more relevant a feature is to the dataset [13].

B. Data Sampling Techniques

We here present three sampling techniques, which stand for the major paradigms in data sampling: random and intelligent under and oversampling.

1 Random Sampling Techniques
The two most common data sampling techniques are random oversampling (ROS) and random undersampling (RUS). *Random oversampling* duplicates instances (selected randomly) of the minority class. *Random undersampling* randomly discards instances from the majority class.

2 Synthetic Minority Oversampling Technique
Chawla et al. proposed an intelligent oversampling method called Synthetic Minority Oversampling Technique (SMOTE) [6]. SMOTE (denoted SMO in this work) adds new, artificial minority examples by extrapolating between preexisting minority instances rather than simply duplicating original examples. The newly created instances cause the minority regions of the feature-space to be fuller and more general.

Algorithm 1: Threshold-Based Feature Selection Algorithm

```
input :
    a. Dataset $D$ with features $X^j, j = 1, \dots, m$;
    b. Each instance $x \in D$ is assigned to one of two classes $c(x) \in \{P, N\}$;
    c. $|P| = |\{x \in D | c(x) = P\}|$, $|N| = |\{x \in D | c(x) = N\}|$;
    d. The value of attribute $X^j$ for instance $x$ is denoted $X^j(x)$;
    e. Metric $\omega \in$ {MI, KS, GM, AUC, PRC}.
output: Ranking $\mathbb{R} = \{r^1, r^2, \dots, r^m\}$ where attribute $X^j$ is the $r^j$-th most significant attribute as determined by metric $\omega$.
for $X^j, j = 1, \dots, m$ do
    Normalize $X^j \mapsto \hat{X}^j = \frac{X^j - \min(X^j)}{\max(X^j) - \min(X^j)}$, $\hat{X}^j \in [0, 1]$;
    for $t \in [0, 1]$ do
        Compute Basic Metrics:
        Classification Rule 1: $\forall\, x \in D, \hat{c}^t(x) = P \iff \hat{X}^j(x) > t$, otherwise $\hat{c}^t(x) = N$.
        $TP(t) = \left|\{x | (\hat{c}^t(x) = P) \cap (c(x) = P)\}\right|,\ TN(t) = \left|\{x | (\hat{c}^t(x) = N) \cap (c(x) = N)\}\right|$,
        $FP(t) = \left|\{x | (\hat{c}^t(x) = P) \cap (c(x) = N)\}\right|,\ FN(t) = \left|\{x | (\hat{c}^t(x) = N) \cap (c(x) = P)\}\right|$,
        $TPR(t) = \frac{|TP(t)|}{|P|},\ TNR(t) = \frac{|TN(t)|}{|N|},\ FPR(t) = 1 - TNR(t),\ FNR(t) = 1 - TPR(t)$,
        $PRE(t) = \frac{|TP(t)|}{|TP(t)| + |FP(t)|},\ NPV(t) = \frac{|TN(t)|}{|TN(t)| + |FN(t)|}$.
    Compute Final Metrics:
```

Metric ω		Calculation
KS$^1(\hat{X}^j)$	=	$\max_{t \in [0,1]} \lvert TPR(t) - FPR(t) \rvert$
GM$^1(\hat{X}^j)$	=	$\max_{t \in [0,1]} \sqrt{TPR(t) \times TNR(t)}$
AUC$^1(\hat{X}^j)$	=	Area under the curve generated by $(FPR(t), TPR(t)), t \in [0, 1]$
PRC$^1(\hat{X}^j)$	=	Area under the curve generated by $(PRE(t), TPR(t)), t \in [0, 1]$
MI$^1(\hat{X}^j)$	=	$\max_{t \in [0,1]} \sum_{\hat{c}^t \in \{P,N\}} \sum_{c \in \{P,N\}} p(\hat{c}^t, c) \log \frac{p(\hat{c}^t, c)}{p(\hat{c}^t) p(c)}$ where $p(\hat{c}^t = \alpha, c = \beta) = \frac{\lvert\{x \mid (\hat{c}^t(x) = \alpha) \cap (c(x) = \beta)\}\rvert}{\lvert P \rvert + \lvert N \rvert}$, $p(\hat{c}^t = \alpha) = \frac{\lvert\{x \mid \hat{c}^t(x) = \alpha\}\rvert}{\lvert P \rvert + \lvert N \rvert}$, $p(c = \alpha) = \frac{\lvert\{x \mid c(x) = \alpha\}\rvert}{\lvert P \rvert + \lvert N \rvert}$, $\alpha, \beta \in \{P, N\}$

```
    Compute the same basic metrics and final metrics (denoted as $\omega^2$) as listed above, but using:
    Classification Rule 2: $\forall\, x \in D, \hat{c}^t(x) = N \iff \hat{X}^j(x) > t$, otherwise $\hat{c}^t(x) = P$.
    $\omega(\hat{X}^j) = \max\left(\omega^1(\hat{x}^j), \omega^2(\hat{x}^j)\right)$ where $\omega^1$ is the original basic metric
Create attribute ranking $\mathbb{R}$ using $\omega(\hat{X}^j) \forall j$
```

TABLE I
ECLIPSE DATASET SUMMARY

Group #	Rel.	thd	#Attri.	Inst. #	*nfp* #	*nfp* %	*fp* #	*fp* %
1	2.0	10	208	377	23	6%	354	94%
	2.1	5	208	434	34	8%	400	92%
	3.0	10	208	661	41	6%	620	94%
2	2.0	5	208	377	52	14%	325	86%
	2.1	4	208	434	50	12%	384	88%
	3.0	5	208	661	98	15%	563	85%
3	2.0	3	208	377	101	27%	276	73%
	2.1	2	208	434	125	29%	309	71%
	3.0	3	208	661	157	24%	504	76%

C. Stability Measure

Previous works have employed different similarity measures to evaluate the stability of feature selection techniques. The measures used include consistency index [14], Hamming distance [15], and entropy [16]. Among these similarity measures, consistency index is the only one which considers bias due to chance. For this reason, we use consistency index in this study. Assuming that the original dataset has n features, and T_i and T_j are two subsets of the n features such that $|T_i| = |T_j| = k$, where $0 < k < n$, the consistency index [14] is defined as follows:

$$I_C\left(T_i, T_j\right) = \frac{dn - k^2}{k\left(n - k\right)} \quad , \tag{2}$$

where d is the cardinality of the intersection between subsets T_i and T_j, i.e., $d = |T_i \cap T_j|$, and $I_C\left(T_i, T_j\right) \in (-1, 1]$. The greater the value of I_C, the higher the similarity between the subsets T_i and T_j.

IV. Data Description

The data is obtained from the publicly available PROMISE software project data repository [17]. We study the software measurement datasets for the Java-based Eclipse project [18]. The software metrics and defect data are aggregated at the software packages level; hence, a program module is a Java package in Eclipse. We consider three releases of the Eclipse system, where the releases are denoted as 2.0, 2.1, and 3.0 [18]. Each system release contains the following information [18]: name of the package for which the metrics are collected (*name*), number of defects reported six months prior to release (*pre-release defects*), number of defects reported six months after release (*post-release defects*), a set of complexity metrics computed for classes or methods and aggregated by using average, maximum, and total at the package level (*complexity metrics*), and structure of abstract syntax tree(s) of the package consisting of the node size, type, and frequency (*structure of abstract syntax tree(s)*).

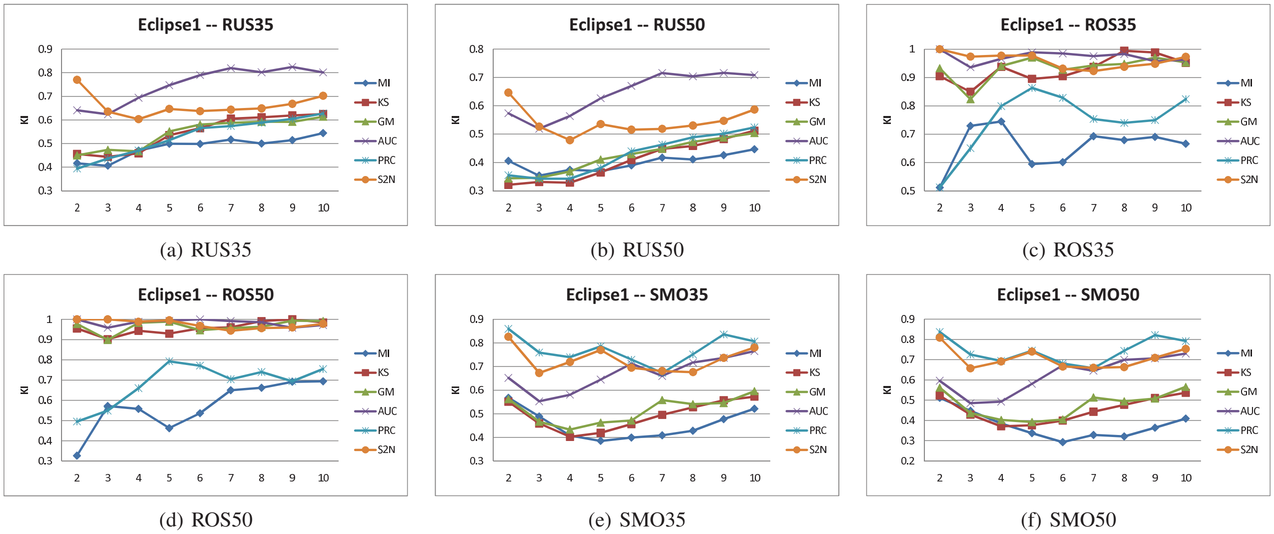
A program module's membership in a given class is determined by a post-release defects threshold, thd. A program module (package) with thd or more post-release defects is labeled as *fp*, while those with fewer than thd defects are labeled as *nfp*. In our study, we use thd={10, 5, 3} for release 2.0 and 3.0, while we use thd={5, 4, 2} for release 2.1. A different threshold is chosen for release 2.1, because we want to maintain relatively similar types of class distributions as those of Releases 2.0 and 3.0. Three groups of datasets are formed, each maintaining a different ratio between *fp* and *nfp* modules as shown in Table I.

V. A Case Study

The experiment is performed on each individual dataset separately, but the results are analyzed and summarized over the respective groups of datasets.

A. Design

As mentioned earlier, the main objective of this study is to investigate the effects of the data sampling techniques on the stability of feature selection. In the experiment, we use



(a) RUS35

(b) RUS50

(c) ROS35

(d) ROS50

(e) SMO35

(f) SMO50

Fig. 1. Eclipse1: KI Values

- six filter-based feature ranking techniques (MI, KS, GM, AUC, PRC, and S2N), and
- six different sampling approaches (RUS35, RUS50, ROS35, ROS50, SMO35, and SMO50), which are made up of three different data sampling techniques each in conjunction with two different post-sampling class ratios, where 35 standards for a 35:65 ratio between *fp* and *nfp* modules, while 50 represents a 50:50 ratio.

We design the procedure of the experiment as follows. For a given dataset and a given data sampling technique,

1) Apply the sampling technique to the original dataset, D, and get a sampled data, D_i. To avoid a biased result, repeat the sampling procedure x times. Therefore, x datasets of same size (same number of data instances), $\{D_1, D_2, ..., D_x\}$, are generated; ($x = 30$ is this study)
2) Apply a particular feature ranking technique to every sampled data, D_i, to obtain the feature subsets, T_i^k, i.e., $\left|T_i^k\right| = k$, where k represents the number of the features retained in each feature subset; ($k = 2, 3, ..., 10$ in this study)
3) Apply the same feature ranking technique to that original dataset to obtain the feature subsets, T_0^k;
4) For a given k value, calculate the single stability index, KI, by the following formula:

$$KI = \frac{1}{x}\sum_{i=1}^{x} I_C\left(T_0^k, T_i^k\right), \tag{3}$$

where I_C is the consistency index defined in Formula (2). This KI is the average of consistency index over x pairs of the feature subsets, each pair selected from the original dataset and one of the x sampled datasets.

Note that in the feature selection process (on steps 2 and 3), a key step is to select a subset of features according to their relevance to the class. In this study, the number of the features retained in each feature subset varies from 2 to 10, that is, $k = \{2, 3, 4, 5, 6, 7, 8, 9, 10\}$. As we have nine datasets, six data sampling approaches, and six feature rankers, we repeat this process (steps 1 through 4) 324 times.

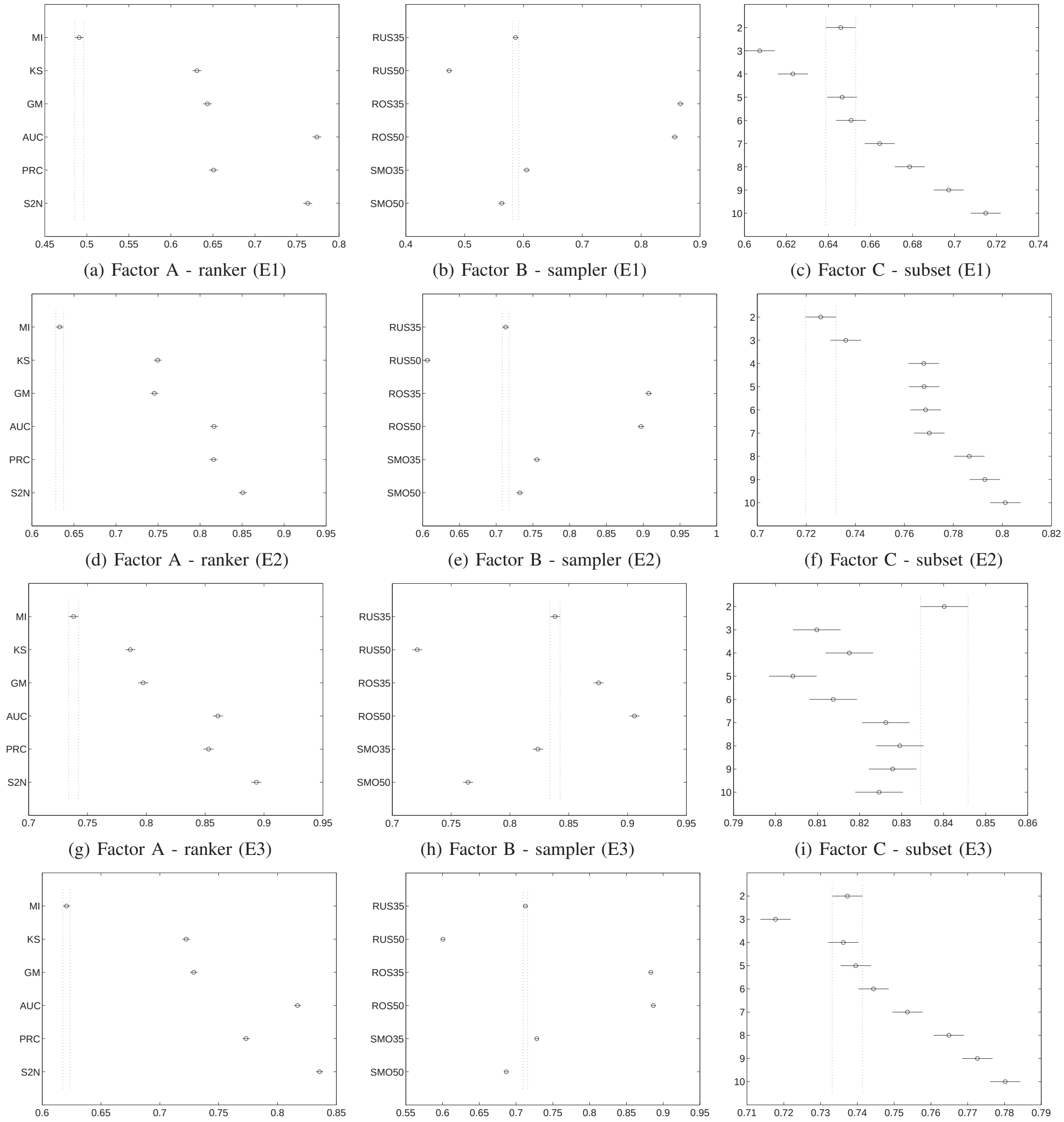
B. *Results and Analysis*

The experiment was performed with the six feature ranking techniques and six data sampling approaches on each of nine Eclipse datasets. Aggregated results on each group of datasets in terms of the stability index (KI) are obtained. Due to space limitations, we only present the results on the first group of datasets as shown in Figure 1. However, the following analysis, findings, and conclusions are summarized over all groups of data. Figure 1 consists of six charts, each displaying for a given sampling approach how the average stability performance (y-axis) of each ranker is affected by the sizes of nine different feature subsets (x-axis), averaged over the three datasets. Some points are observed from the graphics.

- Among the six rankers, S2N and AUC showed higher stability than other rankers for the RUS and ROS sampling techniques, while for the SMO sampling method, S2N and PRC showed higher stability performance. MI always presented relatively low stability compared with others. This is true for all three groups of datasets.
- The size (number of attributes) of a feature subset affected the stability of a feature ranking technique. The trends were not unique and they were dependent on the the specific sampling technique and the given dataset.

In order to examine the significance level of the performance differences, we also performed a three-way ANalysis Of VAriance (ANOVA) F-test on the stability (KI) for each group of the datasets (Eclipse 1, 2 and 3) separately and also for all nine datasets together. The underlying assumptions of ANOVA were tested and validated prior to statistical analysis. The three factors in the test were designed as follows. Factor A represents the six feature ranking techniques, Factor B represents the six sampling approaches, and Factor C represents the number of features retained in the subsets. The null hypothesis is that all the group population means are the same and the alternate hypothesis is that at least one pair of means is different. If the ANOVA results in accepting the alternate hypothesis, a multiple comparison test should be used to detect which group means are different from one another. The ANOVA results showed that all p-values were 0s, meaning the alternative hypothesis was accepted for each group of testing datasets.

We further performed the multiple comparison (MC) test and obtained the results as shown in Figure 2. This figure includes the MC results for each group of datasets (denoted E1, E2, and E3) and all nine datasets together (E-All). Each chart displays a graph with each group mean represented by a symbol (○) and the 95% confidence



(a) Factor A - ranker (E1)

(b) Factor B - sampler (E1)

(c) Factor C - subset (E1)

(d) Factor A - ranker (E2)

(e) Factor B - sampler (E2)

(f) Factor C - subset (E2)

(g) Factor A - ranker (E3)

(h) Factor B - sampler (E3)

(i) Factor C - subset (E3)

(j) Factor A - ranker (E-All)

(k) Factor B - sampler (E-All)

(l) Factor C - subset (E-All)

Fig. 2. Eclipse: Multiple Comparison

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. The multiple comparison outcomes illustrate the following points.

- Among the six rankers, S2N, AUC and PRC showed significantly higher stability than the other three rankers. Furthermore, of the three inferior rankers, KS and GM demonstrated higher stability performance than MI.
- Among the six sampling approaches, ROS35 and ROS50 showed the highest stability behavior, followed by SMO35, RUS35, SMO50, and finally RUS50.
- Between two post-sampling class ratios, 35:65 showed significantly higher stability than 50:50 for the RUS and SMO sampling techniques. The reason may be that the 35:65 target class ratio resulted in fewer changes to the original datasets (instances added/removed) than the 50:50 target class ratio (for RUS and SMO). This difference might be enough to reduce the impact of sampling on the features selected (e.g., increases the stability).
- The stability was increasing along the increment of the size of a subset except at the beginning of the point when $n = 2$ (i.e., number of attributes selected is 2). This trend was observed from Eclipse1 and 2, and all 9 datasets. However, no consistent trend was found for Eclipse3.

Finally, we summarized the stability of FS (averaged over nine various feature subsets) affected by the data sampling techniques for all three groups of datasets (Eclipse1, 2, and 3) as shown in Figure 3. The three curves that represent stability over the three groups of

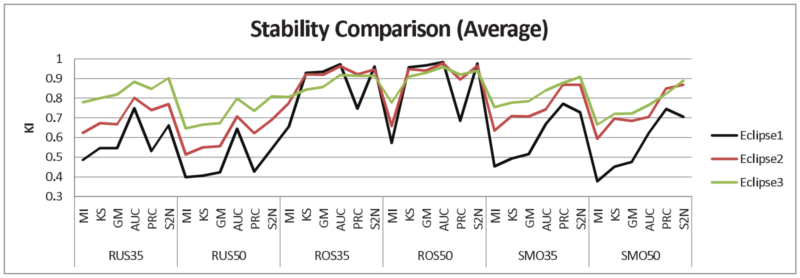


Fig. 3. Stability comparisons over three groups of Eclipse datasets

datasets show the same/similar patterns. It is clearly seen that the FS methods had highest stability performance on the Eclipse3 datasets, then on Eclipse2, and finally on Eclipse1. This is especially true when using the RUS35, RUS50, SMO35, and SMO50 sampling approaches.

VI. Conclusion

This paper presents a strategy that uses feature selection (FS) and data sampling together to cope with the high-dimensionality and class imbalance problems in the context of software defect prediction. Instead of assessing FS techniques by measuring classification performance after the training dataset is modified, this study focuses on another important property of FS – *stability*, more specifically, the sensitivity of a FS method when used with a data sampling technique. More stable FS techniques will reliably give the same features even after sampling has been used, so practitioners can be more confident in those features.

We examined six filter-based feature ranking techniques, five of which are threshold-based feature selection methods (MI, KS, GR, AUC, and PRC), and the remaining one is the signal-to-noise ratio (S2N) method. The three sampling techniques adopted are random undersampling (RUS), random oversampling (ROS), and synthetic minority oversampling (SMO), each combined with two post-sampling class ratios (35:65 and 50:50). The experiments were performed on three groups of datasets from a real-world software project.

The results demonstrate that 1) S2N, AUC, and PRC had higher stability performance than other rankers; 2) ROS35 and ROS50 produced higher stability values than other sampling approaches; (3) post-sampling class ratio between *fp* and *nfp* of 35:65 showed higher stability than the ratio of 50:50 for the RUS and SMO sampling techniques; (4) the stability performance generally increased along the increment of the number of the attributes retained in the feature subset, especially when the dataset is relatively skewed; and (5) the less imbalanced original datasets (prior to sampling) were more likely to obtain higher stability performance when data sampling techniques (such as RUS or SMO) was performed on them.

Future work will include additional case studies with software measurement datasets of other software systems. In addition, different data sampling and FS techniques will be considered in future research.

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] D. J. Dittman, T. M. Khoshgoftaar, R. Wald, and J. V. Hulse, *Handbook of Data Intensive Computing*. Springer, 2011, ch. Feature Selection Algorithms for Mining High Dimensional DNA Microarray Data, pp. 685–710.

[3] H. Liu, H. Motoda, R. Setiono, and Z. Zhao, "Feature selection: An ever evolving frontier in data mining," in *Proceedings of the Fourth International Workshop on Feature Selection in Data Mining*, Hyderabad, India, 2010, pp. 4–13.

[4] K. Gao, T. M. Khoshgoftaar, H. Wang, and N. Seliya, "Choosing software metrics for defect prediction: an investigation on feature selection techniques," *Softw., Pract. Exper.*, vol. 41, no. 5, pp. 579–606, 2011.

[5] S. Shivaji, J. E. W. Jr., R. Akella, and S. Kim, "Reducing feature to improve bug prediction," in *ASE '09 Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering*, Auckland, New Zealand, 2009, pp. 600–604.

[6] 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.

[7] C. Seiffert, T. M. Khoshgoftaar, J. Van Hulse, and A. Napolitano, "Rusboost: A hybrid approach to alleviate class imbalance," *IEEE Transactions on Systems, Man & Cybernetics: Part A: Systems and Humans*, vol. 40, no. 1, January 2010.

[8] 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.

[9] J. L. Lustgarten, V. Gopalakrishnan, and S. Visweswaran, "Measuring stability of feature selection in biomedical datasets," in *AMIA Annu Symp Proc. 2009*, 2009, pp. 406–410.

[10] Y. Saeys, T. Abeel, and Y. Peer, "Robust feature selection using ensemble feature selection techniques," in *ECML PKDD '08: Proceedings of the European conference on Machine Learning and Knowledge Discovery in Databases - Part II*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 313–325.

[11] 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, no. 3, pp. 392–398, February 2010.

[12] S. Alelyani, Z. Zhao, and H. Liu, "A dilemma in assessing stability of feature selection algorithms," in *Proceedings of the 13th International Conference on High Performance Computing and Communications (HPCC)*, Banff, Canada, 2011, pp. 701–707.

[13] X. 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. Knowldege Discovery and Data Mining*. New York, NY: ACM, 2008, pp. 124–132.

[14] L. I. Kuncheva, "A stability index for feature selection," in *Proceedings of the 25th conference on Proceedings of the 25th IASTED International Multi-Conference: artificial intelligence and applications*. Anaheim, CA, USA: ACTA Press, 2007, pp. 390–395.

[15] K. Dunne, P. Cunningham, and F. Azuaje, "Solutions to Instability Problems with Sequential Wrapper-Based Approaches To Feature Selection," Department of Computer Science, Trinity College, Dublin, Ireland, Tech. Rep. TCD-CD-2002-28, 2002.

[16] P. Křížek, J. Kittler, and V. Hlaváč, "Improving stability of feature selection methods," in *Proceedings of the 12th international conference on Computer analysis of images and patterns*, ser. CAIP'07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 929–936.

[17] G. Boetticher, T. Menzies, and T. Ostrand. (2007) Promise repository of empirical software engineering data. [Online]. Available: http://promisedata.org/

[18] 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.

Semantic Interfaces Discovery Server

Laura Maria Chaves
PPGIA - University of Fortaleza
Fortaleza, CE, Brazil
lauramariachaves@gmail.com

José Renato Villela Dantas
SERPRO– Federal Data Processing Service
Fortaleza, CE, Brazil
jose.dantas@serpro.gov.br

Bruno de Azevedo Muniz
PPGIA -University of Fortaleza
Fortaleza, CE, Brazil
brunoamuniz@gmail.com

Júlio Cesar Campos Neto
Instituto Atlântico
Fortaleza, CE, Brazil
julioccn@gmail.com

Pedro Porfírio Muniz Farias
PPGIA-University of Fortaleza; ARCE
Fortaleza, CE, Brazil
porfirio@unifor.br

***Abstract* – This paper presents an architecture for a SERIN (SEmantic RESTful INterface) Discovery Server. SERIN is a semantic interface specification for RESTful Web Services that includes an addressing standard to the Web services. This standard permits the SERIN Discovery Server to use a web crawler to find which hosts implement a specific semantic interface. The presented architecture provides a greater degree of scalability than current known UDDI servers because it is based on semantic interfaces, does not require Web service developers register their services on the server and also presents a semantic interface for software agents to run automated queries on the server.**

Keywords – Semantic Web, RESTful Web service, ontology, semantic interface service discovery, SERIN, semantic interface.

I. INTRODUCTION

Interfaces have been used successfully for interoperability between systems as an abstraction to register a syntactic pattern to be followed by systems that exchange information. Basically interfaces facilitate the division of the multi-layered computer systems, like the division of internal layers in a single object-oriented program, or like division of layers through APIs (Application Programming Interfaces), or distributed processing through IDLs (Interface Description Languages).

Usually, the semantic of each interface is informally communicated through the system documentation, training or personal communication among those involved with the system.

To obtain interoperability in the Semantic Web model, the semantic interface will be defined as an interface whose semantics will be explicitly expressed using ontologies.

In this work SEmantic RESTful INterfaces (SERIN) will be used to specify semantic interfaces. Each host that implements the semantics should implement the methods recommended in the interface.

RESTful Web Services described by semantic interfaces will be called Semantic RESTful Web Services.

Besides presenting a semantic standardization, SERIN also presents a standardization of addressing services, thus it makes it possible for a web crawler to find the hosts that implement a Semantic RESTful Interface.

This paper presents Architecture for a SERIN (SEmantic RESTful INterface) Discovery Server that:

- is designed to be used, mainly, by software agents;
- is designed to use semantic interfaces;
- is based on an automatically constructed index.

The difficulty of finding the requested web service on the Web inevitably remains one of the main motivations of the Semantic Web vision.

Agent creators need to adopt semantic standards in order to allow the exchange of messages between them. Semantic Web Services discovery consists in automating service requests made by a user or machine on the Internet. Several proposals have been presented for Semantic Web Services discovery based on the UDDI (Universal Description Discovery and Integration) server [10]. To discover a Web service in the UDDI server, it is necessary that web services developers explicitly register their services on the UDDI server. This requirement creates a scalability problem.

In addition, UDDI servers, normally, are searched by people that read the descriptions published and select the appropriated services that match with their needs, also, the UDDI servers, originally, are not prepared to work with Semantic Web standards.

UDDI servers typically are used to publish WSDL (Web Services Description Language) [21]. WSDL files encapsulate both abstract interface and connection to a particular web service in a single specification. There are no known cases using a UDDI server to publish only web services abstract interfaces. The use of UDDI server to publish RESTful Web Service Descriptions is also unknown.

Table I. Approaches to Semantic Web Services

	Semantic Web Services		
Web Services	**Annotations in Semantic Service Description**	**Ontologies for Services**	**Annotations in domain Ontologies defining Semantic Interfaces**
SOAP/WSDL based	WSDL-S, SAWSDL	OWL-S, SWSF WSMO	Active OWL
REST based	SA-REST, SBWS(SA-WADL)	EXPRESS, MicroWSMO	SERIN

In order to minimize these problems, this work will propose a RESTful semantic web service discovery server.

The rest of the paper is organized as follows: Section 2 presents proposals for assigning semantics to Web services; Section 3 discusses semantic web services discovery; Section 4 introduces SERIN – SEmantic RESTful INterface; Section 5 shows SERIN Discovery Server architecture; Section 6 presents a Semantic Interface for SERIN Discovery Server; and Section 7 presents our conclusion and future work.

II. SEMANTIC WEB SERVICES

The Semantic Web Service vision [9] is to combine web services with Semantic Web and thereby allow an automatic and dynamic interaction between information systems.

Semantic Web Services intend to use the previously established semantic concepts of an ontology linking them to Web Services components. As shown in Table 1, three strategies are used alternatively to perform this mapping:

- Annotate service descriptions by making them point to ontologies;
- Annotate domain ontologies indicating which Web services exist regarding a given ontology; and
- Create a new specification, independent of the first two. This new specification is usually a meta-ontology that describes the services.

Semantic Web Services can be based on two categories of Web Services: Web Services based on SOAP/WSDL and Web Services based on RESTful. The RESTful web services are stateless and uses only the HTTP methods (GET,PUT,POST,DELETE) to access resources identified by URIs.

A. Semantic Web Services based on SOAP/WSDL

Among the approaches adopted for SOAP/WSDL Semantic Web Services, there is SAWSDL (Semantic Annotation for WSDL) [14]. This approach is based on WSDL-S (Web Service Semantics) [20]. It follows the strategy to create a semantic annotation for service description. To do so, it defines three attributes to WSDL 2.0 extensibility elements: modelReference, liftingSchemaMapping and loweringSchemaMappping that allow semantic annotations on WSDL (Web Services Description Language) components [21].

Examples of ontologies on services, presented in Table I, are the following proposals: OWL-S, SWSF and WSMO.

OWL-S [11] defines an OWL (Ontology Web Language) ontology to describe service properties and capabilities of web services. The OWL-S ontology defines the service concept and three subontologies, known as service profile, which describes, “what the service does”, service model, which tells the customer how to use the service, and service grounding, which specifies the details of how an agent can access a service.

Semantic Web Services Framework, SWSF [15], has its roots in the OWL-S standard. It is composed of an ontology, which provides a conceptual model to describe Web services called Semantic Web Services Ontology (SWSO) [17], and a language for axiomatization called Semantic Web Service Language (SWSL) [16]. This language is more powerful than OWL.

Web Services Modeling Ontology (WSMO) “aims to describe all relevant aspects related to services that are accessible through a Web service interface. Its main objective is to enable task automation (total or partial), e.g. discovery, selection, composition, mediation, execution, and monitoring” [19].

The Third strategy presented in Table I is an annotation indicating which ontology concepts are related to a web service. Active OWL [4] [8] proposes to use ontologies as interfaces. It suggests the restriction to allow only web services (GET, PUT, LIST, DELETE, and POST). Thus, although it was originally proposed for SOAP/WSDL Semantic Web Services (SWS), OWL Active approaches a SWS-REST specification.

Active OWL uses the strategy of associating Web services with ontologies by annotating classes and properties. These annotations indicate the existence of Web services types. Thus, the annotations specify a semantic interface to the five Web services types. Annotations are defined in OWL using Annotation Properties.

B. Semantic Web Services based on REST

In Table I, SA-REST and SBWS are proposals for semantic Web services strategy to semantically annotate services descriptions.

The SA-REST [7] proposal was created for adding semantics to RESTful services, borrowing the SAWSDL idea. The fundamental difference between SAWSDL and SA-REST is that SAWSDL semantic annotations are added in the WSDL (which are not normally adopted for RESTful Web Services). The SA-REST annotations will be added to Web pages, describing the services through the use of micro formats such as RDFa [13] and Gleaning Resource Descriptions from Dialects of Languages [6].

In this proposal, when a system performs a search with predefined criteria for a resource, it cannot know the page address where the searched information semantics is. Thus, the provider would have to somehow provide the page address where the semantics reside so the applicant can access it. Otherwise, the applicant should have a scanner to read every provider page in order to find the predicates above.

SBWS (Semantic Bridge for Web Services) is also proposed by [3] to associate semantics to RESTful Web services. SBWS makes annotation on documents WADL, like the SAWSDL, connecting REST input and output methods to ontologies.

EXPRESS (RESTful Expressing Semantic Services) proposal [2] describes RESTful Web Services using domain ontologies that include descriptions of the features (following a format OWL/EXPRESS). From these descriptions, it generates stubs for access to available services.

This process starts when a service provider describes the entities and their relationships as ontology (as an OWL file). Using the OWL file, a deployment mechanism EXPRESS (EXPRESS Deployment Machine) generates a list of URIs that identify resources, related classes, instances and properties. The service provider then specifies which HTTP methods can be applied to these features (though there is information on how this specification is performed).

The authors state that the use of HTTP methods (GET, PUT, DELETE, POST, and OPTIONS) is sufficient to define consistent operational semantics about all the features of web services.

MicroWSMO [22] consists of service ontology for RESTful Web services, and a method for annotating descriptions of RESTful services.

SERIN is an Active OWL adaptation for Semantic Web Services and REST. It fits the strategy of specifying semantic interface via annotations on ontologies.

The semantic interface is represented by OWL ontologies with annotations that indicate how the five types of web services (GET, PUT, LIST, DELETE, POST) are mapped in the four HTTP methods (GET, PUT, DELETE, POST).

This approach proposes a convention for URLs that address the RESTful semantic Web services dividing them into two components: the host and the semantic interface. Each host that implements a semantic interface is responsible for implementing RESTful Web Services associated with it. This convention permits discover which host implements a interface, so is possible to construct the SERIN discovery server proposed in this paper.

III. SEMANTIC WEB SERVICE DISCOVERY

The semantic web services discovery is the process of finding technical means to access service providers with relevant capabilities, based on semantic annotation. This finding has to be fully automated, since there are no people involved in the process that could help agents in the trial to determine if the service is relevant or not. For this purpose, all parts involved in the semantic annotation has to be understandable by machines, so the agent can put them together neatly for correct conclusions. To achieve this goal, semantic web service discovery explores expressive formalization of knowledge representation and sophisticated techniques of automated reasoning [18].

Among the first, and possibly most influential works on semantic Web services discovery, [12] was limited to semantically annotating web services inputs and outputs. In this approach, "a publishing matches a request when all publishing inputs and outputs match with request inputs and outputs."

Paulocci´s criterion only allows an approximate match. The "Degree of Match" (DoM) concept can be introduced as an attempt to deal with these problems [5]. DoM can be informally defined as a value from an ordered set of values that express how similar two entities are, considering some similarity metric. Such entities may be services, IOPE attributes (Input, Output, Preconditions, Effects), or a specific service operation. A matching algorithm service that calculates DoM can thus be used to classify discovered services according to their relevance to the request.

Semantic pattern matching algorithms are usually complex. They operate by considering that requests from publications and services are not in perfect combination. They are designed to explore semantic functionality explicit description and service request. Algorithms have been offered with different features, suggesting different architecture types, for example peer-to-peer.

Reference [5] considers that the use of these domain ontologies by both service provider and service requestor, though not necessary, significantly simplifies the process of semantic pattern matching by eliminating the need for mediation layers. However, this is an optimistic scenario, not suitable for open information environments, such as the Web.

Therefore, if the same domain ontology is not used for both provider and requestor, considering the approximate character of the matching, a rough mediation layer should be introduced.

The complexity of these approaches is a result of its heavy reliance on reasoning for pattern discovery and matching. This complexity also means that the approaches would not work properly on web-scale, which is a key requirement for Web Services [1].

In the SERIN Discovery Server, services published and requested follow the same interface, i.e. the matching is always accurate in order to avoid the need for complex pattern matching. This proposal allows a scalable solution.

IV. SERIN – SEMANTIC RESTFUL INTERFACE

The REST standard states that resource orientation and the use of HTTP methods are the only operations available.

Semantic RESTful Interfaces (SERIN) is basically annotated ontology whose classes and properties characterize uniquely and semantically available resources. Annotations indicate which RESTful Web Services should be implemented for each resource.

SERIN is based on an earlier specification called Active OWL [8] that applies to Web Services SOAP/WSDL.

The strategy used in SERIN to associate web services with ontologies was to annotate classes and properties of an ontology written in OWL through five notes, indicating the existence of five types of Web services: “get”, “post”, “delete”, “put”, and “list”.

The five types of web services are mapped in the four HTTP methods. The types “get” and “list” will both be mapped to the GET method. The “get” type requires a key as a parameter indicating a single instance to be recovered, while the “list” type has no parameters and returns a list of all class instances. This restriction makes it suitable for use with RESTful Web Services that, normaly, are also based on the HTTP methods.

```
<owl:AnnotationProperty
rdf:about="http://www.unifor.br/
       serinAnnotations.owl#delete"/>
<owl:AnnotationProperty
rdf:about="http://www.unifor.br/
       serinAnnotations.owl#post"/>
<owl:AnnotationProperty
rdf:about="http://www.unifor.br/
       serinAnnotations.owl#get"/>
<owl:AnnotationProperty
rdf:about="http://www.unifor.br/
       serinAnnotations.owl#put"/>
<owl:AnnotationProperty
rdf:about="http://www.unifor.br/
       serinAnnotations.owl#list"/>
```

Listing I. Excerpt from SERIN Annotations Ontology.

In OWL, annotations are defined through Annotation Properties. Listing I shows the definitions of annotations used. To avoid repetition of annotations in an interface, we define the SERINAnnotations ontology to package the required settings. SERINAnnotations ontology can be imported to each RESTful semantic interface.

A. *SERIN annotations for classes and properties*

The annotations syntax is specified by an ontology of SERINAnnotations and through it, each of the five annotations received a URI that uniquely, identifies it. To be formally used, the meaning, i.e., the semantics of each entry must first be informally defined and everyone involved must know this definition.

The option to use only five web service types makes SERIN semantics much simpler than the proposals based on SOAP/WSDL Web services, which apply to any Web service. Since the semantics of resources are properly established by the semantic interface, it remains only to clarify the semantics of annotations used.

Each annotation indicates the existence of a RESTful Web service URL whose semantics are presented in Table II and Table III.

In both Table II and Table III, the identifier <host> represents the host URL that implements the ontology. <Ont> identifier is the URI that identifies the ontology. The identifiers <class> and <prop>, respectively, represent the annotated class and annotated property.

Note that “list” and “get”, two notes are associated with the HTTP GET method. This option was adopted because the HTTP GET has different results, depending on whether or not parameters are used. When used without parameters the GET method returns a list of individuals and when used with an identifier as a parameter the GET method returns only the single individual identified by the parameter reported.

The media-type “application/owl+xml” is used to represent arguments and results.

Annotations cannot be applied to anonymous classes or classes built from expressions. Annotations are not inherited, that is, a child class must specify its own Web services, which will not be inherited from the parent class.

On the current specification, execution errors on Web services should be treated by the application. An expected extension for the current specification would be the treatment of such errors.

Table II Annotations on classes indicating existence of a Web Service, its location (URL) and its Semantics.

Get	URL	GET <host>/serin/<Ont>/<Class>/<rdf:id>
	Semantics	Receives a <rdf:id> identifier as a parameter and returns the its individual
List	URL	GET <host>/serin/<Ont>/<Class
	Semantics	List the individuals in a Class
Post	URL	POST <host>/serin/<Ont>/<Class>/<individual>
	Semantics	Receives as parameter the XML corresponding to the individual and it adds to the class individuals set
Put	URL	PUT <host>/serin/<Ont>/<Class>/<rdf:id>/<individual>
	Semantics	Receives, as a parameter, an OWL corresponding to the individual identified by <rdf:id> and updates the individual information
Delete	URL	DELETE <host>/serin/<Ont>/<Class>/<rdf:id>
	Semantics	Receives, as parameters, an individual identifier and removes this individual from the class individuals set

Table III. Property notes indicating the existence of Web services, its location (URL) and Semantics.

Get	URL	GET <host>/serin/<Ont>/<Class>/<rdf:id>/<prop>
	Semantics	Receives as parameters the individual identifier <rdf:id> and returns the property value for the respective individual <prop>. The annotation "**get**" is only defined for functional properties
List	URL	GET <host>/serin/<Ont>/<Class>/<prop>
	Semantics	List the values associated with property <prop>
Post	URL	POST <host>/serin/<Ont>/<Class>/<rdf:id>/<prop>/<value>
	Semantics	Assigns a new value <value> to property <prop>
Put	URL	PUT <host>/serin/<Ont>/<Class>/<rdf:id>/<prop>/<value>
	Semantics	Updates the property <prop> value <value>
Delete	URL	DELETE <host>/serin/<Ont>/<Class>/<rdf:id>/<prop>/<value>
	Semantics	Delete the value <value> associated with property <prop>

B. *Semantic Web Service addressing*

Figure 1 shows an example in which a personal agent performs a query on a group of car rental agents. The car rental agents implement their web services in each of the hosts of the rental agencies, which are www.rentalA.com, www.rentalB.com and www.rentalC.com. The three hosts implement the same semantic interface, called serin/vehicle. Thus, running the Web service www.rentalA.com /serin/Vehicle.owl/availableModel produces vehicle models available in rentalA, while the other two Web services should, respectively, provide models of vehicles available in rentalB and rentalC.

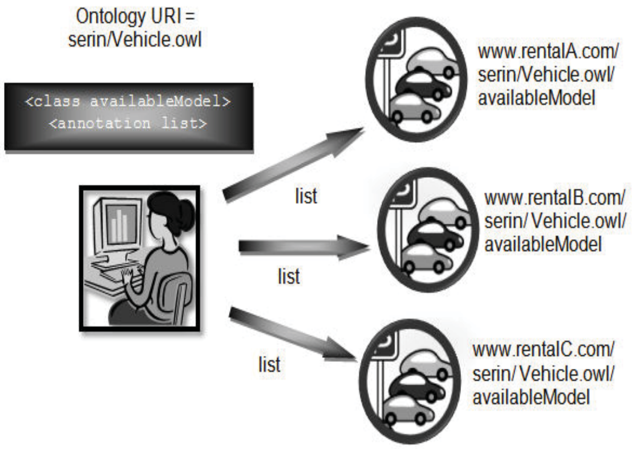


Figure 1. Using Semantic Interfaces RESTful.

Thus, each car rental agent has a RESTful Web service class corresponding to available Model. The personal agent may trigger the GET Web service in each of the implementations obtained in each case, from the list of models available for that store.

The convention used is particularly simple when applied to RESTful Web Services, however, you can apply it to traditional Web services. In the case of architecture SOAP / WSDL, the address constituted under the proposed standard convention refers to a URL of a WSDL file that describes the service [12].

Semantic interfaces are similar to interfaces in OOP (Object Oriented Programming). Each agent who accepts a semantic interface is responsible for implementing its methods.

For each agent, the address (URL) RESTful semantic Web service consists of a variable and a fixed part of the grammar as shown in Table II. The variable part corresponds to the URL of each agent <host > URL while the fixed part, which remains the same for all agents that implement the same Semantic Interface, is derived from the URI of the ontology <Ont uri> and the class name <Class >.

```
<host URL> = "www.rentalAgencyC.com"
<ontology URI> = "serin/Vehicle.owl"
<class name> = "available Model"
<interface URI> = "/ontologies/serin/Vehicle.owl/
          availableModel"
<service URL> = "www.rentalAgencyC.com/ontologies
          /serin/Vehicle.owl/availableModel"
```

Listing II. URL sample of a RESTful Web Service that implements a Semantic Interface

Listing II is presented, as would be the URL of the Web service implemented for the agent rentalC in Figure 1.

V. SERIN DISCOVERY SERVER ARCHITECTURE

The SERIN Discovery Server is a centralized index, which works on architecture composed of four components, as shown in Figure 2:

- A scanning module – a crawler that scans the Web content, updating;
- A repository of indexes (that maps Semantic interfaces to hosts);
- A manual query module - a site where users query the index;
- An automatic query module - semantic Web services which automatic agents query the index.

The scanning module inspects each web host known to identify semantic interfaces that each host implements. The scanning module must also update the list of known hosts and the list of known semantic interfaces. Information obtained by the scanning module, alternatively, can be manually published by those responsible for a host by developers of a semantic interface who wish their site or their interface becomes part of the index.

The manual query module should basically allow a user to query the hosts that implements a given interface. In addition, manual module queries could offer multiple queries of interest to designers and users wishing to build new interfaces or specify agents. For example, which interfaces are registered, and which interfaces are implemented by each host.

The automated query module, mainly, responds to the software agents which hosts implement certain interface. The Software agents that will consult the automated query module must use a semantic interface that will be presented in section VI.

The repository stores the index of hosts and the index of semantic interfaces(SI) as well as the known relationships among them.

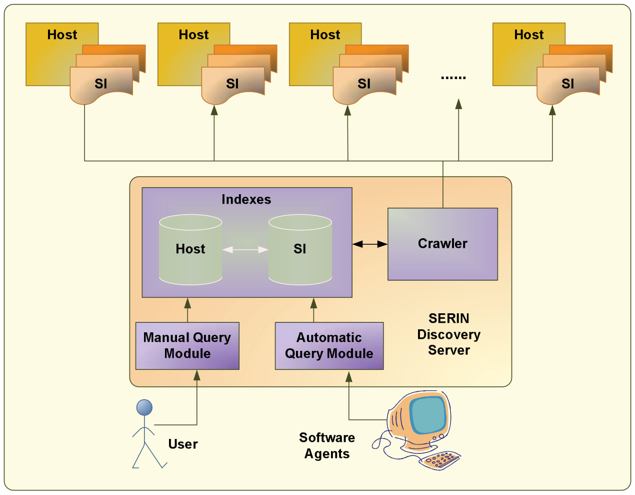


Figure 2. SERIN Discovery Server Architecture.

VI. SERIN DISCOVERY SERVER SEMANTIC INTERFACE

Consultations at the SERIN Discovery Server will be done by implementing the semantic interface called SerinSearch shown in Figure 3. SerinSearch interface has two classes: the class Host and the class Ontology that are connected by two properties implementsOntology and isImplementedBy (inverse property).

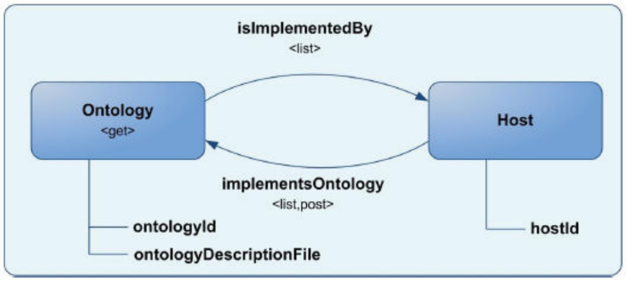


Figure 3. SERIN Search Semantic interface for consultation at the SERIN Discovery Service

These two classes design follow the Ontological Minimum requirements principle. An Ontology class has only two attributes (as data type properties) ontologyId and ontologyDescriptionFile. While the class Host has only one data type property called hostId.

The ontology has appropriate annotations with the Web services that the SERIN Discovery Server must implement. Table IV presents the respective URL and semantics for each of the four methods.

Table IV. SERIN Discovery Server Web Service description.

isImplementedBy <list>	**Method**	GET
	URL	`<host>/serin/SerinSearch/Ontology/<OntId>/isImplementedBy`
	Semantics	Set of hosts that implement a specific semantic interface .
implementsOntology <list,post>	**Method**	GET
	URL	`<host>/serin/SerinSearch/Ontology/<HostId>/implementsOntology`
	Semantics	Returns the interfaces that a host implements.
	Method	POST
	URL	`<host>/serin/SerinSearch/Ontology/<HostId>/implementsOntology/<Ont id>`
	Semantics	Returns in which the host implements a specific interface. The discovery server confirms this information by host direct inspection.
Ontology <get>	**Method**	GET
	URL	`<host>/serin/SerinSearch/Ontology/<Ont Id>`
	Semantics	Returns an ontology class instance including its description file.

A. *SERIN Discovery Server automatic search*

The sequence diagram in Figure 4 describes an automated search at the SERIN Discovery Server and a following query to hosts that implement an interface called semantic Ont1.

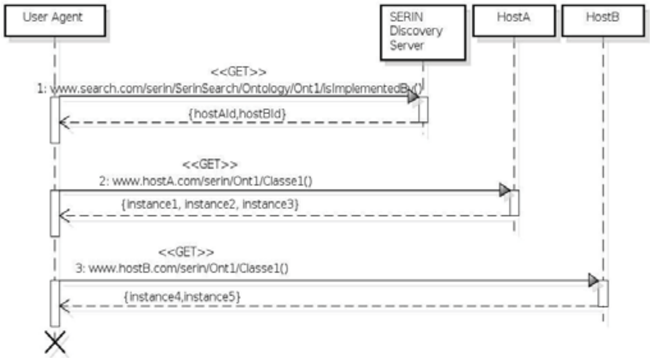

User Agent
SERIN Discovery Server
HostA
HostB
<<GET>>
1: www.search.com/serin/SerinSearch/Ontology/Ont1/isImplementedBy()
{hostAId,hostBId}
<<GET>>
2: www.hostA.com/serin/Ont1/Classe1()
{instance1, instance2, instance3}
<<GET>>
3: www.hostB.com/serin/Ont1/Classe1()
{instance4,instance5}

Figure 4. SERIN Discovery Server automatic query sequence diagram.

The user agent wants to find hosts that implement the semantic interface Ont1. It sends a query as a GET URL www.busca.com/serin/SerinSearch/Ontology/Ont1/isImplementedBy to the SERIN Discovery Server. The server returns the list {hostAId, hostBId}, which contains the identifiers of the two hosts that implement Ont1 ontology.

First, the user agent will consult HostA. The agent sends a GET request through the URI www.hostA.com/serin/Ont1/Classe1. HostA receives the GET request, which contains the resource related to Class1. It performs a query in its database. Finally it sends back a list with all individuals in Class1.

After, sending a request to HostB performs the same query.

B. Index updating at SERIN Discovery Server

The scanning module can make its scan by direct inspection or by asking the host about which semantic interfaces it implements.

The sequence diagram in Figure 5 shows an example of scanning the hosts and updating of indexes in the SERIN Discovery Server. This service intends to search for semantic interfaces that exist in these hosts. With this, SERIN server will update the index database repository. This database will have a list of found semantic interfaces, as well as the relationship of semantic interfaces with each host.

The discovery server performs the following steps at the hostA.

The discovery server sends a GET request, represented by the URL www.hostA.com/serin/ to determine which semantic interfaces that HostA has. The HostA response indicates that it implements the semantic interface Ont1.

Discovery server searches the semantic interface Ont1 at the index repository using the method hasOntology (Ont1Id). The server receives a negative answer about the existence of interface Ont1 in the repository.

The server asks the discovery HostA about the Ont1 interface description. It receives a description file as an answer. Then, it includes the semantic interface Ont1 in the index repository. Finally, the method insertIsImplementedBy updates the index repository indicating, through their respective parameters, that hostAId implements the semantic interface Ont1Id.

For HostB, steps six to eleven are executed. In step six, it identifies that HostB implements two semantic interfaces: Ont1 (which already exists in the index repository) and Ont2 (to be included in the index repository).

The discovery server sends a GET request, represented by the URL www.hostA.com/serin/ to determine which semantic interfaces that HostA has. The HostA response indicates that it implements the semantic interface Ont1.

Discovery server searches the semantic interface Ont1 at the index repository using the method hasOntology (Ont1Id). The server receives a negative answer about the existence of interface Ont1 in the repository.

The server asks the discovery HostA about the Ont1 interface description. It receives a description file as an answer. Then, it includes the semantic interface Ont1 in the index repository. Finally, the method insertIsImplementedBy updates the index repository indicating, through their respective parameters, that hostAId implements the semantic interface Ont1Id.

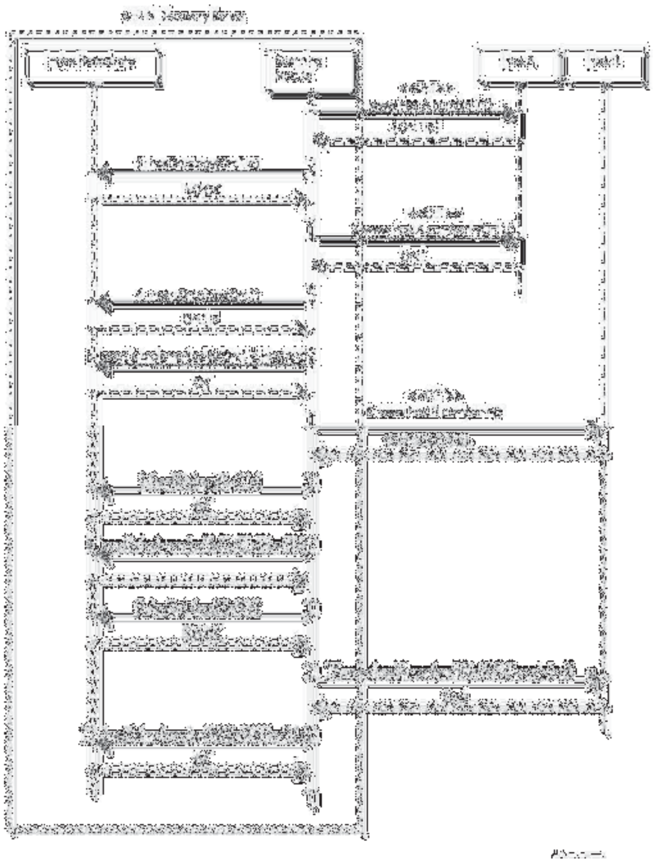


Figure 5. Semantic interface discovery at various hosts sequence diagram

For HostB, steps six to eleven are executed. In step six, it identifies that HostB implements two semantic interfaces: Ont1 (which already exists in the index repository) and Ont2 (to be included in the index repository).

VII. CONCLUSION

The first SERIN (Semantic RESTful Interface) contribution is to utilize semantic interface concepts on the context of Web services.

The interface concept is traditionally used to provide interoperability at various levels of computing, whether in-house programs, at API level, or in distributed computing models. Semantic Interfaces fit the concept of interface technologies related to the Semantic Web and, in particular, RESTful Web Services.

Following the REST approach, SERIN uses only five service types that follow HTTP methods semantics.

In addition to semantic standardization, a second SERIN contribution is to standardize service URLs.

This work presented how address standardization adopted in SERIN may be used to discover semantic Web services. Such discovery can be performed in three ways, namely, by direct inspection, by consultation at a centralized index, and by broadcasting a P2P network.

In particular, architecture for a SERIN Discovery Server was designed. This work presented its functionalities. It also indicates a semantic interface to RESTful web services through which the server can be accessed by programs.

In order to accelerate the discovery, SERIN architecture standardized two Web Services to be implemented by each host that implements a semantic interface. Through the first service, the host responds to the interface it implements. The second service answers a file that describes each interface.

The main distinction this server presents is the fact that is not necessary for suppliers to register their services (like UDDI server does). In contrast, the index server is powered primarily by a crawler that scours the Web to identify hosts that implement each semantic interface.

This innovative feature, which arises from the use semantic interfaces concept, has a great impact on the scalability of the index server. In this respect, the difference between an UDDI server and a SERIN Discovery Server is similar to the difference between a server based on automatically constructed indexs, such as Google, and the former service offered by Yahoo, which depended on the initiative of the developers to register each Web page on the server.

Besides a SERIN Discovery Server is designed to use semantic interfaces, while a UDDI server usually publishes WSDL files having interface specification and implementation in the same package.

Finally, a SERIN Discovery Server is designed to be used by software agents, while the UDDI by people who, after consulting the specifications would be able to build programs to access the services described.

In future work, we will emphasize mainly the actual construction of the proposed server, whose success depends on the spread of semantic interface SERIN. In addition, we also propose the study of mechanisms for ontologies and interfaces evolution.

The annotations in SERIN can be extended to specify non-functional aspects of web services, for example, quality of service and security.

REFERENCES

[1] Alowisheq, A. e Millard, D.E. 2009. EXPRESS: EXPressing RESTful Semantic Services. Web Intelligence and Intelligent Agent Technology, IEEE/WIC/ACM International Conference on. 3, (2009), 453-456.

[2] Alowisheq, A. et al. 2009. EXPRESS: EXPressing RESTful Semantic Services Using Domain Ontologies. ISWC ’09: Proceedings of the 8th International Semantic Web Conference (Berlin, Heidelberg, 2009), 941–948.

[3] Battle, R. e Benson, E. 2008. Bridging the semantic Web and Web 2.0 with Representational State Transfer (REST). Web Semantics: Science, Services and Agents on the World Wide Web. 6, 1 (2008), 61 - 69.

[4] Campos Neto, J.C. 2007. Active OWL - Uma arquitetura para integrar serviços Web e ontologias. Universidade de Fortaleza (UNIFOR).

[5] Cardoso, J. 2007. Semantic Web services : theory, tools and applications. Information Science Reference.

[6] Gleaning Resource Descriptions from Dialects of Languages (GRDDL): 2007. http://www.w3.org/TR/grddl/. Accessed: 2011-06-10.

[7] Lathem, J.D. et al. 2007. SA-REST: BRING THE POWER OF SEMANTICS TO REST-BASED WEB SERVICES Electronic Version Approved:

[8] Marques, T.C. et al. 2008. Active Ontologies - an Approach for Using Ontologies as Semantic Web Services Interfaces. SEKE’08 (2008), 847-852.

[9] McIlraith, S.A. e Martin, D.L. 2003. Bringing Semantics to Web Services. IEEE Intelligent Systems. 18, (2003), 90-93.

[10] OASIS UDDI Specification TC | OASIS: 2005. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=uddi-spec. Accessed: 2011-07-13.

[11] OWL-S’ Relationship to Selected Other Technologies: 2004. http://www.w3.org/Submission/2004/SUBM-OWL-S-related-20041122/. Accessed: 2010-05-06.

[12] Paolucci, M. et al. 2002. Semantic Matching of Web Services Capabilities. The Semantic Web — ISWC 2002. I. Horrocks e J. Hendler, orgs. Springer Berlin / Heidelberg. 333-347.

[13] RDFa Primer: 2008. http://www.w3.org/TR/xhtml-rdfa-primer/. Accessed: 2011-06-10.

[14] Semantic Annotations for WSDL and XML Schema — Usage Guide: 2007. http://www.w3.org/TR/sawsdl-guide/. Accessed: 2011-06-07.

[15] Semantic Web Services Framework (SWSF) Overview: 2005. http://www.w3.org/Submission/SWSF/. Accessed: 2010-05-06.

[16] Semantic Web Services Language (SWSL): 2005. http://www.w3.org/Submission/SWSF-SWSL/. Accessed: 2011-06-13.

[17] Semantic Web Services Ontology (SWSO): 2005. http://www.w3.org/Submission/SWSF-SWSO/. Accessed: 2011-06-13.

[18] Studer, R. 2007. Semantic web services : concepts, technologies, and applications. Springer.

[19] Web Service Modeling Ontology (WSMO): 2005. http://www.w3.org/Submission/WSMO/. Accessed: 2010-05-08.

[20] Web Service Semantics - WSDL-S: 2005. http://www.w3.org/Submission/WSDL-S/. Accessed: 2010-05-06.

[21] Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language: 2007. http://www.w3.org/TR/2007/REC-wsdl20-20070626/. Accessed: 2010-04-29.

[22] MicroWSMO: 2008. http://www.wsmo.org/TR/d38/v0.1/. Accessed: 2011-12-09.

Cloud Application Resource Mapping and Scaling Based on Monitoring of QoS Constraints

Xabriel J. Collazo-Mojica
S. Masoud Sadjadi
School of Computing and Information Sciences
Florida International University
Miami, FL, USA
{xcoll001, sadjadi}@cs.fiu.edu

Jorge Ejarque
Rosa M. Badia
Grid Computing and Clusters Group
Barcelona Supercomputing Center
Barcelona, Spain
{jorge.ejarque, rosa.m.badia}@bsc.es

***Abstract*—Infrastructure as a Service (IaaS) clouds promise unlimited raw computing resources on-demand. However, the performance and granularity of these resources can vary widely between providers. Cloud computing users, such as Web developers, can benefit from a service which automatically maps performance non-functional requirements to these resources. We propose a SOA API, in which users provide a cloud application model and get back possible resource allocations in an IaaS provider. The solution emphasizes the assurance of quality of service (QoS) metrics embedded in the application model. An initial mapping is done based on heuristics, and then the application performance is monitored to provide scaling suggestions. Underneath the API, the solution is designed to accept different resource usage prediction models and can map QoS constraints to resources from various IaaS providers. To validate our approach, we report on a regression-based prediction model that produces mappings for a CPU-bound cloud application running on Amazon EC2 resources with an average relative error of 17.49%.**

***Index Terms*—cloud computing; QoS; resource allocation.**

I. Introduction

Cloud computing presents the illusion of infinite capacity of computational resources. In the case of Infrastructure as a Service (IaaS) clouds, these resources are typically offered in bundles with specific amounts of CPU, Memory, and Network. Solution developers are thus presented with the problem of ensuring the performance non-functional requirements of an application by mapping it to one or more of these bundles, and to monitor and change this mapping if the workload changes. We present our work on an autonomic service which monitors Quality of Service (QoS) metrics of cloud applications and suggests bundle mappings which would ensure the required performance.

The problem of resource allocation in the cloud has been studied before, and various techniques to solve it has been proposed. Ganapathi et at. [1] utilize statistical machine learning to predict resource usage of an application in the cloud. Islam et al. [2] estimate CPU resource usage by simulating a cloud provider. Previous solutions have monitored low-level resources for their prediction, i.e., they would monitor CPU usage. Our solution targets QoS assurance by learning if the currently allocated resources are delivering the required QoS constraints. That is, we monitor if the application's performance non-functional requirements are being met (e.g. the response time), and based on this knowledge, we then adjust the resource allocation.

The key idea of our solution is the use of a Service Oriented Architecture (SOA) approach in which users provide a descriptive model of their application and get back mappings in various IaaS providers. These mappings emphasize the assurance of the Quality of Service metrics from the applications model. An initial mapping is done based on heuristics, and then we monitor the application's performance to provide scaling suggestions via a callback interface. Underneath this API, the solution accepts different resource usage prediction models and allows allocation in different IaaS providers.

The main technical challenges of this work were the experimentation with alternative regression techniques, and the implementation of a resource usage prediction engine. In this work, we propose the use of a linear regression model that provides adequate performance as well as good accuracy. For the implementation of the engine, we developed a rule-based system that calculates which IaaS resource bundles can ensure the applications QoS constraints. The solution then recommends the bundles with the best fit for various IaaS providers.

To validate our approach, we present three experiments. In the first experiment, we test for correct behavior with a simulation test that sends an application model and monitoring data to the prototype system. In our second experiment, we run the same model against a real IaaS environment. We gather the performance data and then scale the application according to the solution's suggestions. Finally, we also present a time and scalability analysis of the solution. The results show that the prototype correctly ensures QoS constraints of a CPU-bound cloud application.

The main contributions of this paper follow. First, we present the design of an autonomic solution for cloud application resource mapping and scaling based on monitoring of QoS constraints. Second, we provide details on a prototype implementation and how we dealt with the technical challenges. Finally, we asses the validity of the idea by presenting experiments on functionality and scalability.

II. Background and Methodology

A. Background

1) Previous Work: This work is part of our overarching "Distributed Ensemble of Virtual Appliances" (DEVA) project. In [3], we discussed the advantage of simplifying the solution development workflow by presenting the developer with a model-based visual designer to draft the software dependencies and resource requirements of their cloud application.

In [4], we formally defined our DEVA model approach. This model is based on the notion of Virtual Appliances, defined by Sapuntzakis et al. [5], that represent self-configurable software applications and OS as updatable image files. These appliances can be instantiated on top of IaaS providers such as Amazon EC2 [6]. DEVAs represent groups of Virtual Appliances with QoS constraints between each appliance, and general policies for the whole ensemble.

2) Problem Definition: Given that we have a model of an application's architecture and its desired QoS constraints, we can think of a model-to-instance transformation to different IaaS providers. For this transformation to work effectively, we also need a model representation of the performance of the application on top of resources from any given provider. That is, given a DEVA model, known IaaS providers and its bundles, and potentially known workloads, we want to transform the model (i.e. map the model) to instances that would ensure the QoS constraints specified in the model.

B. Methodology

1) Approach: To do this QoS-to-Resources mapping, we propose the steps illustrated in Figure 1. First, a user designs the DEVA model in the DEVA-Designer. When ready to deploy, the user chooses an IaaS provider. This request is (1) sent to a Transformation Engine, which has no previous knowledge of the submitted model, and thus delegates the creation of a preliminary mapping to an API Mapper (2). The Engine then makes the proper API calls to the specific IaaS provider (3). The IaaS provider (4, 5) instantiates the model. Note that the IaaS provider does not know about DEVAs, and only processes its own API calls. Note further that in this work we assume that the software provisioning is already done, i.e., that virtual appliances are available in the provider's image repository. Monitoring is done on each appliance to gather QoS data (6, 7). This data is eventually (8) fed back to the Transformation Engine. The Engine then (9) decides whether the currently assigned resources are appropriate for the QoS specifications in the model. If it is the case that a change of resource allocation is needed, the engine delegates the construction of a change request (10), and then sends that change to the IaaS provider (11). Finally, the provider makes the necessary changes to the instance to better comply with the model's QoS constraints (12).

2) Limitations: Our approach depends on previous monitoring data to make good resource mappings. To have good sample points, we propose the following. As we target cloud applications, the cloud developer (i.e. the user of our solution) first deploys their application in a "staging" mode. In this mode, the cloud developer runs performance tests on his application using different IaaS bundles. These performance tests generate data points that are sent to the proposed Monitoring API, and therefore improve the accuracy of our solution. As typical IaaS instances are billed by the hour, the cost of launching an application on different bundles for performance testing is negligible compared to the eventual QoS improvement. Although we are working in automatizing this learning phase, we do not report it as part of this paper.

III. Design and Implementation

In this section, we report on the main objectives driving the design of our solution. We also include implementation details on how we dealt with the technical challenges.

A. Design

We have designed the Transformation Engine as a service. In designing this service, we had two main Design Objectives:

1) to have a simple interface that can be used by our research team as well as others, and
2) to create a service that supports various resource mapping techniques for experimentation.

Although this work is integrated with the DEVA-portal project, it can also be used as a stand alone solution. This is why we chose to interact in terms of Resource Description Framework (RDF) [7] tuples. That is, the API expects and returns DEVA models wrapped in this interoperable model representation framework. Other research teams can use our mapping service by either using our DEVA modeling approach, or by first parsing their application representation to a DEVA using techniques such as OWL's Web Ontology Language [8]. This allows us to comply with Design Objective 1.

Table I presents the proposed REST API. We chose the REST technology because of Design Objective 1. First, a POST is done as explained in Section II-B1. This request has to include the DEVA model to be transformed. The API processes the model and responds with a link to the newly created resource. A subsequent GET to that link will return the transformed DEVA with the newly generated mappings. An API call to delete a mapping is also provided.

Monitoring data is expected as POST requests that include data points. The data points collected include the amount of the metric being monitored. A callback API subscribes with a POST or unsubscribes with a DELETE consumer that want to monitor model changes. Model mapping changes trigger these callbacks. Note that this design does not hint at any specific resource allocation technique, and thus allows us to implement a solution that complies with Design Objective 2.

B. Implementation

Underneath the API, the transformation engine is implemented as a simple resource usage prediction framework. For an initial mapping, the solution follows the steps presented in Figure 2. First, the DEVA model is received by the API. Then,

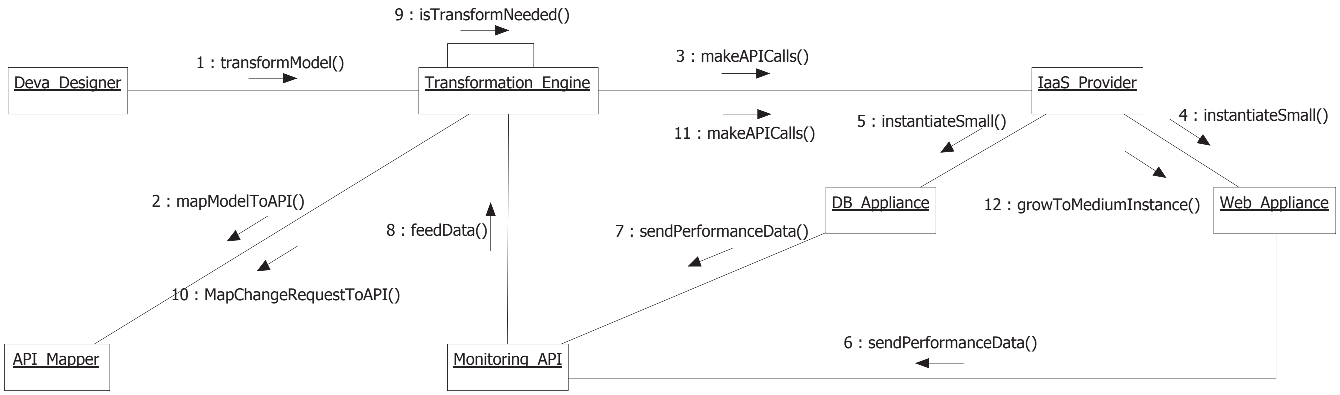


Fig. 1. UML Collaboration Diagram modeling the interaction of the solution with an IaaS provider.

REST HTTP API	Description
POST /deva/mappings	**Expects**: a DEVA model in RDF format in the body. **Optional**: a :callback_URL parameter if a callback_URL is provided, then it will be subscribed to this mapping. **Returns**: a HTTP 201 created status, with a link to the newly created mapping. Note that clients must persist the :uuid for later recall. **Error**: An HTTP 400 bad request if the model could not be parsed as a valid DEVA RDF.
GET /deva/mappings/:uuid	**Expects**: empty body **Returns**: an RDF/XML representation of the DEVA model with the mapping done to all supported IaaS providers. **Error**: a HTTP 404 not found if mapping does not exist.
DELETE /deva/mappings/:uuid	**Expects**: empty body **Returns**: a HTTP 200 OK. **Error**: a HTTP 404 not found if mapping does not exist.
POST /deva/mappings/:uuid/data_point	**Expects**: performance monitoring data of appliances of the specified mapping. Data Point should specify Appliance id, Connection id, and data value. (I.e. {appliance = 2, connection = {db-consumer, db-provider}, data_value = 500}) **Returns**: a HTTP 200 OK. **Error**: a HTTP 404 not found if mapping does not exist.
POST /deva/mappings/:uuid/subscriptions	**Expects**: an URL callback address to query when/if the specified mapping changes. Mappings could change in response to underprovisioned / overprovisioned resources as attested by monitoring data. The URL can be any valid endpoint. **Returns**: a HTTP 200 OK. **Error**: A HTTP 404 not found if mapping does not exist. **Error**: A HTTP 400 bad request if the callback_URL is not provided or malformed.
DELETE /deva/mappings/:uuid/subscriptions	**Expects**: A :callback_url parameter. **Returns**: A HTTP 200 OK. **Error**: a HTTP 404 not found if mapping does not exist. **Error**: A HTTP 400 bad request if the callback parameter is not provided or malformed.

TABLE I
APPLICATION PROGRAMMING INTERFACE FOR THE SOLUTION.

the model is processed by a set of rules that identify the virtual appliances included. For each one of the virtual appliances, a check is done on whether the engine has sufficient data to apply resource usage prediction based on machine learning techniques. If no sufficient data points have been gathered, then the solution applies a set of heuristic rules. If sufficient data is present, then it applies machine learning. In any case, a transformed model is created which includes IaaS bundles mappings. In the current prototype, we built an engine that utilizes rules to accomplish the above steps. Specifically, we are using the rules inference support that comes with the JENA 2.6.4 semantic web framework [9].

For the machine learning phase, we chose a multivariate linear regression model. We are experimenting with other machine learning techniques, but we only report on the regression-based one in this paper. The machine learning module receives the data points collected from the Monitoring API (see Figure 1) and estimates the QoS being achieved with the current resource allocation. In general, the parameters estimated are as follows:

$$\begin{aligned} targetMetric \quad &= \quad A_1 * CPU + A_2 * Memory + \\ & \qquad A_3 * Network + A_4 * Workload + B \end{aligned}$$

Where the $targetMetric$ would be the metric we are trying to ensure. Each one of the A_i coefficients describe how important the estimation of the ith term is.

For example, given enough data points, a CPU-bound application will have a large A_1 coefficient compared to the others. The $Workload$ and the $targetMetric$ are known, but the CPU, $Memory$ and $Network$ parameters are not. This formula has three degrees of freedom, therefore trying to optimize the parameters would be expensive and many data points would be needed.

To avoid this expensive calculation, we apply a key insight: given that most IaaS providers offer bundles, rather than arbitrary combinations of CPU, $Memory$ and $Network$, we can reduce the complexity of the problem by estimation on bundled resources instead of trying to guess if an application

Amazon [6] ("Instance type")	**t1.micro**: 2.2Ghz (varies), 613MB, "Low" network **m1.small**: 1.1GHz, 1.7GB, "Moderate" network **m1.large**: 4.4GHz, 7.5GB, "High" network **m1.xlarge**: 8.8GHz, 15GB, "Moderate" network
Rackspace [11] ("Flavor")	**Flavor 1**: 256 MB RAM **Flavor 2**: 512 MB RAM **Flavor 3**: 1024 MB RAM **Flavor 4**: 2048 MB RAM **Flavor 5**: 4096 MB RAM **Flavor 6**: 8192 MB RAM **Flavor 7**: 15872 MB RAM
ElasticHost [10] (No bundles.)	Anything in the following ranges: 2-20Ghz CPU, 1-8GB RAM

TABLE II
EXAMPLE RESOURCE BUNDLES OF THREE IAAS PROVIDERS.

is bound by any specific resource:

$$targetMetric = A_1 * Bundle + A_2 * Workload + B \quad (1)$$

We further reduce the calculation and estimation errors by keeping a regression table that includes all available bundles. This effectively reduces the problem to a simple linear regression:

$$Bundle \longrightarrow targetMetric = A_1 * Load + B \quad (2)$$

Table II presents a sample of the resource bundles available from three different IaaS providers. For each provider, we construct a regression that matches each of their resource bundles against the application, and then we choose the best fit (I.e. the cheapest one). Some $targetMetric$s should be approached by the left, and others by the right, so we take this into account for the fit. For example, a "maximum response time" metric would be fit by the left, while a "minimum required throughput" metric would be fit by the right.

Note that some IaaS providers, such as ElasticHost [10], do not provide discrete bundles and instead the customer can choose exactly the amount of resources needed. In these cases, we simply quantize the range according to the bundles available from an arbitrary competitor.

IV. EXPERIMENTS AND ANALYSIS

The following experiments were designed as a proof of concept for our approach. For each one of them, we describe the experiment, present results and elaborate a short analysis.

A. *Experiment 1 - Simulation*

For this experiment, we test our Transformation Engine with a DEVA model composed of one Virtual Appliance that simulates the behavior of a CPU-bound Web Framework like Ruby on Rails. The model includes a maximum response time constraint. We first train the system with synthetic data points that cover various IaaS standard bundles as seen in Table II. After the training, we observe what mappings the system suggest to comply with different response time targets for a fixed workload of 3 request per second. Figure 3 shows the results. On the x-axis, we present response time targets, while on the y-axis we present the suggested mapping to the bundles of a particular IaaS provider. Note that this experiment is a validation of the approach and that the numbers do not reflect the real performance of the IaaS service from Amazon, Rackspace or ElasticHost.

Subfigure 3a presents the simulation results for mapping the model to the bundles of Amazon. As can be seen, for very strict response times of 100 to 400ms, a mapping to a "m1.xlarge" bundle is required. As the response time constraint relaxes, the solution maps the model to smaller resource bundles.

Subfigure 3b presents the simulation results for mapping the model to the bundles of Rackspace. Although the mapping was gradual on the case of Amazon, in this case we can observe that for a CPU-bound model and a target response time of around 600ms, the solution suggest that we switch from a "flavor 4" bundle to a "flavor 6", skipping over "flavor 5".

Subfigure 3c presents the simulation results for mapping the model to the quantized bundles of ElasticHost. According to the simulation, a mapping can be achieved for targets above 450ms, yet a response time of 425ms or less can not be achieved.

B. *Experiment 2 - Real Allocation*

For this experiment, we train the system in a similar way as in Experiment 1, but data is gathered from real instances on Amazon EC2 running a CPU-bound Ruby on Rails application. We provision instances for the standard Amazon bundles as seen in table II and then we generate a positive slope linear workload on each of the bundles to gather performance data points. We plot the mapping our solution recommends for workloads of 3, 6, and 10 requests per second. We also include two additional baseline mappings for comparison: a mapping that always over-provisions, and a mapping that always under-provisions resources.

Figure 4 presents the results. Note that the baseline mapping that always under-provisions is not able to fulfill the QoS, while the baseline that always over-provisions can fulfill the QoS, but utilizes a maximum of resources all the time. We can observe that our solution suggests bundle mappings that depend on the target metric, in this case the response time, as well as in the workload. Further, our solution fulfills the QoS of the application with a minimum relative error of 5.61%, maximum of 34.07%, and average of 17.49%.

C. *Experiment 3 - Time and Scalability Analysis*

For this experiment, we asses the time complexity and scalability of the solution. We ran these experiments on a 2.8GHz Intel i7 quad core machine with 8GB of available RAM. Figure 5 presents the results.

For the time analysis, we use the solution's API to POST a new model, POST a variable number of performance samples to the monitoring API, and then GET back the transformed models with mappings. Note that this is a worst case scenario, as normally the performance samples will not be gathered this fast. On subfigure 5a, we plot the response time of our solution against the number of collected data points from the monitoring API. The results suggest that the prototype can respond to GET model requests, which could trigger the

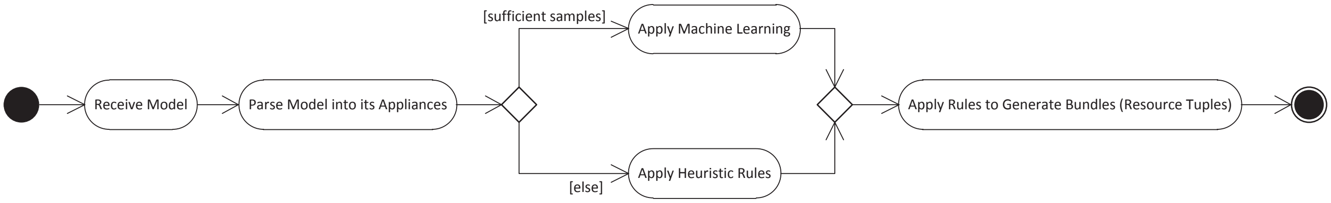
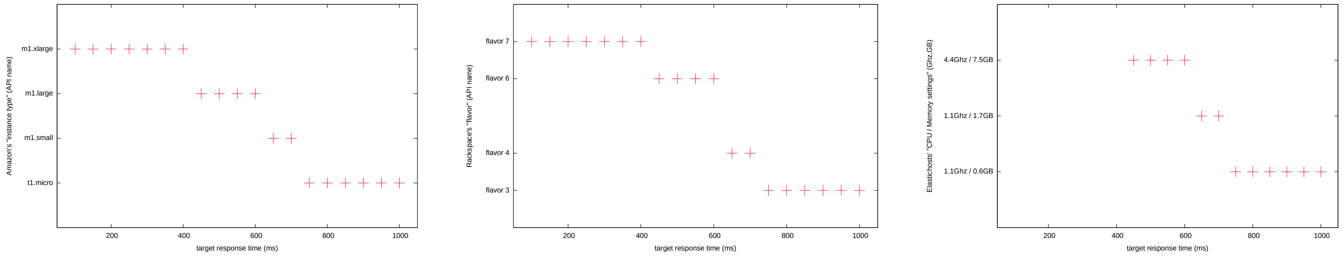


Fig. 2. UML Activity Diagram modeling the steps to map a DEVA model to IaaS Resources.



(a) Amazon Mapping.

(b) Rackspace Mapping.

(c) ElasticHost Mapping.

Fig. 3. Evaluation of the solution using simulated data points for a fixed workload of 3 request per second.

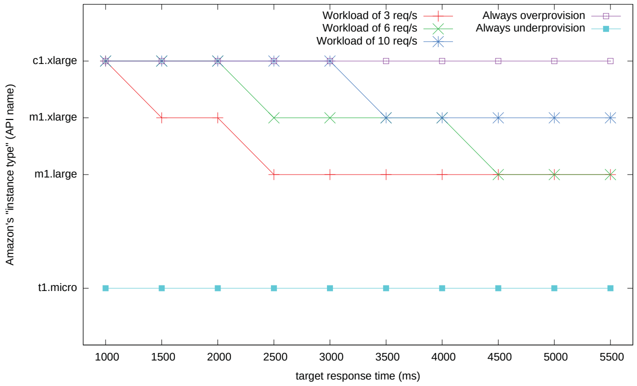


Fig. 4. Evaluation of the solution running a CPU-bound application on top of Amazon EC2. Note that results are discrete data points, but are shown with lines with points.

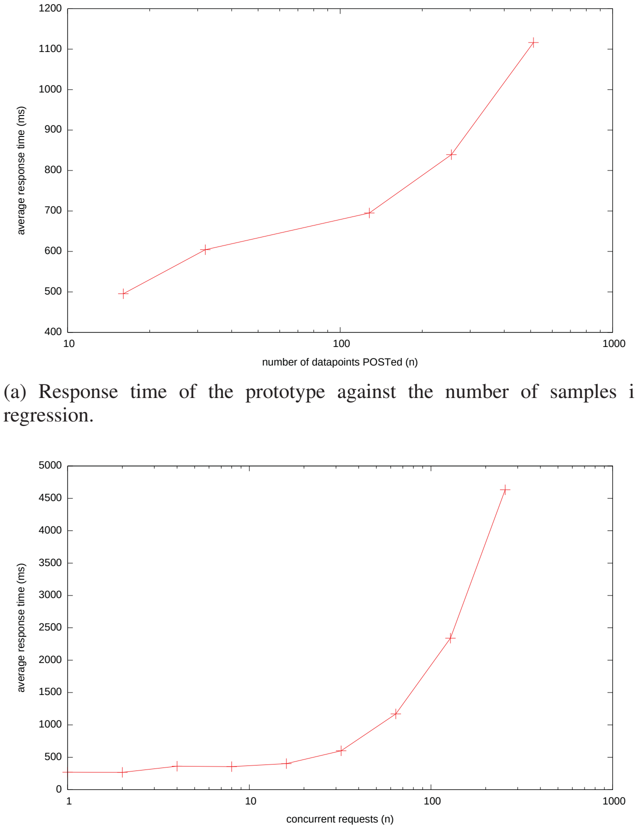
machine learning algorithm, in less than a second for up to 256 gathered data points. For a sample size of 512, we get a response time of 1116ms. For bigger sample sizes, the performance starts to quickly degrade in a quadratic manner. Nonetheless, experiential data suggest that we only need the performance test data points that we discussed in Subsection II-B2 and recent workload data points to achieve acceptable accuracy.

For the scalability analysis, we fix the acquired data points for a particular model to 64, and plot the response time of our solution against concurrent GET model requests. Subfigure 5b presents the results. The solution can achieve a subsecond response time for up to 32 concurrent requests. At 64 concurrent requests, the solution can respond in an average of 1170ms. For bigger concurrent requests, the performance starts to quickly degrade in a quadratic manner, even more so than in the previous experiment. For practical cases, our solution can respond to a maximum of 100 concurrent requests with an average response time of 2000ms.

V. Related Work

In [12], Stewart and Chen presented an implementation of an offline profile-driven performance model for cluster-based multi-component online services. Their model includes a detailed specification of the application. Although our solution also utilizes a detailed specification of the application, our work in [3] makes it straightforward for a solution developer to construct it. Additionally, we do online monitoring of the application to respond to workload changes. In [13], Sadjadi et al. proposed a regression model that estimates CPU usage for long-running scientific applications. In our work, we are estimating bundles of CPU, Memory, and Network, and the solution is targeting web cloud application workalods.

In [2], Islam et al. estimate CPU usage by simulating a cloud provider. They contrast the use of linear regression and neural networks. Our work includes both a simulation as well as a real experiment of the solution on top of Amazon EC2, although we do not compare different machine learning models. In [14] Villegas and Sadjadi presented an IaaS solution that can have as input a model of a cloud application with non-functional specifications. Our work is complementary, as our DEVA models could be used for input to their solution. Also, in our work, we do not assume that the IaaS provider understands our model, and thus our solution is IaaS-agnostic. In [1], Ganapathi et al. utilize statistical machine learning to predict resource usage of an application in the cloud. Their workload is similar to the workload of [13], that is, batch processing of long-running jobs. We focus on web cloud application workloads.



(a) Response time of the prototype against the number of samples in regression.

(b) Response time of the prototype against concurrent requests with samples fixed to 64.

Fig. 5. Time and Scalability Analysis of the solution.

Some IaaS clouds like Amazon's EC2 already provide an auto-scaling API [6]. These APIs monitor low-level resources like CPU-usage, instead of our approach of directly monitoring key QoS metrics like response time. These vendor APIs focus on resource usage, while our solution's focus is on application performance non-functional requirements (i.e. QoS constraints).

In [15], Ejarque et al. propose the usage of semantics for enhancing the resource allocation in distributed platforms. They propose a set of extensions in resource ontologies and a set of rules for modeling resource allocation policies. A similar approach has been followed in their subsequent paper [16]. In these two cases, rules are used to model equivalences and mappings between the different cloud providers models. Thus, when the system receives a request following a providers model, it can be automatically transformed to another provider by applying the mapping rules to the original request. Our work is complementary, as we apply this rule mapping approach to a different problem. We map application descriptions, provided as DEVA models, into IaaS resource bundles.

VI. Conclusion

In this paper, we first presented the design of an autonomic solution for cloud application resource mapping and scaling based on monitoring of QoS constraints. We then provided details on the prototype implementation and how we dealt with the technical challenges. Finally, we assessed the validity of the approach by presenting experiments on functionality and scalability.

For future work, we intend to expand the prototype in two directions. In the near future, we will introduce other machine learning algorithms into our resource allocation framework and produce a comparison to see which techniques work best for this problem. In the longer term, we intent to expand our solution to consider not only vertical, but also horizontal scaling of Virtual Appliances. That is, to dynamically modify the DEVA model architecture if the QoS requirements are hard to achieve with the current one.

Acknowledgment

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 and IIP-0829576.

References

[1] A. Ganapathi, Y. Chen, A. Fox, R. Katz, and D. Patterson, "Statistics-driven workload modeling for the cloud," in *IEEE 26th International Conference on Data Engineering Workshops*, 2010, pp. 87–92.

[2] S. Islam, J. Keung, K. Lee, and A. Liu, "Empirical prediction models for adaptive resource provisioning in the cloud," in *The International Journal of Grid Computing and Escience*. Natl ICT Australia, Software Engn Res Grp, Sydney, NSW, Australia, 2012, pp. 155–162.

[3] 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.

[4] X. J. Collazo-Mojica and S. M. Sadjadi, "A Metamodel for Distributed Ensembles of Virtual Appliances," in *International Conference on Software Engineering and Knowledge Engineering*, May 2011, pp. 560–565.

[5] C. Sapuntzakis, D. Brumley, R. Chandra, N. Zeldovich, J. Chow, M. S. Lam, and M. Rosenblum, "Virtual appliances for deploying and maintaining software," *USENIX Large Installation Systems Administration Conference*, pp. 181–194, Aug 2003.

[6] "Amazon Elastic Compute Cloud," March 2012. [Online]. Available: http://aws.amazon.com/ec2/

[7] "Resource Description Framework (RDF)," March 2012. [Online]. Available: http://www.w3.org/RDF/

[8] "OWL Web Ontology Language," March 2012. [Online]. Available: http://www.w3.org/TR/owl-ref/

[9] "Apache JENA," March 2012. [Online]. Available: http://incubator.apache.org/jena/

[10] "ElasticHosts," March 2012. [Online]. Available: http://www.elastichosts.com/

[11] "Rackspace Cloud," March 2012. [Online]. Available: http://www.rackspace.com/cloud/

[12] C. Stewart and K. Shen, "Performance Modeling and System Management for Multi-component Online Services," in *2nd Symposium on Networked Systems Design & Implementation*, May 2005, pp. 71–84.

[13] 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 *IPDPS*, 2008, pp. 1–8.

[14] D. Villegas and S. M. Sadjadi, "Mapping Non-Functional Requirements to Cloud Applications," *International Conference on Software Engineering and Knowledge Engineering*, Jun. 2011.

[15] J. Ejarque, R. Sirvent, and R. Badia, "A Multi-agent Approach for Semantic Resource Allocation," in *Cloud Computing Technology and Science*, 2010, pp. 335–342.

[16] J. Ejarque, J. Alvarez, R. Sirvent, and R. Badia, "A Rule-based Approach for Infrastructure Providers' Interoperability," in *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*, 2011, pp. 272–279.

An Empirical Study of Software Metric Selection Techniques for Defect Prediction

Huanjing Wang, Taghi M. Khoshgoftaar, Randall Wald, and Amri Napolitano
{huanjing.wang@wku.edu, khoshgof@fau.edu, rwald1@fau.edu, amrifau@gmail.com}

***Abstract*—In software engineering, a common classification problem is determining the quality of a software component, module, or release. To aid in this task, software metrics are collected at various states of a software development cycle, and these metrics can be used to build a defect prediction model. However, not all metrics are relevant to defect prediction. One solution to finding the relevant metrics is the data preprocessing step known as feature selection. We present an empirical study in which we evaluate the similarity of eighteen different feature selection techniques and how the feature subsets chosen by each of these techniques perform in defect prediction. We look at similarity in addition to classification because many applications seek a diverse set of rankers, and similarity can be used to find which rankers are too close together to provide diversity. The classification models are trained using three commonly-used classifiers. The case study is based on software metrics and defect data collected from multiple releases of a large real-world software system. The results show that the features fall into a number of identifiable clusters in terms of similarity. In addition, the similarity clusters were somewhat predictive of the clusters based on classification ranking: rankers within a similarity cluster had similar classification performance, and thus ended up in the same or adjacent classification clusters. The reverse was not true, with some classification clusters containing multiple unrelated similarity clusters. Overall, we found that the signal-to-noise and ReliefF-W rankers selected good features while being dissimilar from one another, suggesting they are appropriate for choosing diverse but high-performance rankers.**

I. Introduction

In the practice of software quality assurance, software metrics are often collected and associated with modules, which have their number of pre- and post-release defects recorded. A software defect prediction model is often built to ensure the quality of future software products or releases. However, not all software metrics are relevant for predicting the fault proneness of software modules. Software metrics selection (or feature selection) prior to training a defect prediction model can help separate relevant software metrics from irrelevant or redundant ones.

In this paper, we focus on feature selection of software metrics for defect prediction. During the past decade, numerous studies have examined feature selection with respect to classification performance, but very few studies focus on the similarity of feature selection techniques. The purpose of studying this similarity is to make it easier to select a set of diverse rankers, ensuring that none of those chosen are so similar to the others as to provide no additional diversity in the collection of rankers. In this study, we perform similarity analysis on eighteen different feature selection techniques, eleven of which were recently developed and implemented by our research group. We evaluate the similarity of two filters on a dataset by measuring the consistency between the two feature subsets chosen. We also evaluate the effectiveness of defect predictors that estimate the quality of program modules, e.g., fault-prone (*fp*) or not-fault-prone (*nfp*). Three different classifiers (learners) are used to build our prediction models. The empirical validation of the similarity measure and model performance was implemented through a case study of four consecutive releases of a very large telecommunications software system (denoted as LLTS). To our knowledge this is the first study to examine both similarity and classification performance of feature rankers in the software engineering domain.

The experimental results show that using both the similarity and classification performance of the rankers gave different types of clusters. Similarity produced a number of smaller clusters, with four or five rankers being about the largest sizes found (and two and one being common sizes as well). Classification gave a smaller number of clusters, with the largest cluster (also the best-performing cluster) having nine members. In both cases, the clusters often contained feature rankers which would not immediately seem to have much in common; although this makes sense for classification performance, it is an intriguing result for the similarity (indicating that the rankers may be more related than one might expect). We also found that the similarity groups were generally predictive of the classification groupings: members within a single similarity group are in the same or adjacent classification groups. Finally, we noted that signal-to-noise and ReliefF-W performed very well in terms of classification while choosing two or fewer features in common.

The rest of the paper is organized as follows. We review relevant literature on feature selection techniques in Section II. Section III provides detailed information about the 18 feature selection techniques. Section IV describes the datasets used in the study, presents similarity results and analysis, and shows model performance results and analysis. Finally, in Section V, the conclusion is summarized and suggestions for future work are indicated.

II. Related Work

The main goal of feature selection is to select a subset of features that minimizes the prediction errors of classifiers. Feature selection can be broadly classified as *feature ranking* and *feature subset selection*. Feature ranking sorts the attributes according to their individual predictive power, while feature subset selection finds subsets of attributes that collectively have good predictive power. Feature selection can also be categorized as *filters* and *wrappers*. Filters are algorithms in which a feature subset is selected without involving any learning algorithm. Wrappers are algorithms that use feedback from a learning algorithm to determine which feature(s) to include in building a classification model.

A number of papers have studied the use of feature selection techniques as a data preprocesing step. Guyon and Elisseeff [1] outline key approaches used for attribute selection, including feature construction, feature ranking, multivariate feature selection, efficient search methods, and feature validity assessment methods. A study by Liu and Yu [2] provides a comprehensive survey of feature selection algorithms and presents an integrated approach to intelligent feature selection. Jeffery et al. [3] compare the similarity between gene lists produced by 10 different feature selection methods. They conclude that sample size clearly affects the ranked gene lists produced by different feature selection methods.

Feature selection has been applied in many data mining and machine learning applications. However, its application in the software quality and reliability engineering domain is limited. Chen et al. [4] have studied the applications of wrapper-based feature selection in the context of software cost/effort estimation. They concluded that the reduced dataset improved the estimation. In a recent study [5] by Gao et al., a comparative investigation in the context of software quality estimation is presented for evaluating a proposed hybrid attribute selection approach, in which feature ranking is first used to reduce the search space, followed by a feature subset selection.

III. Filter-based Feature Rankers

This work focuses on filter-based feature ranking. Filter-based feature ranking techniques rank features independently without involving any learning algorithm. We chose this class of feature selection algorithm because for large datasets, feature subset evaluation (including wrappers) can be computationally prohibitive. In this work, the feature rankers (filters) chosen can be placed into two categories: eleven threshold-based feature selection techniques (TBFS) that were developed by our research team and seven non-TBFS feature selection techniques including six commonly-used filters and a new filter technique called signal-to-noise.

A. Non-TBFS Feature Selection Techniques

Seven non-TBFS filter-based feature ranking techniques were used in this work: chi-squared (CS) [6], information gain (IG) [6], gain ratio (GR) [6], two versions of ReliefF (ReliefF, RF, and ReliefF-W, RFW) [7], symmetric uncertainty (SU) [8], and signal-to-noise (S2N) [9]. All of these feature selection methods, with the exception of signal-to-noise, are available within the WEKA machine learning tool [6]. Outside of RFW, WEKA's default parameter values were used. RFW is the ReliefF technique with the weight by distance parameter set to "true". Since most of these methods are widely known and for space considerations, the interested reader can consult with the included references for further details.

Signal-to-noise ratio is a measure used in electrical engineering to quantify how much a signal has been corrupted by noise. It is defined as the ratio of the signal's power to the noise's power corrupting the signal. The signal-to-noise (S2N) can also be used as feature ranking method ([9]). For a binary class problem (such as fp, nfp), the S2N is defined as the ratio of the difference of class means ($\mu_{fp} - \mu_{nfp}$) to the sum of standard deviation of each class ($\sigma_{fp} + \sigma_{nfp}$). If one attribute's expression in one class is quite different from its expression in the other, and there is little variation within the two classes, then the attribute is predictive. Therefore, S2N favors attributes where the range of the expression vector is large, but where most of that variation is due to the class distribution. Because it is rarely used as a feature ranking technique, we use our own implementation of S2N.

B. Threshold-based Feature Ranking Techniques

Eleven threshold-based feature selection techniques were developed and implemented by our research group within WEKA [6]. The procedure is shown in Algorithm 1. First each attribute's values are normalized between 0 and 1 by mapping F^j to $\hat{F}^j$. The normalized values are treated as posterior probabilities. Each independent attribute (software predictor variable) is then paired individually with the class attribute (fault-prone or not-fault-prone label) and the reduced dataset is evaluated using eleven different classifier performance metrics based on a set of posterior probabilities. In standard binary classification, the predicted class is assigned using the default decision threshold of 0.5. The default decision threshold is often not optimal, especially when the relative class distribution is imbalanced. Therefore, we propose the use of performance metrics that can be calculated at various points in the distribution of $\hat{F}^j$. At each threshold position, the values above the threshold are classified as positive, and negative otherwise. We then consider swapping the positive and negative, i.e. values above the threshold are classified as negative, and positive otherwise. Whichever direction of the positive and negative labeling produces the more optimal attribute values is used. In a binary classification problem such as fault-prone (positive) or not-fault-prone (negative), there are four possible classification rates: true positive rate (TPR), true negative rate (TNR), false positive rate (FPR), false negative rate (FNR), as well as one additional commonly-used performance metric, precision (PRE). These four classification rates can be calculated at each threshold $t \in [0, 1]$ relative to the normalized attribute $\hat{F}^j$. The threshold-based feature ranking technique utilizes the classification rates as described below.

Algorithm 1: Threshold-based Feature Selection Algorithm

```
input :
1. Dataset $D$ with features $F^j, j = 1, \ldots, m$;
2. Each instance $x \in D$ is assigned to one of two classes $c(x) \in \{fp, nfp\}$;
3. The value of attribute $F^j$ for instance $x$ is denoted $F^j(x)$;
4. Threshold-based feature ranking technique $\omega \in$ {FM, OR, PO, PR, GI, MI,
KS, DV, GM, AUC, PRC};
5. A predefined threshold: number (or percentage) of the features to be selected.
output:
Selected feature subsets.
for $F^j, j = 1, \ldots, m$ do
    Normalize $F^j \mapsto \hat{F}^j = \frac{F^j - \min(F^j)}{\max(F^j) - \min(F^j)}$;
    Calculate metric $\omega$ using attribute $\hat{F}^j$, $\omega_i(\hat{F}^j)$.
Create feature ranking $\mathbb{R}$ using $\omega_i(\hat{F}^j)\forall j$.
Select features according to feature ranking $\mathbb{R}$ and a predefined threshold.
```

- *F-measure (FM)*: is a single value metric derived from the F-measure that originated from the field of information retrieval [6]. The maximum F-measure is obtained when varying the decision threshold value between 0 and 1.
- *Odds Ratio (OR)*: is the ratio of the product of correct (TPR times TNR) to incorrect (FPR times FNR) predictions. The maximum value is taken when varying the decision threshold value between 0 and 1.
- *Power (PO)*: is a measure that avoids common false positive cases while giving stronger preference for positive cases [10]. Power is defined as:

  $$\text{PO} = \max_{t \in [0,1]} \left((TNR(t))^k - (FNR(t))^k \right)$$

  where $k = 5$.
- *Probability Ratio (PR)*: is the sample estimate probability of the feature given the positive class divided by the sample estimate probability of the feature given the negative class [10]. PR is the maximum value of the ratio when varying the decision threshold value between 0 and 1.
- *Gini Index (GI)*: measures the impurity of a dataset. GI for the attribute is then the minimum Gini index at all decision thresholds $t \in$[0, 1].
- *Mutual Information (MI)*: measures the mutual dependence of the two random variables. High mutual information indicates a large reduction in uncertainty, and zero mutual information between two random variables means the variables are independent.
- *Kolmogorov-Smirnov (KS)*: utilizes the Kolmogorov-Smirnov

TABLE I
SOFTWARE DATASETS CHARACTERISTICS

	Data	#Metrics	#Modules	*%fp*	*%nfp*
LLTS	SP1	42	3649	6.28%	93.72%
	SP2	42	3981	4.75%	95.25%
	SP3	42	3541	1.33%	98.67%
	SP4	42	3978	2.31%	97.69%

statistic to measure the maximum difference between the empirical distribution functions of the attribute values of instances in each class [11]. It is effectively the maximum difference between the curves generated by the true positive and false positive rates as the decision threshold changes between 0 and 1.

- *Deviance (DV)*: is the residual sum of squares based on a threshold t. That is, it measures the sum of the squared errors from the mean class given a partitioning of the space based on the threshold t and then the minimum value is chosen.
- *Geometric Mean (GM)*: is a single-value performance measure which is calculated by finding the maximum geometric mean of TPR and TNR as the decision threshold is varied between 0 and 1.
- *Area Under ROC (Receiver Operating Characteristic) Curve* (AUC): has been widely used to measure classification model performance [12]. The ROC curve is used to characterize 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.
- *Area Under the Precision-Recall Curve* (PRC): is a single-value measure that originated from the area of information retrieval. The area under the PRC ranges from 0 to 1. The PRC diagram depicts the trade off between recall and precision.

IV. EXPERIMENTS

A. Dataset

Experiments conducted in this study used software metrics and defect data collected from a real-world software project, a very large telecommunications software system (denoted as LLTS) [5]. LLTS contains data from four consecutive releases, which are labeled as SP1, SP2, SP3, and SP4. The software measurement datasets consist 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, fault-prone (*fp*), or not fault-prone (*nfp*). A program module with one or more faults is considered *fp*, and *nfp* otherwise. Table I lists the characteristics of the four release datasets utilized in this work. An important characteristic of these datasets is that they all suffer from class imbalance, where the proportion of *fp* modules is much lower than that of *nfp* modules.

B. Experimental Design

We first used 18 filter-based rankers to select the subsets of attributes. We ranked the features and selected the top $\lceil \log_2 n \rceil$ features according to their respective scores, where n is the number of independent features for a given dataset. The reasons why we select the top $\lceil \log_2 n \rceil$ features include (1) related literature does not provide guidance on the appropriate number of features to select; and (2) a recent study [13] showed that it was appropriate to use $\lceil \log_2 n \rceil$ as the number of features when using WEKA [6] to build Random Forests learners for binary classification in general and imbalanced datasets in particular. Although we used different learners here, a preliminary study showed that $\lceil \log_2 n \rceil$ is still appropriate for various learners. In this study, six ($\lceil \log_2 42 \rceil = 6$) features are selected.

The experiments were conducted to discover the similarity of 18 rankers and the impact of rankers and learners on defect prediction.

C. Experiments - Similarity between 18 Rankers

In this study, we examine the similarity between 18 filter-based feature ranking techniques (rankers). The empirical validation of the different rankers was implemented through a case study of four consecutive releases (SP1, SP2, SP3, and SP4), of a very large telecommunications software system (denoted as LLTS).

Consistency index [14] is used to measure the degree of similarity between the metric rankings of two feature selection techniques. In this paper, we use the consistency index because it takes into consideration bias due to chance. We compute the consistency index between two feature subsets as follows. Let T_i and T_j be subsets of features, where $|T_i| = |T_j| = k$. The consistency index [14] is obtained as follows:

$$I_C\left(T_i, T_j\right) = \frac{dn - k^2}{k\left(n - k\right)}, \tag{1}$$

where n is the total number of features in the dataset, d is the cardinality of the intersection between subsets T_i and T_j, and $-1 < I_C\left(T_i, T_j\right) \leq +1$. The greater the consistency index, the more similar the subsets are. As noted earlier, k is set to 6 in this study. If the two rankings are the same, then the consistency index is 1. If there is no overlap between two rankings, then the consistency index is *-0.1667*. For grouping rankers, we consider two rankers to be similar if they share four out of the top six features. Tables II through V show the experimental results; for convenience, pairs with 4 or more features in common are bolded, while those with one or zero in common are printed in italics.

1) Seven Non-TBFS Rankers
The experimental results show that among the seven non-TBFS rankers (CS, IG, GR, RF, RFW, SU, and S2N), RF and RFW generate very similar feature subsets, in fact the same feature subsets for the datasets SP2 and SP4; the feature subsets generated by RF and RFW are dissimilar to the feature subset generated by the other five ranker groups. In general, we can group these seven rankers into three groups; {RF, RFW}, {CS, IG, S2N}, and {GR, SU}. The rankers within each group produce similar feature subsets.
2) 11 Threshold-based Rankers
We can divide the 11 TBFS rankers into five groups, {PRC, DV, FM, PO}, {AUC, KS}, {PR, OR}, {MI, GM}, and {GI}. The rankers in each group generate similar feature subsets as defined above. We also observe that OR, PR, and GI are particularly dissimilar to the other eight rankers.
3) All 18 Rankers
When we consider all eighteen rankers in two groups (7 non-TBFS and 11 TBFS), we can observe that CS is similar to FM, PO, DV, and PRC; IG is similar to PO, DV, PRC; and S2N is similar to FM, PO, and PRC. We also observe that RF and RFW are still dissimilar to all other rankers.

D. Experiments - Defect Prediction Model Performance

To evaluate a feature selection technique, similarity of feature selection may not be enough. We should also consider classification model performance. Software defect prediction models have been used to improve the fault prediction and risk assessment process. Finding faulty components in a software system can lead to a more reliable final system and reduce development and maintenance costs.

TABLE II
SP1, Similarity Between Rankers

	CS	GR	IG	RF	RFW	SU	FM	OR	PO	PR	GI	MI	KS	DV	GM	AUC	PRC	S2N
CS		0.2222	**0.8056**	*0.0278*	*0.0278*	0.4167	**0.6111**	0.4167	**0.6111**	0.4167	0.4167	**0.6111**	0.4167	**0.6111**	0.4167	0.2222	**0.6111**	0.4167
GR			0.2222	*0.0278*	*0.0278*	**0.6111**	0.2222	*0.0278*	0.2222	*-0.1667*	*-0.1667*	0.2222	0.2222	0.2222	0.2222	0.2222	0.2222	*0.0278*
IG				*0.0278*	*0.0278*	0.4167	**0.6111**	0.2222	**0.6111**	0.2222	0.2222	**0.6111**	0.4167	**0.6111**	0.4167	0.2222	**0.6111**	0.4167
RF					**0.8056**	*0.0278*	*0.0278*	*-0.1667*	*0.0278*	*-0.1667*	*-0.1667*	*0.0278*	*0.0278*	*0.0278*	*0.0278*	*0.0278*	*0.0278*	0.2222
RFW						*0.0278*	*0.0278*	*0.0278*	*0.0278*	*0.0278*	*0.0278*	*0.0278*	*0.0278*	*0.0278*	*0.0278*	*0.0278*	*0.0278*	0.2222
SU							0.4167	0.2222	0.4167	*-0.1667*	*-0.1667*	0.4167	0.2222	0.4167	0.2222	0.2222	0.4167	0.2222
FM								0.4167	**0.8056**	0.2222	0.2222	**0.6111**	**0.8056**	**0.6111**	**0.6111**	0.4167	**1.0000**	**0.8056**
OR									0.4167	**0.6111**	**0.6111**	*0.0278*	0.2222	0.2222	*0.0278*	*-0.1667*	0.4167	0.4167
PO										0.2222	0.2222	0.4167	**0.6111**	**0.6111**	0.4167	0.4167	**0.8056**	**0.6111**
PR											**1.0000**	*0.0278*	0.2222	0.2222	*0.0278*	*-0.1667*	0.2222	0.2222
GI												*0.0278*	0.2222	0.2222	*0.0278*	*-0.1667*	0.2222	0.2222
MI													**0.6111**	**0.8056**	**0.6111**	**0.6111**	**0.6111**	0.4167
KS														**0.6111**	**0.8056**	**0.6111**	**0.8056**	**0.6111**
DV															0.4167	0.4167	**0.6111**	0.4167
GM																**0.6111**	**0.6111**	0.4167
AUC																	0.4167	0.2222
PRC																		**0.8056**
S2N																		

TABLE III
SP2, Similarity Between Rankers

	CS	GR	IG	RF	RFW	SU	FM	OR	PO	PR	GI	MI	KS	DV	GM	AUC	PRC	S2N
CS		*-0.1667*	**0.8056**	*0.0278*	*0.0278*	0.2222	**0.6111**	0.2222	**0.8056**	0.2222	0.2222	**0.6111**	**0.6111**	**0.8056**	**0.6111**	**0.8056**	**0.8056**	**0.6111**
GR			*-0.1667*	*-0.1667*	*-0.1667*	0.4167	*-0.1667*	*0.0278*	*-0.1667*	*-0.1667*	*0.0278*	*-0.1667*	*-0.1667*	*-0.1667*	*-0.1667*	*-0.1667*	*-0.1667*	*-0.1667*
IG				*0.0278*	*0.0278*	0.2222	0.4167	0.2222	**0.6111**	0.2222	0.2222	**0.6111**	**0.6111**	**0.6111**	**0.6111**	**1.0000**	**0.8056**	**0.6111**
RF					**1.0000**	*-0.1667*	*0.0278*	0.2222	*0.0278*	0.2222	0.2222	*0.0278*	*-0.1667*	*0.0278*	*-0.1667*	*0.0278*	*0.0278*	*0.0278*
RFW						*-0.1667*	*0.0278*	0.2222	*0.0278*	0.2222	0.2222	*0.0278*	*-0.1667*	*0.0278*	*-0.1667*	*0.0278*	*0.0278*	*0.0278*
SU							0.2222	0.2222	0.4167	*0.0278*	0.2222	0.4167	0.4167	0.2222	0.4167	0.2222	0.2222	*0.0278*
FM								*0.0278*	**0.8056**	*0.0278*	*0.0278*	**0.6111**	0.4167	**0.6111**	0.4167	0.4167	**0.6111**	**0.6111**
OR									0.2222	**0.6111**	**0.6111**	0.2222	*0.0278*	0.2222	*0.0278*	0.2222	0.2222	0.2222
PO										0.2222	0.2222	**0.8056**	**0.6111**	**0.8056**	**0.6111**	**0.6111**	**0.8056**	**0.6111**
PR											**0.8056**	0.2222	*0.0278*	0.2222	*0.0278*	0.2222	0.2222	*0.0278*
GI												0.2222	*0.0278*	0.2222	*0.0278*	0.2222	0.2222	*0.0278*
MI													**0.8056**	**0.8056**	**0.8056**	**0.6111**	**0.6111**	0.4167
KS														**0.6111**	**1.0000**	**0.6111**	0.4167	0.2222
DV															**0.6111**	**0.6111**	**0.8056**	**0.6111**
GM																**0.6111**	0.4167	0.2222
AUC																	**0.8056**	**0.6111**
PRC																		**0.8056**
S2N																		

TABLE IV
SP3, Similarity Between Rankers

	CS	GR	IG	RF	RFW	SU	FM	OR	PO	PR	GI	MI	KS	DV	GM	AUC	PRC	S2N
CS		**0.8056**	**1.0000**	0.2222	0.2222	**1.0000**	**0.6111**	*-0.1667*	**0.8056**	*-0.1667*	*0.0278*	**1.0000**	**0.6111**	**0.8056**	**0.6111**	**0.8056**	**0.8056**	**0.8056**
GR			**0.8056**	0.2222	0.2222	**0.8056**	0.4167	*0.0278*	**0.6111**	*0.0278*	*0.0278*	**0.8056**	0.4167	**0.6111**	0.4167	**0.8056**	**0.6111**	**0.6111**
IG				0.2222	0.2222	**1.0000**	**0.6111**	*-0.1667*	**0.8056**	*-0.1667*	*0.0278*	**1.0000**	**0.6111**	**0.8056**	**0.6111**	**0.8056**	**0.8056**	**0.8056**
RF					**0.8056**	0.2222	0.4167	*-0.1667*	0.2222	*-0.1667*	0.2222	0.2222	0.2222	0.2222	0.2222	0.2222	0.2222	0.2222
RFW						0.2222	0.4167	*-0.1667*	0.2222	*-0.1667*	0.2222	0.2222	*0.0278*	0.2222	*0.0278*	0.2222	0.2222	0.2222
SU							**0.6111**	*-0.1667*	**0.8056**	*-0.1667*	*0.0278*	**1.0000**	**0.6111**	**0.8056**	**0.6111**	**0.8056**	**0.8056**	**0.8056**
FM								*-0.1667*	**0.6111**	*0.0278*	0.2222	**0.6111**	0.4167	**0.8056**	0.4167	0.4167	**0.8056**	**0.6111**
OR									*-0.1667*	**0.8056**	*0.0278*	*-0.1667*	*-0.1667*	*-0.1667*	*-0.1667*	*-0.1667*	*-0.1667*	*-0.1667*
PO										*-0.1667*	*0.0278*	**0.8056**	**0.8056**	**0.8056**	**0.6111**	**0.8056**	**0.8056**	**0.8056**
PR											0.2222	*-0.1667*	*-0.1667*	*0.0278*	*-0.1667*	*-0.1667*	*0.0278*	*-0.1667*
GI												*0.0278*	*0.0278*	0.2222	*0.0278*	*0.0278*	0.2222	*0.0278*
MI													**0.6111**	**0.8056**	**0.6111**	**0.8056**	**0.8056**	**0.8056**
KS														**0.6111**	**0.8056**	**0.6111**	**0.6111**	**0.6111**
DV															**0.6111**	**0.6111**	**1.0000**	**0.6111**
GM																0.4167	**0.6111**	0.4167
AUC																	**0.6111**	**0.8056**
PRC																		**0.6111**
S2N																		

TABLE V
SP4, Similarity Between Rankers

	CS	GR	IG	RF	RFW	SU	FM	OR	PO	PR	GI	MI	KS	DV	GM	AUC	PRC	S2N
CS		**0.6111**	0.4167	*0.0278*	*0.0278*	**0.6111**	**0.6111**	*-0.1667*	**0.6111**	*-0.1667*	*0.0278*	0.2222	0.2222	**0.6111**	*0.0278*	*0.0278*	**0.6111**	**0.6111**
GR			0.4167	*-0.1667*	*-0.1667*	**1.0000**	**0.6111**	*-0.1667*	**0.6111**	*-0.1667*	*-0.1667*	0.2222	0.2222	**0.6111**	*0.0278*	*0.0278*	**0.6111**	0.4167
IG				*0.0278*	*0.0278*	0.4167	**0.6111**	*0.0278*	**0.6111**	*0.0278*	*-0.1667*	0.4167	0.2222	**0.6111**	0.4167	0.4167	**0.6111**	0.4167
RF					**1.0000**	*-0.1667*	*0.0278*	*-0.1667*	*0.0278*	*-0.1667*	*-0.1667*	*-0.1667*	*-0.1667*	*0.0278*	*-0.1667*	*0.0278*	*0.0278*	0.2222
RFW						*-0.1667*	*0.0278*	*-0.1667*	*0.0278*	*-0.1667*	*-0.1667*	*-0.1667*	*-0.1667*	*0.0278*	*-0.1667*	*0.0278*	*0.0278*	0.2222
SU							**0.6111**	*-0.1667*	**0.6111**	*-0.1667*	*-0.1667*	0.2222	0.2222	**0.6111**	*0.0278*	*0.0278*	**0.6111**	0.4167
FM								*0.0278*	**1.0000**	*0.0278*	*-0.1667*	0.2222	0.2222	**1.0000**	*0.0278*	0.2222	**1.0000**	**0.8056**
OR									*0.0278*	**0.8056**	0.2222	0.2222	*0.0278*	*0.0278*	0.2222	0.2222	*0.0278*	*0.0278*
PO										*0.0278*	*-0.1667*	0.2222	0.2222	**1.0000**	*0.0278*	0.2222	**1.0000**	**0.8056**
PR											0.2222	*0.0278*	*0.0278*	*0.0278*	0.2222	*0.0278*	*0.0278*	*0.0278*
GI												*-0.1667*	*-0.1667*	*-0.1667*	*-0.1667*	*-0.1667*	*-0.1667*	*-0.1667*
MI													**0.8056**	0.2222	**0.6111**	**0.6111**	0.2222	0.2222
KS														0.2222	0.4167	**0.6111**	0.2222	0.2222
DV															*0.0278*	0.2222	**1.0000**	**0.8056**
GM																**0.6111**	*0.0278*	*0.0278*
AUC																	0.2222	0.2222
PRC																		**0.8056**
S2N																		

1) Classifiers: In this study, software defect prediction models are built with three well-known classification algorithms: naïve Bayes (NB), multilayer perceptron (MLP), and logistic regression (LR). All three learners themselves do not have a built-in feature selection capability and are commonly used in the software engineering and other data mining applications. All classifiers were implemented in the WEKA tool with default parameter settings [6] except MLP. Based on preliminary research, the 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.

2) Performance Metric: Because using traditional performance measures such as classification accuracy can give misleading results on imbalanced data, we use a performance metric that considers the ability of a classifier to differentiate between the two classes: the area under the ROC (Receiver Operating Characteristic) curve (AUC). A perfect classifier provides an AUC that equals 1. It has been shown that AUC has lower variance and more reliability than other performance metrics (such as precision, recall, F-measure) [15]. Note that the metric used to measure the performance of the classifiers is completely independent from the metric in the TBFS algorithm. AUC is used both to select the most predictive subset of features in TBFS and to evaluate the classification models constructed using this set of features.

3) Experimental Results: During the experiments, ten runs of five-fold cross-validation were performed. First, we ranked the attributes using the eighteen rankers separately. Once the attributes are ranked, the top six attributes are selected to yield the final training data. After feature selection, we applied the classifier to the training datasets with the selected features, and then we used AUC to evaluate the performance of the classification model. In total, 18 rankers × 4 datasets × 10 runs × 5 folds = 3600 combinations of feature ranking techniques were employed, and correspondingly 3600 × 3 classifiers = 10800 classification models were built.

All the results are reported in Table VI through Table VIII. Note that each value presented in the tables is the average over the ten runs of five-fold cross-validation outcomes. From these tables, we can observe that although a given ranker might perform best in combination with one learner, this may not be true when other learners are used to evaluate models. For example, S2N performed best on average in terms of AUC when the NB and LR classifiers are used. However, this is not true when the MLP classifier is used; in that case, PO performed best. The results also demonstrate that although no particular ranker dominates the others, we can conclude that S2N, PO, and PRC are most often the best techniques, while PR, OR, and GR are rarely optimal. Our recent study [5] shows that the reduced feature subsets can have better or similar prediction performance compared to the complete set of attributes (original data set).

We also conducted a two-way ANalysis Of VAriance (ANOVA) F test ([16]) to statistically examine the various effects on the performances of the classification models. The two-way ANOVA test in this study includes two factors: the first represents 18 rankers, and the second represents the three learners. In this ANOVA test, the results from all four datasets were taken into account together. In this study, we also performed the multiple comparison tests using Tukey's honestly significant difference (HSD) criterion [16]. All tests of statistical significance utilize a significance level α of 5%. Both ANOVA and multiple comparison tests were implemented in

TABLE VI
CLASSIFICATION PERFORMANCE, NB

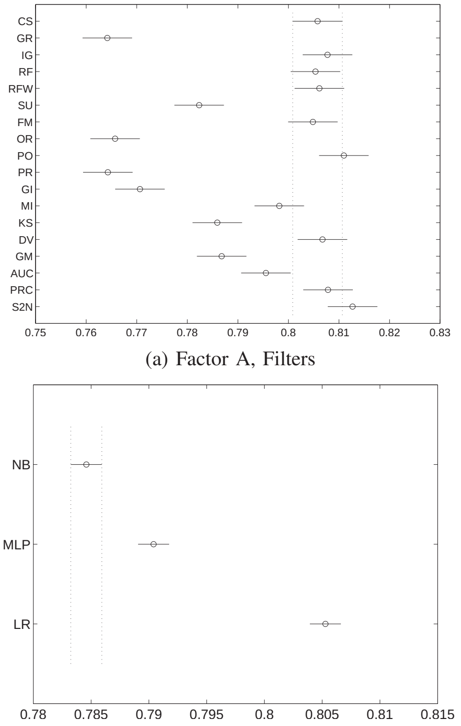
	SP1	SP2	SP3	SP4	Average
CS	0.7846	0.8108	0.8184	0.7696	0.7958
GR	0.7346	0.7613	0.7808	0.7519	0.7571
IG	0.7831	0.8081	0.8118	0.7794	0.7956
RF	0.7879	0.8053	0.8305	0.7731	0.7992
RFW	0.7882	0.8081	0.8190	0.7735	0.7972
SU	0.7865	0.7729	0.7882	0.7592	0.7767
FM	0.7822	0.8074	0.8176	0.7731	0.7951
OR	0.7405	0.8060	0.7181	0.7558	0.7551
PO	0.7891	0.8071	0.8141	0.8023	0.8031
PR	0.7345	0.7963	0.7179	0.7605	0.7523
GI	0.7341	0.7982	0.7678	0.6997	0.7500
MI	0.7739	0.8010	0.8119	0.7788	0.7914
KS	0.7722	0.7750	0.8125	0.7588	0.7796
DV	0.7820	0.8099	0.8163	0.7874	0.7989
GM	0.7716	0.7740	0.8165	0.7586	0.7802
AUC	0.7685	0.8072	0.7947	0.7683	0.7847
PRC	0.7885	0.8131	0.8120	0.7953	0.8022
S2N	0.7995	0.8142	0.8067	0.8129	0.8083

TABLE VII
CLASSIFICATION PERFORMANCE, MLP

	SP1	SP2	SP3	SP4	Average
CS	0.7943	0.8117	0.8126	0.7914	0.8025
GR	0.7475	0.7545	0.7688	0.7464	0.7543
IG	0.7926	0.8099	0.8209	0.8103	0.8084
RF	0.7948	0.8119	0.8191	0.7619	0.7969
RFW	0.7955	0.8139	0.8303	0.7598	0.7999
SU	0.7875	0.7847	0.7843	0.7504	0.7767
FM	0.7917	0.8127	0.8163	0.7924	0.8033
OR	0.7665	0.8058	0.7244	0.7550	0.7630
PO	0.7955	0.8133	0.8261	0.8010	0.8090
PR	0.7666	0.7970	0.7299	0.7576	0.7628
GI	0.7660	0.7963	0.7813	0.7495	0.7733
MI	0.7855	0.7945	0.8249	0.7865	0.7978
KS	0.7836	0.7749	0.8201	0.7607	0.7848
DV	0.7923	0.8095	0.8180	0.7959	0.8039
GM	0.7812	0.7779	0.8150	0.7701	0.7861
AUC	0.7754	0.8099	0.8260	0.7769	0.7970
PRC	0.7977	0.8151	0.8105	0.7908	0.8035
S2N	0.8047	0.8039	0.8108	0.7972	0.8042

TABLE VIII
CLASSIFICATION PERFORMANCE, LR

Ranker	SP1	SP2	SP3	SP4	Average
CS	0.8021	0.8229	0.8354	0.8153	0.8189
GR	0.7688	0.7935	0.7805	0.7816	0.7811
IG	0.8014	0.8176	0.8361	0.8216	0.8192
RF	0.8103	0.8221	0.8354	0.8118	0.8199
RFW	0.8091	0.8233	0.8387	0.8142	0.8213
SU	0.7993	0.7909	0.8040	0.7802	0.7936
FM	0.8023	0.8235	0.8298	0.8088	0.8161
OR	0.7816	0.8158	0.7422	0.7768	0.7791
PO	0.8041	0.8255	0.8334	0.8202	0.8208
PR	0.7784	0.8053	0.7467	0.7808	0.7778
GI	0.7787	0.8062	0.7918	0.7780	0.7887
MI	0.7934	0.7983	0.8354	0.7942	0.8053
KS	0.7902	0.7788	0.8338	0.7705	0.7933
DV	0.8020	0.8186	0.8323	0.8164	0.8173
GM	0.7886	0.7755	0.8327	0.7799	0.7942
AUC	0.7845	0.8154	0.8318	0.7882	0.8049
PRC	0.8041	0.8265	0.8282	0.8123	0.8178
S2N	0.8176	0.8279	0.8336	0.8231	0.8256



(a) Factor A, Filters

(b) Factor B, Learners

Fig. 1. Tukey's HSD, Classification

MATLAB.

The ANOVA table is omitted due to space restrictions; however, the p-values for both of the main factors were zero, indicating that classification performance was not the same for all groups in each of the main factors. The Tukey's HSD results are presented in Figure 1, displaying graphs with each group mean represented by a symbol ($\circ$) and the 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. The figure shows the following facts: for the classification models built with LLTS datasets, we can divide the 18 rankers into four groups, {GR, PR, OR, GI}, {SU, KS, GM}, {MI, AUC}, and {FM, RF, CS, RFW, DV, IG, PRC, PO, S2N}, with the clusters ordered by their performances from worst to best. The rankers from different groups performed significantly different, while the rankers from same group performed similarly (were not significantly different from one another). For the last group, S2N performs best. In addition, in terms of the learners, all three were significantly different from each other, with LR performing best and MLP and NB following in that order.

Comparing results in Section IV-C and Section IV-D, we can make the following observations: the clusters based on similarity are smaller and more numerous than the ones based on classification. This makes sense, because while rankers in the same similarity group are expected to perform similarly (and thus be in the same classification cluster), rankers sharing a classification cluster may have nothing in common and therefore be in separate similarity clusters. In addition, the S2N and RFW rankers both performed well (in the top-performing classification group) while having very distinct choices of features (very low similarity), suggesting that these are appropriate for selecting diverse but high-performing rankers.

V. Conclusion

Feature (software metric) selection plays an important role in the software engineering domain. This work investigates the similarity of eighteen filter-based feature ranking techniques on a real-world software project. Generally speaking, RF and RFW are similar to each other and are dissimilar to all other rankers. We also built classification models using NB, MLP, and LR on the smaller subsets of selected attributes. The experimental results demonstrate that the signal-to-noise and ReliefF-W rankers performed well on average while being dissimilar enough to maintain diversity. In addition, both similarity and classification performance can be used to build clusters, and these can be combined to help select the best feature rankers.

Future work may include experiments using additional datasets from other software engineering and non-software engineering domains, and experiments with other ranking techniques and classifiers for building classification models.

References

[1] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *Journal of Machine Learning Research*, vol. 3, pp. 1157–1182, March 2003.

[2] 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.

[3] I. Jeffery, D. Higgins, and A. Culhane, "Comparison and evaluation of methods for generating differentially expressed gene lists from microarray data," *BMC Bioinformatics*, vol. 7, no. 1, pp. 359+, July 2006.

[4] 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.

[5] 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*, vol. 41, no. 5, pp. 579–606, 2011.

[6] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd ed. Morgan Kaufmann, 2005.

[7] I. Kononenko, "Estimating attributes: Analysis and extensions of RELIEF," in *European Conference on Machine Learning*. Springer Verlag, 1994, pp. 171–182.

[8] M. A. Hall and L. A. Smith, "Feature selection for machine learning: Comparing a correlation-based filter approach to the wrapper," in *Proceedings of the Twelfth International Florida Artificial Intelligence Research Society Conference*, May 1999, pp. 235–239.

[9] M. Wasikowski and X. wen Chen, "Combating the small sample class imbalance problem using feature selection," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, pp. 1388–1400, 2010.

[10] G. Forman, "An extensive empirical study of feature selection metrics for text classification," *J. Mach. Learn. Res.*, vol. 3, pp. 1289–1305, 2003.

[11] T. M. Khoshgoftaar and N. Seliya, "Fault-prediction modeling for software quality estimation: Comparing commonly used techniques," *Empirical Software Engineering Journal*, vol. 8, no. 3, pp. 255–283, 2003.

[12] T. Fawcett, "An introduction to ROC analysis," *Pattern Recognition Letters*, vol. 27, no. 8, pp. 861–874, June 2006.

[13] 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.

[14] L. I. Kuncheva, "A stability index for feature selection," in *Proceedings of the 25th conference on Proceedings of the 25th IASTED International Multi-Conference: artificial intelligence and applications*. Anaheim, CA, USA: ACTA Press, 2007, pp. 390–395.

[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] 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.

Progressive Clustering with Learned Seeds: An Event Categorization System for Power Grid

Boyi Xie†, Rebecca J. Passonneau†, Haimonti Dutta†,
Jing-Yeu Miaw, Axinia Radeva, Ashish Tomar
Center for Computational Learning Systems
Columbia University
New York, USA 10027
Email: †{xie@cs,becky@cs,haimonti@ccls}.columbia.edu

Cynthia Rudin
MIT Sloan School of Management
Massachusetts Institute of Technology
Cambridge, USA 02139
Email: rudin@mit.edu

Abstract—**Advances in computational intelligence provide improved solutions to many challenging software engineering problems. Software has long been deployed for infrastructure management of utilities, such as the electric power grid. System intelligence is in increasing demand for system control and resource allocation. We present a model for electrical event categorization in a power grid system: Progressive Clustering with Learned Seeds (PCLS) – a learning method that provides stable and promising categorization results from a very small labeled data. It benefits from supervision but maximally allows patterns be discovered by the data itself. We find it effectively captures the dynamics of a real world system over time.**

I. Introduction

Advances in computational intelligence provide improved solutions to many challenging software engineering problems. Software has long been deployed for infrastructure management of utilities, such as the electric power grid or telecommunication systems. System intelligence is in increasing demand for system control and resource allocation. Our work applies machine learning to a problem for the low power electrical grid that directly services customers. Over the past half dozen years, we have worked closely with Consolidated Edison of New York, a major utility company in New York City, on a project to apply machine learning techniques to the secondary electrical grid. The results have been used to maintain the reliability of the secondary electrical grid.

The goal of our project is to develop interpretable models on a year-by-year basis to rank secondary structures (manholes and service boxes) with respect to their vulnerability to a serious event, such as fire or explosion. We use a supervised ranking algorithm, thus have a need for labeled data that indicates which structures are vulnerable in a given year for training our models. The main source of data for labeling the structures consists of Con Edison's Emergency Control System (ECS) trouble tickets. They document electrical events, such as interruptions of service, and engineers' efforts to redress any problems. The task we address in this paper is how we apply machine learning to the trouble tickets in order to sort them into those that document serious events on structures, and those that document non-serious events. Serious events result in positive labels on the implicated structures; non-serious events are included along with serious events in the representation of a structure's past history. Thus the ability to learn a good ranking model for structures depends on our ticket classification.

In this paper, we describe our approach, Progressive Clustering with Learned Seeds (PCLS) – a learning method adapted to this domain that provides stable and promising categorization results from a very small labeled data set. PCLS benefits from supervised learning but maximally allows patterns be discovered by the data itself. We found that it effectively captures the dynamics of a time-varied real world system.

In this context, the goal of our event categorization system is to classify ECS tickets with respect to whether the reported "trouble" is a) a serious event, b) a low-grade event (a minor interruption or disruption of service), or c) not relevant. The semantics of the three-way classification is somewhat subjective and contingent. Certain events are unequivocally serious, such as manhole explosions. However, many tickets pertain to events that can be serious or not, depending on a wide range of factors. Further, the language in the trouble tickets changes over time. From a small, expert-labeled sample for a single region, we initially hand-crafted a set of classification rules (see section III). We use these to initialize a clustering approach for the earlier years of data. To initialize clusters for later years, learned decision trees from the clusters help seed the clusters.

The issue of gradual shifts in the semantics of document classes is potentially a very general one, thus our approach could apply to many problems where results of previous supervised learning must be adapted to changing contexts. Section II presents related work, followed by motivation (section III). We describe the data sets and the general domain in Section IV. Section V introduces a new semi-supervised approach, Progressive Clustering with Learned Seeds (PCLS). Section VI reviews three learning methods and our evaluation procedure. Section VII presents results of our experiments. At last, section VIII briefly summarizes our contribution.

II. Related Work

Early work on incremental learning [1], [2] attempted to build learners that could distinguish between noise and change. While they deal with concept learning, they address the same general problem we face. Utgoff [2] presents an incremental decision tree algorithm that restructures the tree as needed for new instances. Training instances are retained in the tree to facilitate restructuring, thus constituting meta-knowledge, meaning knowledge distinct from what is learned, but which facilitates learning.

There has been much previous work on cluster seeding to address the limitation that iterative clustering techniques (e.g. K-Means and Expectation Maximization (EM)) are sensitive to the choice of initial starting points (seeds). The problem addressed is how to select seed points in the absence of prior knowledge. Kaufman and Rousseeuw [3] propose an elaborate mechanism: the first seed is the instance that is most central in the data; the rest of the representatives are selected by choosing instances that promise to be closer to more of the remaining instances. Pena et al. [4] empirically compare the four initialization methods for the K-Means algorithm and illustrate that the random and Kaufman initializations outperform the other two, since they make K-Means less dependent on the initial choice of seeds. In K-Means++ [5], the random starting points are chosen with specific probabilities: that is, a point p is chosen as a seed with probability proportional to p's contribution to the overall potential. Bradley and Fayyad [6] propose refining the initial seeds by taking into account the modes of the underlying distribution. This refined initial seed enables the iterative algorithm to converge to a better local minimum.

*CL*ustering through decision *Tree* construction (CLTrees) [7] is related to ours in their use of decision trees (supervised learning) for generation of clusters. They partition the data space into data and empty regions at various levels of details. Their method is fundamentally different from ours and do not capture the aspect of incremental learning over time.

III. Motivation

We have been working with Con Edison to develop a machine learning approach to predict serious events in secondary structures (manholes and service boxes). Our task is to produce for each borough, for a given year, a ranked list of the borough's structures with respect to their vulnerability to a serious event in the near future. The prediction problem is challenging. Only 0.1-5.0% of the tens of thousands of structures per borough experience a serious event each year, depending on borough, year and the definition of serious event. Causes of these rare events, if they can be detected, are often indirect (insulation breakdown), can depend on a complex of factors (number of cables per structure), and develop slowly. Evaluation consists of a blind test of a ranked list against the serious events that occur in the following year, with emphasis on the top of the ranked lists, which are used to prioritize Con Edison repair work.

To label the structures for supervised learning, we rely on the ticket classes described in the introduction. As described in [8], we developed a data mining, inference and learning framework to rank structures. It relies on a supervised bipartite ranking algorithm that emphasizes the top of a ranked list [9]. For any given year, a structure that is identified as the location of a problem in a serious event ticket gets a positive label; all other structures get negative labels. The feature descriptions of the labeled structures also depend heavily on the ticket classes: across boroughs, years and modeling methods, at least half the features in our ranking models represent how many serious or low-grade events a structure has experienced in the recent or remote past.

We had no a priori gold standard for classifying tickets. Based on the intuitions of two domain experts, we initially used a trouble type assigned to tickets by Con Edison dispatchers as the indicator of seriousness. Of the roughly two and a half dozen trouble types we use (the constituency varies somewhat across boroughs), two are unequivocally serious (for explosions and fires), and some are almost never serious (e.g., flickering lights). However, there is one category in particular (smoking manholes) that is both very large and can be serious or not. In previous work [10], we applied corpus annotation methods to elicit a definition by example of the three classes: serious event, low-grade event, and irrelevant. Two experts labeled a carefully designed sample of 171 tickets. That they achieved only modest interannotator agreement on the first pass (κ=0.49; [11]) indicates that the classes are rather subjective. Based on a second pass involving adjudication among the experts, we developed a rule-based method to classify tickets. We produced a small fixed set rules, along with a large fixed set of regular expressions, to capture generalizations we observed in the hand-labeled data. To test and refine the rules we applied them to large random samples, modifying them based on our judgments of their accuracy. As reported in [10] the rule-based classes improved the ranking results, particularly at the top of the list for the most vulnerable structures (one in every five structures was affected).

Development of the hand-crafted rules (HCR) required approximately 2,500 person hours. While they improved over the use of the assigned trouble type, our goal in the experiments reported here is to boost the improvement further, and to adapt the rules over time and regions. In particular, we need an approach that generalizes over time and space: the different boroughs have different infrastructure and histories, slightly different sublanguages (in the sense of [12]), and we use different subsets of trouble types as data. Rather than adapting the rules manually, which would be costly, we seek an automated method.

While the notions of *relevant* or *serious* events have some generality, the specific realization of the three ticket class changes from borough to borough, and from year to year. This can be illustrated by comparing the discriminative words over time. If we take the ticket classes produced by our hand-crafted rules and compare the list of words that discriminate the classes from year to year, we find that only about half the discriminative words overlap. For relevant versus non-relevant tickets, a comparison of the discriminative vocabulary for each successive pair of years from 2001 to 2005 shows that the average overlap in vocabulary is only 56.27% (sdev=0.05).

IV. Data Sources

We have been working with Con Edison Emergency Control System (ECS) tickets from three New York city boroughs: Manhattan, Brooklyn and Bronx. The experiments reported here pertain to Manhattan, which is our primary focus. We use tickets from 1996 through 2006, consisting of 61,730 tickets. The tickets are textual reports of secondary events, such as manhole fires, flickering lights in a building, and so on; they are generated when a customer or city worker calls the ECS line. The ECS tickets in our dataset range in length from 1 to 550 lines, with a similarly wide range of information content. ECS tickets can have multiple entries from different individuals, some of which is free text, some of which is automatically entered. These tickets exhibit the fragmentary language, lack of punctuation, acronyms and special symbols characteristic of trouble tickets from many arenas.

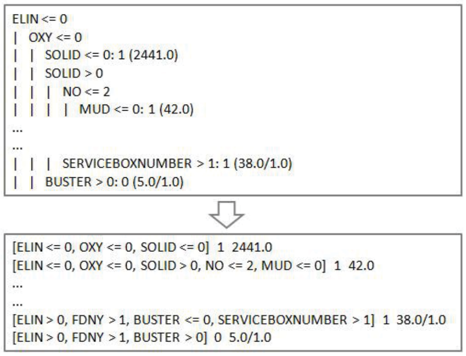
Structures are labeled with respect to a given year, thus we apply automated ticket classification or clustering on a year-by-year basis. We first classify tickets into relevant versus irrelevant events, then classify the tickets in the relevant class into serious versus low-grade events. The serious versus low-grade event classes are highly skewed. Of 61,730 tickets for ten years of Manhattan ECS (relevant trouble types only), about 43.67% represent relevant events, and only 15.92% of these are serious.

V. Progressive Clustering with Learned Seeds

Progressive Clustering with Learned Seeds is a method adopted after consideration of the tradeoffs of supervised and unsupervised approaches. We aimed to minimize the sensitivity of K-Means clustering to the initial seed points by biasing the initial centroids closer to the optimal ones using prior knowledge about the document classes.

Procedure 1 Progressive Clustering with Learned Seeds

```
1: Tree path extraction
2: Path scoring
3: Class contribution calculation
4: Seed points retrieval
5: K-Means clustering using retrieved seeds
```



```
ELIN <= 0
|  OXY <= 0
|  |  SOLID <= 0: 1 (2441.0)
|  |  SOLID > 0
|  |  |  NO <= 2
|  |  |  |  MUD <= 0: 1 (42.0)
...
...
|  |  |  SERVICEBOXNUMBER > 1: 1 (38.0/1.0)
|  |  BUSTER > 0: 0 (5.0/1.0)
```

```
[ELIN <= 0, OXY <= 0, SOLID <= 0]  1  2441.0
[ELIN <= 0, OXY <= 0, SOLID > 0, NO <= 2, MUD <= 0]  1  42.0
...
...
[ELIN > 0, FDNY > 1, BUSTER <= 0, SERVICEBOXNUMBER > 1]  1  38.0/1.0
[ELIN > 0, FDNY > 1, BUSTER > 0]  0  5.0/1.0
```

Fig. 1. Tree path extraction, which converts a decision tree model to a collection of paths

A. Tree path extraction

We train a decision tree model on the previous year's data, convert it into a set of paths as illustrated in Figure 1, and extract path attributes.

The following attributes are extracted for each path: (1) Length - the number of terms it contains; (2) Coverage - the number of instances addressed; (3) Accuracy - the rate of correctly classified instances; (4) Label - the class it predicts. We assign scores for each path using the first three attributes.

B. Path scoring

The scoring process formalizes the intuition that an optimal rule relies on fewer features, has greater coverage and higher accuracy. To score a path, we first compute homogeneous scores for all three attributes, in particular, to make them uniformly distributed within the range [0,1]. Subsequently, we use coefficients to weight each attribute and calculate a final score, also in [0,1].

For path i, we calculate its ScoreLength, ScoreCoverage and ScoreAccuracy separately using the following formulas:

$$ScoreLength_i = \frac{l_{max} - l_i}{l_{max} - l_{min}} \tag{1}$$

where l_i is the length of the path i, and l_{max} and l_{min} are the lengths of the longest and shortest paths. $ScoreLength_i \in [0, 1]$ is a uniform distribution.

$$ScoreCoverage_i = \frac{CoverageNorm_i}{CoverageNorm_{max}} \tag{2}$$

$$CoverageNorm_i = log(c_i + 1) \tag{3}$$

where c_i is the coverage of the path i, i.e. the number of instances related to path i in the training data. Because there is a big gap in coverage among a set of paths, e.g. from thousands to only a few, the logarithm function is used to smooth the data into a uniform distribution. By a further normalization, $ScoreCoverage_i \in [0, 1]$.

$$ScoreAccuracy_i = \frac{a_i}{a_{max}} \tag{4}$$

where a_i is the accuracy of the path i. $ScoreAccuracy_i \in [0, 1]$.

In summary, paths that are shorter, have more coverage and are more accurate score higher. After scoring each attribute, we calculate the final score for path i

$$\begin{aligned} Score_i = {} & \lambda_l \cdot ScoreLength_i \\ & + \lambda_c \cdot ScoreCoverage_i \\ & + \lambda_a \cdot ScoreAccuracy_i \end{aligned} \tag{5}$$

where $\lambda_l + \lambda_c + \lambda_a = 1$; λ_l, λ_c and λ_a are coefficients for the attribute length, coverage and accuracy respectively. They are used to weight each path attribute in the scoring function. Notice that, because each attribute score is normalized, the final score is also a uniform distribution and $Score_i \in [0, 1]$.

C. Class contribution calculation

Due to the data skew and the differential role of each class in the structure ranking problem, we next rank paths on a per class basis. We assign a quantity to each path representing its relative contribution to the class it predicts. The class contribution for each path exaggerates the discriminative power by an exponential function that increase the larger scores and decreases the smaller ones and is then normalized.

$$ScoreTransform_i = Base^{Score_i} \tag{6}$$

where $Score_i$ is the score for path i, $Base$ is a base constant of the exponential function.

$ScoreTransform_i$ can be regarded as the raw contribution of path i. If there are N_c paths belonging to class c, the sum of the scores $\sum_{i=1}^{N_c} ScoreTransform_i$ contributes the whole class. The ratio of $ScoreTransform_i$ and the sum can be regarded as the contribution of path i, and is normalized to [0,1]. Where c is the index for the class:

$$Contribution_{(i,c)} = \frac{ScoreTransform_i}{\sum_{i=1}^{N_c} ScoreTransform_i} \tag{7}$$

Paths are ranked for each class separately by their contribution. Seed points will be selected according to each class's path rank list.

D. Seed points retrieval

For seed points retrieval, we prefer to choose paths with a higher class contribution, and selects a reasonable number of data for each class from the decision tree model. Given a per class ranking of paths, we use the number of decision tree paths for each class to determine the proportion of seeds for each class, which reflects the relative importance of the information we have learned from the decision tree model.

$$NumOfSeeds_c = P \cdot N_{total} \cdot \frac{count(path_i, c)}{count(path_i)} \tag{8}$$

where N_{total} is the total number of instances in the data set that we want to cluster, P is the percentage of data to be seed points, $count(path_i, c)$ is the number of paths related to class c and $count(path_i)$ is the total number of paths extracted from the decision tree model.

When using a path from the tree trained in the prior year to retrieve instances in the current year, there may be no instances, or their may be more than needed. In the latter case, we randomly select the desired number.

E. Clustering with Learned Seed points

For the initial centroids, we hope to minimize

$$\Theta = \sum_{k=1}^{K} \|\vec{c}_{init_k} - \vec{c}_{opt_k}\| \tag{9}$$

where $\vec{c}_{init_k}$ is the initial centroids and $\vec{c}_{opt_k}$ is the optimal centroids.

Since the classification precision is usually high and stable (see Section VII), by utilizing the paths selected from the decision tree we can select the initial centroids to be more appropriately located in the overall space. Before the K-Means optimization procedure, we initialize the cluster centroid using the seed points we retrieved.

$$\vec{c}_{init_k} = \vec{\mu}_k = \frac{\sum_{n=1}^{N_k} \vec{x}_n}{N_k} \tag{10}$$

where $\vec{x}_n$ is the n^{th} instance selected by paths that belong to class k. N_k is the total number of instances that were found related to class k.

We select initial centroids $\vec{\mu}_k$ to seed the clusters, then apply K-Means.

VI. Methods

We seek an automated or semi-automated approach that can classify tickets for the structure ranking task, with adaptation to each borough and time frame. To reiterate, our goal is to improve upon the Hand-Crafted Rule (HCR), thus we use their output as a baseline. Then we compare three learning methods: (1) C4.5 Decision Tree (DT), (2) K-Means Clustering (KM), (3) Progressive Clustering with Learned Seeds (PCLS). In this section, we first introduce our data representation and feature selection method. Next we briefly contrast the strengths and weaknesses of DT and KM; PCLS was described in section V. Then we describe our evaluation method.

A. Data preprocessing

We use *bag-of-words* document representation, with feature selection to reduce dimensionality. There are an estimated 7,500 distinct unigrams in each year's data, not counting misspellings and word fragments; we use about 10% (750 terms) as features. Previous experiments with spelling normalization reduced the vocabulary by 40%, but had an inconsistent and modest impact. In the experiments reported here, we filter out line separators and other lines with little or no text.

Feature selection was the same for all three classification approaches and was always performed on data from prior year(s). We compared the performance of Bi-Normal Separation, Chi-Square, F-Measure and Information Gain [13] for feature selection. Information Gain exhibited the most stable and consistent performance across different boroughs, years and data representation formats (Boolean, TFIDF, TF). The results reported here all rely on Information Gain for feature selection, and absolute term frequency (TF) as the bag-of-words vector values.[1]

B. Baseline: Hand Crafted Rules

The hand-crafted rules rely on three types of information: other Con Edison databases indicating the voltage; global properties of the ticket such as length and ticket trouble type;

[1] Absolute TF performs better than normalized TF, presumably because it indirectly represents the length of the ticket, a factor in determining seriousness.

meta-data we assign to indicate signs of seriousness or type of work performed, based on pattern-matching for terms in the ticket. There is only one set of hand-crafted rules that was bootstrapped from a small labeled dataset and it is used for all year's data.

C. Decision Trees and Clustering

We used the Weka [14] implementation of the C4.5 decision tree [15] for an interpretable, supervised approach to our classification tasks. In general, the decision tree models exhibited good precision, but with poor recall on serious events, which had a negative effect on structure ranking in that too few structures were labeled as serious. Decision trees are relatively interpretable in comparison to other learning methods because the paths in the tree can be converted to rules for each class being learned. In contrast, the strengths of K-means clustering are speed, a lack of dependence on labeled training data, and high recall. The weaknesses are poor precision, and lack of robust performance due to the sensitivity to initial centroids.

D. Evaluation

To evaluate the performance of DT, KM and PCLS, we performed intrinsic and extrinsic evaluations [16]. The intrinsic evaluation is to compare the predicted event labels with labels generated by HCR, as measured by recall, precision and F-measure. In the context of our project, the event labeling is in the service of the structure ranking problem and has a crucial impact. We are therefore able to perform an extrinsic evaluation on the structure ranking task. To reiterate, events classified as serious in the year for training the ranking model determine which structures are labeled as vulnerable. Consequently, the extrinsic evaluation provides the most compelling evidence for the merit of the event categorization. Our extrinsic evaluation consists in generating a distinct set of structure labels and features for each event classification method, and comparing the ranked lists that the ranking model yields when relying on each event categorization method.

VII. Experimental Framework and Results

Our goal is to bootstrap from the rule-based method at some point in the past, then to rely solely on the automated methods from that point forward. We use PCLS, where we cluster the current year Y_i of tickets, seeding the clusters with seed points selected using the learned trees from the prior year Y_{i-1}, then in the subsequent year Y_{i+1}, apply the decision tree of Y_i for seeding clusters for year Y_{i+1}.

We report intrinsic and extrinsic evaluation results to compare ticket classes produced by HCR, C4.5 decision trees, KM and PCLS. Figure 2 schematically represents the experimental setup: bootstrap automated classification from HCR in 2001, then evaluate automated methods for ticket classification for 2002-2006. Intrinsic evaluation applies to each year. We report extrinsic results for two ranked lists, for the years 2006 and 2007: the ranked list trained on 2005 data is given a blind evaluation against 2006 data, and the ranked list trained on 2006 is evaluated against 2007 data. The ranking models are trained using ticket classes to label structures for the training year, and features based on ticket classes for all prior years.

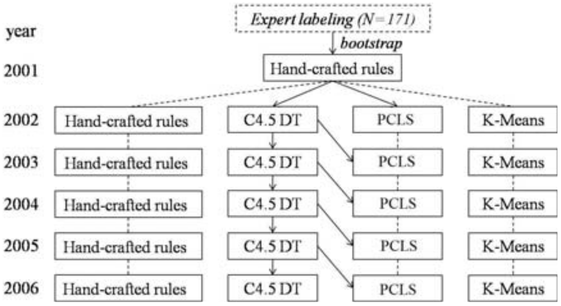


Fig. 2. Experiment setup. We compare Progressive Clustering with Learned Seeds with hand-crafted rules, C.45 decision tree, and K-Means methods.

A. Intrinsic results

For PCLS, results shown here use 50% of total instances as seed points to generate initial centroids, the weights of attributes are $\lambda_l = 0.1$, $\lambda_c = 0.2$ and $\lambda_a = 0.7$ for path scoring, and $Base = 2^{110}$ for class contribution. The results in Table I show that PCLS dramatically improves over K-means with random seeding for the serious versus low-grade event classification task; average F-measure for both classes is always larger for PCLS (significantly better for 4 out of 5 years). Compared with the C4.5 decision tree, PCLS exhibits a better recall and F-measure on the serious class (each is better for 4 out of 5 years, and far better for 2005 and 2006) while maintaining a competitive overall performance (in particular, better F-measure for 2004-2006). For the classification of relevant versus irrelevant events, we achieve similar results but do not present them here due to space limitations. Naive Bayes (NB) and SVM classifiers are also experimented. NB has worse performance. SVM achieves similar results but provides less human interpretable model than DT, such as the criterion of tree node. Thus, NB and SVM results are not reported here.

B. Extrinsic results

To rank structures, we use a bipartite ranking algorithm that focuses on the top of a ranked list [9]. For evaluation, we report AUC (Area Under the receiver operating characteristic Curve), DCG (Discounted Cumulative Gain, a weighted version of AUC that favors the top of the list) and PNorm scores, as recommended in [9]. A blind evaluation assesses how well the ranked list predicts structures that had particularly serious events in the year following the training year, where we have no access to the ECS tickets.

TABLE II
Extrinsic evaluation by AUC, DCG and PNorm measures. Higher score is preferred for AUC and DCG, and lower score is preferred for PNorm.

Measure	HCR	DT	KM	PCLS
AUC	0.524	0.516	0.540	**0.560**
DCG	30.789	30.625	31.447	**31.834**
PNorm	1.24E+09	1.29E+09	1.15E+09	**1.06E+09**

TABLE I
INTRINSIC EVALUATION OF SERIOUS VERSUS LOW-GRADE EVENT CATEGORIZATION. THE RESULTS FOR C4.5 DECISION TREE, K-MEANS CLUSTERING AND PCLS ARE COMPARED WITH THE LABELS FROM HAND-CRAFTED RULES.

year	class	C4.5 Decision Tree (DT)				K-Means Clustering (KM)				PCLS			
		Pre	Rec	F	Avg. F	Pre	Rec	F	Avg. F	Pre	Rec	F	Avg. F
2002	serious event	.701	.603	.648	.746	.024	.162	.042	.366	.610	.716	.659	**.718**
	low-grade event	.835	.852	.843		.757	.634	.690		.879	.698	.778	
2003	serious event	.675	.491	.569	.668	.256	.536	.347	.319	.328	.428	.371	**.497**
	low-grade event	.813	.725	.767		.908	.174	.292		.792	.514	.623	
2004	serious event	.633	.376	.472	.599	.251	.577	.350	.429	.688	.414	.517	**.606**
	low-grade event	.706	.745	.725		.631	.425	.508		.690	.701	.695	
2005	serious event	.536	.635	.581	.657	.624	.895	.735	.688	.560	.932	.700	**.696**
	low-grade event	.780	.692	.733		.731	.571	.641		.691	.694	.693	
2006	serious event	.603	.554	.577	.646	.008	.128	.016	.320	.692	.791	.738	**.712**
	low-grade event	.708	.722	.715		.547	.726	.624		.710	.664	.686	

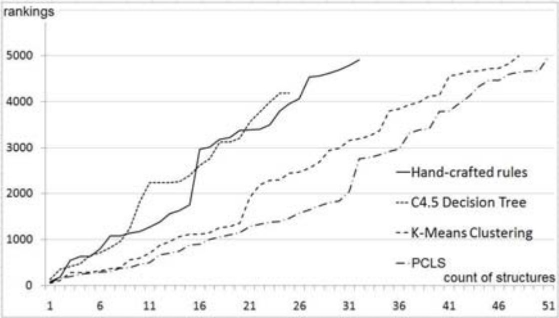


Fig. 3. Visualized results of extrinsic evaluation. It shows how the structures that actually have events are captured at the top 5,000 rank list. The vertical axis is the ranking, and the horizontal axis is a count of the number of structures. A lower curve is preferred, and the one more stretched to the right is preferred.

Table II summarizes the results of blind evaluation for Manhattan; larger scores for AUC and DCG, and lower for PNorm, correspond to superior performance. PCLS excels the other methods in all three measures. Figure 3 plots the vulnerable structures from the blind evaluation year (horizontal axis) against the top 5000 structures in the ranked lists from the four methods (vertical axis). As shown, PCLS outperforms the other methods in terms of all three measures (AUC, DCG, PNorm). Since the actual performance and scores are not linear correlated, even though PCLS is literally only 3.7% higher in AUC and 1.2% higher in DCG, the improvement is actually quite substantial. From Figure 3, in the top 5000 of the rank list, PCLS retrieves 51 vulnerable structures and C4.5 DT is 31, which is 64.5% improvement. When compared to KM, PCLS captures 3 more structures and the positions of these structures in the list are significantly ranked higher. Moreover, KM labels many more tickets as serious, as indicated by the very low precision and relatively high recall for this class (Table I), leading to much higher computational complexity of the structure ranking task for KM in contrast to PCLS.

VIII. CONCLUSION

We have presented an electrical event categorization task on a power grid application system. The characteristics of the categorization problem require an approach that can adapt existing knowledge about the data model over time.

We developed a semi-supervised learning method, Progressive Clustering with Learned Seeds, that suited the problem. Intrinsic evaluation displays the stability and consistency of knowledge preservation in accordance with a set of very limited labeled data. Extrinsic evaluation shows the superior actual performance on a blind test. Our problem engineering method can also be implemented and adapted to other domains providing an exemplary approach that uses computational intelligence technology for industrial applications.

REFERENCES

[1] J. C. Schlimmer and R. H. Granger, “Incremental learning from noisy data,” *Machine Learning*, vol. 1, no. 3, pp. 317–354, 1986.

[2] P. E. Utgoff, “Incremental induction of decision trees,” *Machine Learning*, vol. 4, no. 2, pp. 161–186, 1989.

[3] L. Kaufman and P. J. Rousseeuw, *Finding Groups in Data: An Introduction to Cluster Analysis*. Wiley, Canada, 1990.

[4] J. M. Pena, J. A. Lozano, and P. Larranaga, “An empirical comparison of four initialization methods for the K-means algorithm,” *Pattern Recognition Letters*, vol. 20, pp. 1027 – 1040, 1999.

[5] D. Arthur and S. Vassilvitskii, “K-means++: The advantages of careful seeding,” in *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms*, June 2007, pp. 1027 – 1035.

[6] P. S. Bradley and U. M. Fayyad, “Refining initial points for K-means clustering,” in *Proceedings of the 15th International Conference on Machine Learning (ICML)*, 1998, pp. 91 – 99.

[7] B. Liu, Y. Xia, and P. S. Yu, “Clustering through decision tree construction,” in *Proceedings of the 9th International Conference on Information and Knowledge Management (CIKM)*, McLean, VA, 2000.

[8] C. Rudin, R. J. Passonneau, A. Radeva, H. Dutta, S. Ierome, and D. Isaac, “A process for predicting manhole events in manhattan,” *Mach. Learn.*, vol. 80, no. 1, pp. 1–31, Jul. 2010.

[9] C. Rudin, “The P-Norm Push: A simple convex ranking algorithm that concentrates at the top of the list,” *Journal of Machine Learning Research*, vol. 10, pp. 2233–2271, Oct 2009.

[10] R. J. Passonneau, C. Rudin, A. Radeva, and Z. A. Liu, “Reducing noise in labels and features for a real world dataset: Application of nlp corpus annotation methods,” in *Proceedings of the 10th International Conference on Computational Linguistics and Intelligent Text Processing*, 2009.

[11] J. Cohen, “A coefficient of agreement for nominal scales,” *Educational and Psychological Measurement*, vol. 20, pp. 37–46, 1960.

[12] R. Kittredge, “Sublanguages,” *American Journal of Computational Linguistics*, pp. 79–84, 1982.

[13] G. Forman, “An extensive empirical study of feature selection metrics for text classification,” *Machine Learning Research*, pp. 1289–1305, 2003.

[14] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*. San Francisco, CA: Morgan Kaufmann, 2005.

[15] J. R. Quinlan, *C4.5: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann Publishers, 1993.

[16] J. R. Galliers and K. Sparck Jones, “Evaluating natural language processing systems,” Computer Laboratory, University of Cambridge, Tech. Rep. Technical Report 291, 1993.

Multi-Objective Optimization of Fuzzy Neural Networks for Software Modeling

Kuwen Li, Marek Z. Reformat, Witold Pedrycz
Electrical and Computer Engineering
University of Alberta
Edmonton, AB, Canada
{kuwen, reformat, wpedrycz}@ualberta.ca

Jinfeng Yu
College of Computer Science and Technology
Harbin, Heilongjiang Province
China, 150001

Abstract—**Software modeling is used to provide better understanding of features and attributes describing software artifacts. This knowledge is applied to improve practices leading to development of software that is easier to maintain and reuse.**

In the paper we use Fuzzy Neural Networks (FNNs) for modeling purposes. This type of models allow for extracting knowledge in the form of if-then rules. The process of constructing FNN models involves structural as well as parametric optimization. Additionally, software data representing different classes of artifacts is often unbalanced creating difficulties in building models that "cover" all classes evenly.

We propose an application of Multi-Objective Evolutionary Computing as a tool for constructing FNN models. We use combinations of different measures to represent quality of developed models from the point of view of their classification capabilities. We extract rules from the constructed models, and prune them to obtain clear and simple representations of software artifacts.

I. Introduction

Development of models of different objects and artifacts is one of the most popular approaches that allow us to gain knowledge about these objects and artifacts , and provide us with tractable approximations to reality. There is an abundance of definitions of the word 'model'. A common definition, expressed in the words of Neelamkavil [1], is very illustrative: "A model is a simplified representation of a system intended to enhance our ability to understand, predict and possibly control the behavior of the system."

Among all different models the ones that are of special interest are white-box models. These models allow us to "look inside" and find relations that exists between attributes and features describing considered objects and artifacts. Such an approach is used in software engineering where software objects are modeled in order to gain knowledge about relations between software metrics describing these objects and their quality attributes. Development of software object models is challenging due to imbalance of software data , i.e., uneven distribution of different classes of software objects. This means that developed models are not able to model classes of objects that are poorly represented in the available data. This also affects extraction of knowledge, i.e., the knowledge about these classes is incomplete and unreliable.

In this paper we address the problem of building software models based on imbalanced data via combining a number of methods and approaches: Fuzzy Neural Networks (FNNs) as data models with abstraction and knowledge extraction capabilities, Evolutionary Computing (EC) as an optimization tool with easiness of introduction of different objectives, Pareto approach as a multi-objective optimization method with the ability to "generate" a set of results satisfying different objectives, and different measures as representations of model quality: accuracy, specificity, sensitivity(recall), and precision.

II. Fuzzy Neural Network Model

The fuzzy OR/AND neurons [2] are the fundamental parts of our Fuzzy Neural Network model. The OR/AND neuron arranges AND and OR neurons in the way illustrated in Figure 1.

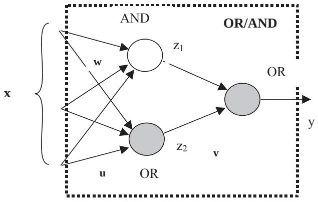


Figure 1. Architecture of OR/AND neuron

In essence, the outputs of two "input neurons" are aggregated (weighted) using the OR neuron located in the output layer:

$$z_1 = \text{AND}(\mathbf{x}; \mathbf{w}); \quad z_2 = \text{OR}(\mathbf{x}; \mathbf{u}); \quad y = \text{OR}(\mathbf{z}; \mathbf{v}).$$

The evident advantage of the OR/AND neurons resides with their significant interpretability capabilities. Note that if $v_1 = 1$ and $v_2 = 0$ we end up with a "pure" *and*-wise aggregation. The combination of $v_1 = 0$ and $v_2 = 1$ leads to the "pure" *or*-wise aggregation of the inputs. A whole spectrum of situations in-between the pure *and* and *or* aggregations is captured by the intermediate values of the connections of the OR neuron in the output layer.

We use the OR/AND neuron as the fundamental part of the fuzzy neural network model. However, we add connections from OR and AND neurons from the first layer of OR/AND neurons to OR neurons of the second layer of the OR/AND neurons. As the result we combine several OR/AND neurons to form a united model. In this model, each neuron in the first layer of each OR/AND neuron is connected to all OR neurons of the second layer.
A single OR/AND neuron represents one output of a modeled system. The final output of the model is determined via comparing the outputs of all OR/AND neurons. In the case of classification – this comparison identifies the "winning" class. The FNN structure dealing with 3 outputs – classes – is shown in Figure 2.

There are two ways to implement the class selection function. One is using *Maximum Output*, and another uses *Threshold Value*. For the *Maximum Output* we compare the output values of all OR neurons. The one with the maximum value is selected, and the output of the model is generated according to the index of this output. For the *Threshold Value*, a threshold value *t* is defined. The output values of each OR neuron is compared with *t*. If there is one and only one output with its value greater than or equal to *t*, the output of the model is the class represented by this OR neuron. Otherwise, the output of the model is not determined.

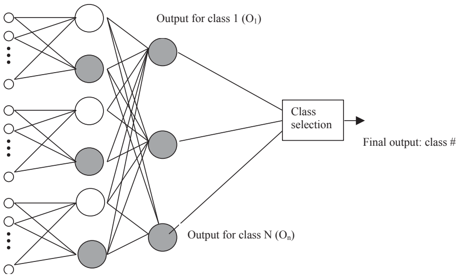


Figure 2. General structure of FNN

The most widely used algorithm to construct a FNN is the back propagation [3]. The objective used during that process is focused on minimization of the sum of squared errors between predicted and original outputs. The main advantage of this approach is its simplicity.

An interesting and important approach to construct FNNs has emerged with the introduction of evolutionary computing [4, 5]. Different evolutionary algorithms are applied for optimization-based learning of FNNs [6, 7]. They allow us to focus not only on parametric but also on structural optimization of the FNNs [8].

III. Multi-Objective Optimization

A multi-objective optimization problem is an optimization problem involving several criteria or model design objectives. Many real-world problems involve simultaneous optimization of several incommensurable and often competing requirements. While in a single-objective optimization the optimal solution is usually clearly defined, this does not hold for multi-objective optimization problems. If the objectives are opposing, then the problem is to find the best possible design or model that satisfies multiple objectives. In this case, instead of a single solution a set of alternative solutions is obtained.

Given a set of solutions to the problem, their partial ordering can be determined using the principle of dominance: a solution is clearly better than (dominating) another solution, if it is better or equal in all objectives, but at least better in one objective. Using this principle, the set of best solutions is found by removing all solutions that are dominated by at least one other solution. This set of indifferent solutions is referred to as a Pareto set.

Generally speaking, solving multi-objective problems is difficult. In an attempt to solve these problems in an acceptable timeframe, specific multi-objective evolutionary algorithms [9, 10, 11] have been developed.

IV. FNN Construction

A. Construction of Models with Skewed Data

The evolutionary-based optimization offers a great flexibility in exploiting different objective functions. This allows for building FNNs that satisfy very complex objective functions addressing different features of the network, as well as dealing with imbalanced data sets. The traditional techniques for constructing networks lead to the development of models that are "good" for biggest classes, but "ignore" smallest classes.

B. FNN and MOO: Concept

In order to construct models based on data sets with uneven distribution of classes, as well as to extract rules identifying relationships among data attributes representing these classes, a novel approach for model development is proposed.

In a nutshell, the main idea of the approach is to replace construction of a single FNN with development of a number of FNNs where each of them takes care of a single class. This process leads to finding the best model for each class. These single-class models are used to extract if-then rules describing the relationships among data attributes for each class. These rules are then combined and pruned to create a model that gives good classification rate and "treats all classes in an uniform way".

The development of FNNs for different classes is done simultaneously using an evolutionary-based technique – Pareto multi-objective evolutionary strategy. This multi-objective optimization targets all class at the same time. This leads to development of a number of different models. A subset of these models constitutes a Pareto surface. Each model that belongs to this subset is non-dominated – it means that none of the models from the whole set has better performance across all classes.

An additional advantage of application of an evolutionary-based optimization method is flexibility in selection of objectives that control a construction process of FNNs. In such case, the proposed approach allows for building FNNs that satisfy different requirements regarding classification rates.

C. FNN and MOO: Optimization Process

A multi-objective optimization process is used here in such a way that a classification rate for a single class is a single optimization objective. Therefore, a number of objectives is equal to the number of classes. As the result of such an optimization process, a set of solutions – data models – is obtained. These models constitute a Pareto surface.

Representation of Network Structures: An important aspect of application of evolutionary-based optimization to construct FNNs is to map a structure of an FNN to a chromosome string. Two aspects are important here: a selection of attributes that constitute inputs to AND/OR nodes, and an adjustment of connection weights. Therefore, a chromosome is built out of two segments:

- the segment that is "responsible" for selection of attributes that become inputs to the neurons, let's call it a *variable indexes part*;
- the segment that determines connection weights – let's call it a *connection weights part.*

Each position of the *variable indexes part* is an integer that identifies the attributes' index. The values of these integers are in the range from *0* to *n-1* (for *n*-dimensional data). The length of this segment is determined by the number of classes *c* (equal to the number of OR/ AND nodes) in the dataset, and the maximum number of inputs to a single OR/AND node – *i*. The overall length of the segment *variable indexes part* is *c*i*. The *connection weights part* is a string of floating point numbers. Each number represents a single connection weight and can be in the range from 0 to 1. The number of connection weights for FNN is *(2*i+2*c)*c*.

As it can be seen, the structure of the FNN is partially fixed – it contains so many OR/ AND nodes as different classes. It means each output of the FNN is associated with a single class.

Optimization Objective Functions: The objective of the optimization process is to construct FNNs that provide the best classification rates for a single class. This means that selection of the objective function that governs the optimization process is important.

For any two-category classification process, a confusion or error matrix can be built, Table I. This matrix summarizes classification capabilities of a model: values ***a*** (true positives) and ***d*** (true negatives) represent proper classifications, while ***b*** (false positive) and ***c*** (false negative) misclassifications.

Based on this matrix a number of different measures can be calculated. The measures that are used in our fitness functions are:

$$accuracy = \frac{a+d}{a+b+c+d}$$

that defines the ratio of all correctly classified data points to all data points,

$$sensitivity\,(recall) = \frac{a}{a+c}$$

that represents the percent of actual positive data points classified as positive data points,

$$specificity = \frac{d}{b+d}$$

that represents the percent of actual negative data points classified as negative data points, and

$$precision = \frac{a}{a+b}$$

that is the ratio of actual positive data points classified as positive to all data point classified as positive. It can be said that *sensitivity* and *specificity* are somehow reciprocal to each other. The *sensitivity* is for positive data points, while *specificity* is its equivalent for negative data points. Another observation is related to *sensitivity* (called also *recall*) and *precision*. Higher *sensitivity* means that almost all of the positive data points will be included in the classification results. However, at the same time, some negative data points can be predicted as positive ones, what leads to low values of *precision*.

Table I. Confusion matrix

actual **predicted**	POSITIVE	NEGATIVE
POSITIVE	***a***	***b***
NEGATIVE	***c***	***d***

Of course, *accuracy* is a very important measure – it means that the classifier is able to properly classify positive and negative data points. However, in the case of large imbalance in a number of data points that belong to each class, *accuracy* measure is not able to assure a high classification rate for each class. If 90 per cent of data points belong to the class *negative* and only 10 per cent to the class *positive*, then high *accuracy* can be achieved by correct classification of the class *negative* only. At the same time, this can lead to large misclassification (large ***b*** and ***c***) for the class *positive*. Therefore, there is a need to use some other measures that "take care" of large ***b*** and ***c.***

As the result of these investigations, fitness functions used here are constructed based on *accuracy*, *sensitivity*, *specificity* and *precision*. Two fitness functions are defined:

- one that uses a product of *sensitivity, specificity* and *accuracy* (*FF*$_{SSA}$)

$$F_{SSA} = Sensitivity * Specificity * Accuracy$$

- one that uses a product of *accuracy*, *recall* and *precision* – (*FF*$_{APR}$)

$$F_{APR} = Accuracy * Recall * Precision$$

Each of the fitness functions represents different way of suppressing ***b*** (false positives): F_{SSA} does it in the reference to ***d*** (true negatives), while F_{APR} in the reference to ***a*** (true negatives).

The presented above fitness functions are defined for a binary (two-category) classification problem. In order to focus on a single class, a special processing of confusion matrix is performed during evaluation of models. This processing means collapsing a multi-class confusion matrix into a two-class matrix. In such a case, the multi-class matrix is being transformed into a number (equal to the number of classes) of matrixes. Each of these matrixes is related to two classes: a class of interest – let say a class A, and the other class – let say a class non-A – obtained by fusion of all other classes. Using these two classes, all classification rates (accuracy, sensitivity, specificity and precision) are calculated.

As it has been explain earlier, each model is evaluated by a set of objectives – a set of fitness functions where each of them is related to a single class. Each model is "measured" by a set of fitness function:

$$\{F_{SSA_{class_1}}, F_{SSA_{class_2}}, \dots, F_{SSA_{class_c}}\}$$

or

$$\{F_{APR_{class_1}}, F_{APR_{class_2}}, \dots, F_{APR_{class_c}}\}$$

The proposed method is used to construct FNNs therefore there is one more thing that need to be explained – how the classification is determine based on the fuzzy output. In order to cope with that, a method with a *Threshold Value* (Section II) is used here. If the output of model associated with one of the classes is above the threshold then it indicates that a given data point belongs to this class. It if happens that two outputs are above the threshold – the model indicates that the point should belong to both classes. This result is seen as misclassification.

D. FNN and MOO: Pruning and Merging Processes

The Pareto-based optimization process generates a number of models that form a Pareto surface. Such an optimization process is repeated with different fitness functions and more models are generated. All these models are used in a process of knowledge – rules – extraction. Before the rules extraction takes place, a pruning process of models is performed. Its purpose is to reduce complexity of the models by decreasing a number of input variables to AND/OR neurons.

Model pruning is carried out by changing the thresholds for OR and AND neurons. For an AND neuron, the input variable has significant influence on the final output when the weight for this variable to the neuron is close to 0. For an OR neuron, the input has stronger effect on the output if the weight is closer to 1. So, for an AND neuron, the inputs with weight values close to 1 will not contribute much to the output. The same happens on an OR neuron if the weights values are close to 0. If we change the threshold values to close to1 for AND neurons, and close to 0 for OR neurons, we could find inputs that are not very useful. By changing threshold values, we could cut off some non-significant inputs and simply model, possibly with little change to the accuracy of the model. In our experiments, we change the threshold for AND and OR neurons separately, while ensuring the output accuracy will not drop more than 5%.

After we trim models for each class, we extract rules, select some of them, and combine the selected rules together into a single model for classification of all classes. In the selection process, we compare outputs of all rule sets (models[1]) constructed for each class. The rule set that provides the best classification rate for a single class is considered as the part of the final model. The process is performed for each class.

V. DATA DESCRIPTION

An experiment has been performed to generate software data required for illustration of the proposed approach to construct FNN models. In the experiment, objects of the system EvIdent [12] have been independently analyzed by three software architects and ranked according to their quality attributes: complexity, maintainability and usability. Quantitative software measures of these objects have been compiled.

A. Software System Description

EvIdent is a user-friendly, algorithm-rich, graphical environment for the detection, investigation, and visualization of novelty and/or change in a set of images as they evolve in time or frequency. It is written in Java and C++ and based on VIStA, an application-programming interface (API) developed at the National Research Council. 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.

B. Data Set

Only Java-based EvIdent/VIStA objects have been used here. For each of the 366 software objects, three programmers, named 'A', 'D' and 'V', were asked to independently rank objects' maintainability, complexity and usability from 1 (very poor) to 5 (very good). 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 the values assigned to each point by three programmers.

For the purpose of the experiments presented in the paper, we have combined rankings (objects) 1 and 2 into the *class1*, have renamed rank (objects) 3 into the *class2*, and have combined rankings (objects) 3 and 4 into the *class3*. Despite this, the objects are very unevenly distributed among the three classes. All three programmers have identified most of the objects as belonging to the *class3*. Using the "standard" approach to construct models – best overall classification rate – the models would "concentrate" on the *class3* ignoring the *class1* and *class2*. However, in the case of software engineering applications the most important are *class1* and *class2*, and rules generated for them. Objects of these classes need to recognized and better understood.

[1] In the case of an FNN the terms "rule set" and "model" are exchangeable. The FNN is de facto a set of rules.

VI. Experimental Results

The presented experiments are performed for a data set "generated" by the programmer V evaluating usability of objects. Among 366 objects the programmer V identified 30 objects as of low usability (*class1*), 74 objects as of medium usability (*class2*), and 262 objects as of high usability (*class3*). The continuous attributes of the original data set are fuzzified into three fuzzy sets (Low -Medium-High), and the discrete attributes are modified using 1-out-of-n technique. Constructed FNNs have: c=3, and i=5.

A. Model Construction

Before we start the description of experiments conducted according to the proposed approach, we show the confusion matrix obtained when only the accuracy criterion is used as the fitness function, Table II. The table includes the number of classifications for each class averaged over 10 splits with 5 experiments per split for the testing subset. The average number of samples for each class is: 12.0 for the *class1*, 28.9 for the *class2*, and 105.1 for the *class3*. It is easily observed that a single objective optimization does not provide decent results – all 50 models are doing well for the *class3*, but other two classes are not being classified at all. The multi-objective optimization is presented below.

Firstly, we present results of the experiments with SSA measures used for three objectives – one to maximizing classification capabilities for the *class1* (see the end of Section III.C for detailed explanation), another for the *class2*, and another for the *class3*, Table III. A sample Pareto surface for the optimization with SSA measures is shown in Figure 3. It can be observed that the models (each mark represents a single model) are relatively uniformly spread over the obtained Pareto surface.

Secondly, we have performed the experiments with APR measures, also for three objectives – classification of *class1*, classification of *class2*, and classification of *class3*. The obtained results are similar to the ones obtained with SSA. Due to space limitations we do not presented them here. A sample Pareto surface for the optimization with APR measures is shown in Figure 4.

Table II. Average values of correct classification for each class; the models have been constructed using accuracy as a single objective

Actual Predicted	***Class1***	***Class2***	***Class3***
Class1	***0.00 ± 0.00***	0.00 ± 0.00	0.00 ± 0.00
Class2	0.00 ± 0.00	***0.02 ± 0.14***	0.00 ± 0.00
Class3	10.50 ± 2.63	25.12 ± 4.40	***99.22 ± 7.79***

B. Knowledge Extraction

Models for each objective (classification rate for each individual class) are used as sources of knowledge expressed in the form of if-then rules. Before extraction of rules we performed pruning process (Section IV.D). The pruning thresholds have been set up in a way that accuracy of obtained models does not degrade more than 5%. It has resulted in the threshold values 0.65 for AND nodes, and 0.15 for OR nodes.

Table III. Average values of correct predictions for each class of data; the models have been constructed using SSA measures.

Results when model optimized for predicting Class1			
Actual Predicted	***Class1***	***Class2***	***Class3***
Class1	***6.70 ± 1.82***	6.22 ± 2.96	4.96 ± 2.38
Class2	0.40 ± 0.61	***1.70 ± 2.64***	5.62 ± 6.93
Class3	0.62 ± 0.73	2.50 ± 2.35	***40.16 ± 25.09***
Results when model optimized for predicting Class2			
Class1	***0.08 ± 0.27***	0.30 ± 0.93	1.06 ± 4.37
Class2	6.98 ± 2.85	***15.22 ± 3.49***	17.58 ± 5.70
Class3	0.30 ± 0.61	1.56 ± 1.69	***31.36 ± 25.99***
Results when model optimized for predicting Class3			
Class1	***1.18 ± 1.91***	1.04 ± 1.73	1.12 ± 2.18
Class2	2.20 ± 3.23	***3.86 ± 4.50***	2.10 ± 2.28
Class3	1.60 ± 1.07	8.40 ± 2.32	***84.44 ± 5.87***

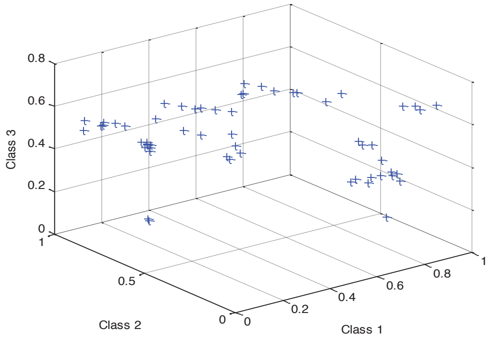


Figure 3. The Pareto 3D scatter plot for SSA objectives

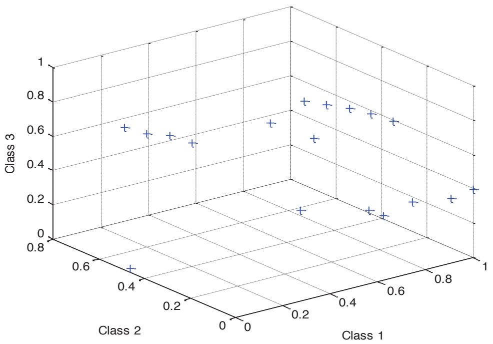


Figure 4. The Pareto 3D scatter plot for APR objectives

The rules obtained from the models generated with SSA measures are presented below (in similar way the models obtained with APR can be processed).

if {LOC[2] is Low}$_{0.239}$ and {MNL4 is High}$_{0.049}$ and {MAXP is 1}$_{0.332}$ and
{CONS is 3}$_{0.159}$ (0.450)
or {TYPE is 1}$_{0.300}$ or {LOC is Low}$_{0.996}$ or {MAXP is 1}$_{0.552}$ or
{CONS is 3}$_{0.225}$ (0.404)
or {WDC is Medium}$_{0.209}$ and {MIC is 2}$_{0.447}$ and {MIC is 4}$_{0.633}$ and
{OPER is Medium}$_{0.169}$ and {OPER is High}$_{0.407}$ (0.979)
or {MIC is 2}$_{0.187}$ or {MIC is 4}$_{0.570}$ or {OPER is High}$_{0.658}$ (0.846)
then USABILITY is **Low**

In the rule shown above, the subscript values represent weights associated with predicates and indicate their importance in the rule. The value in the (normal) brackets at the end of each antecedent represents importance of this antecedent and its contribution to the activation of the rule. This also applies to the rules for *Medium* and *High* usability.

if {CBO is Medium}$_{0.784}$ or {MNL1 is Low}$_{0.854}$ or
{HLOR is High} $_{0.923}$ or {MAXL is 4}$_{0.462}$ or
{PROM is Medium}$_{0.607}$ (0.466)
or {MIC is 3}$_{0.592}$ and {MEMB is High}$_{0.614}$ and {OVRM is 3}$_{0.102}$ (0.777)
or {ADEC is 5}$_{0.827}$ or {MIC is 3}$_{0.867}$ or {CONS is 3}$_{0.590}$ or
{MEMB is High}$_{0.915}$ (0.551)
then USABILITY is **Medium**

if {LOC is Low}$_{0.889}$ or {MAXL is 5}$_{0.703}$ or {MAXP is 10}$_{0.890}$ or
{MEMB is Low}$_{0.923}$ or {WMC2 is Medium}$_{0.440}$ (0.981)
or {HLPL is Low}$_{0.672}$ or {HLOR is High}$_{0.788}$ or {MIC is 1}$_{0.514}$ or
{OVRM is 15}$_{0.462}$ or {DEMV is High}$_{0.917}$ (0.379)
then USABILITY is **High**

C. Merged Models

So far, we have showed the classification results for rule sets (models) obtained using the multi-objective optimization with SSA and APR measures. The performance of these models is generally good, but our experiments with the combined models have shown that the classification rates are better when the sets of rules (models) "specialized" in a single class are put together.

In this section, we present results of models that have been put together with the sets of trimmed rules representing models created for single classes. A number of different combinations have been investigated. Each merged model has been "built" using three sets of rules: one set from a model optimized for *class1*, one from a model optimized for *class2*, and one from a model optimized for *class3*. Each of these models can be selected from any optimization experiment using APR and SSA measures. Among all possible combinations the best prediction results have been obtained for two merged models built with the following rules: for *class3* – rules obtained using SSA measures, for *class2* – rules obtained using APR measures, for the *class1* – rules obtained with APR (one option), and SSA (another option) measures, Table IV. The comparison with Table I shows a noteworthy improvement.

VI. CONCLUSIONS

In this paper we propose a new approach for the development of rule-based fuzzy classification systems in the presence of imbalanced datasets. The approach is focused on classification performance, as well as extraction of knowledge. Based on the experimental results with software engineering datasets, we have found that Pareto-based multi-objective (a single objective for each class) optimization approach is more effective than the traditional approach based on overall accuracy of classification. The method can be used for datasets with several classes. We can also conclude that combining models developed based on multiple classification measures (sensitivity, specificity, precision and accuracy) can offer better performance results than combining models developed for a single measure.

Table IV. Confusions matrixes for merged models.

Merged models: APR for class1, APR for clas2, SSA for class 3			
Actual Predicted	***Class1***	***Class2***	***Class3***
Class1	***7***	9	6
Class2	5	***19***	29
Class3	2	6	***63***
Merged models: SSA for class1, APR for clas2, SSA for class 3			
Class1	***9***	10	7
Class2	4	***18***	29
Class3	1	6	***62***

REFERENCES:

[1] F. Neelamkavil, Computer Simulation and Modeling, John Wiley & Sons Inc, 1987.

[2] K. Hirota, and W. Pedrycz, "OR/AND neuron in modeling fuzzy set connectives", IEEE Trans. on Fuzzy Systems, 2, 1994, pp. 151-161.

[3] W. Pedrycz, and F. Gomide, An Introduction to Fuzzy Sets; Analysis and Design, MIT Press, 1998.

[4] D.E. Goldberg, Genetic Algorithms in Search, Optimization& Machine Learning, Addison-Wesley, Reading, MA, 1989.

[5] Z. Michalewicz, Genetic Algorithms+ Data Structures =Evolution Programs, Springer, Berlin, 1994.

[6] A. Blanco, M. Delgado, and M. C. Pegalajar, "A real-coded genetic algorithm for training recurrent neural networks", Neural Networks, 14, 2001, pp. 93-105.

[7] R.P. Paiva, and A. Dourado, "Interpretability and learning in neuro-fuzzy systems", Fuzzy Sets and Systems, 147, 2004, pp. 17-38.

[8] W. Pedrycz, and M. Reformat, "Genetically Optimized Logic Models", Fuzzy Sets and Systems, 50(2), 2005, pp. 351-371.

[9] C. A. Coello, "A Comprehensive Survey of Evolutionary -Based Multiobjective Optimization Techniques", Knowledge and Information Systems, 1(3), 1999, pp. 269-308.

[10] D. A. Van Veldhuizen and G. B. Lamont, "Multiobjective Evolutionary Algorithms: Analyzing the State-of-the-Art", Evolutionary Computation, 8(2), 2000, pp. 125-147.

[11] E. Zitzler, M. Laumanns, L. Thiele, C. M. Fonseca, and V. Grunert da Fonseca, "Performance Assessment of Multiobjective Optimizers: An Analysis and Review", IEEE Transactions on Evolutionary Computation, 7(2), 2003, pp. 117-132.

[12] 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.

[2] Due to space constrains only some acronyms are explained: LOC–lines of code, MNL4–median length of method name, MAXL–max no of levels, CONS–no of constructors, TYPE–1=GUI, WDC–weighted no of decision based on nesting level, MIC–method invocation coupling, OPER–no of operations, CBO–coupling between objects, MNL1–max length of method name, MAXP–max no of parameters, HLOR–Halstead no of operators, PROM–% of protected members, MEMB–no of members, ADEC–mean no of decisions/method, OVRM–no of overridden methods.

Generating Performance Test Scripts and Scenarios Based on Abstract Intermediate Models

Leandro T. Costa, Ricardo M. Czekster, Flávio M. de Oliveira,
Elder M. Rodrigues, Maicon B. da Silveira, Avelino F. Zorzo
Faculty of Informatics (FACIN)
Pontifical Catholic University of Rio Grande do Sul (PUCRS)
Porto Alegre – RS – Brasil
Email: leandro.teodoro@acad.pucrs.br, ricardo.czekster@pucrs.br, flavio.oliveira@pucrs.br,
elder.rodrigues@pucrs.br, bernardino@acm.org, avelino.zorzo@pucrs.br

***Abstract*—Performance testing involves knowledge not only about the application to be tested, its usage, and the execution infrastructure; it also requires understanding of the performance test automation tools employed – scripting, monitoring and configuration details. Performance script generation is highly technology-dependent, with great variations from one test engine (the workload generator) to another. The scenarios, however, should not depend on that. In this paper, we present a technology-independent format for test scenarios and test cases, henceforth denominated *abstract intermediate models*, and the process for deriving specific test scripts from them. Hence, we can easily reuse test scenarios with different workload generators, allow the performance engineers to edit the scenarios before the script generation, and abstract details related to configuration of workload generators and monitors.** [1] [2]

I. Introduction

Performance testing is a highly specialized task. It involves knowledge about the application to be tested, its usage profile, and the infrastructure where it will operate. Furthermore, because it involves intensive test automation, performance testing requires understanding of the performance test automation tools that will be used. Hence, there is a bottleneck in productivity of performance engineering teams, due to the increasing complexity of the performance testing tools and the workload to generate the test case/scripts.

An approach that could be used to automatically generate performance test cases/scripts is Model-based Testing (MBT) [8]. MBT can be used not only to automatically derive performance test scenarios from the System Under Test (SUT) models, but also to automate the test execution. The SUT could be modelled under a wide range of modeling languages, such as state machine diagrams [9], Specification and Description Language (SDL) [12] and Unified Modeling Language (UML) [3]. Probably, the most widely used modeling language in the industry is UML. The UML provides a notation for modeling some important characteristics of applications, allowing the development of automatic tools for model verification, analysis and code generation. Performance is one of these characteristics; it is the object of the UML Performance Profile [10], which defines a standard way to represent performance information in a UML model.

In previous works [5] [15], we described our technique for deriving the test scenarios and the corresponding scripts from UML diagrams. During experiments, we learned that script generation is highly technology-dependent, with great variations from one test engine (called workload generator) to another. The scenarios, however, should not depend on that, because they should be reused to generate scripts for several workload generators. Thus, it would be wise to separate technology details from the process of generating test scenarios.

Information Technology (IT) companies, specially global companies, frequently use different workload generators, due to costs, marketing and/or different target execution platforms. The reuse of the scenarios, combined with automatic script generation, reduces overall testing effort. In addition to contribute to a technology migration, the test scenarios could have test information in a clear format and common to many technologies. Therefore, this information could be easily used to generate scripts to a wide range of performance tools, e.g. LoadRunner [11] and Visual Studio [13].

In this paper, we describe our approach to generate test scenarios and test cases, which could be used to derive scripts to a wide range of workload generator. In order to accomplish that, we defined a technology-independent format for test scenarios, which we call *abstract intermediate models*, and the process for deriving specific test scripts from them. Then, we can easily reuse test scenarios with different workload generators, also allowing the performance engineers to edit the scenarios before the script generation and abstract details related to configuration of workload generators and monitors.

The structure of this paper is organized as follows. Section II discusses related background. Section III explains the conversion from UML diagrams to test scenarios, discussing major trade-offs and advantages. Section IV provides an example demonstrating the abstract test scenarios that were derived automatically from the UML diagrams.

II. Background

Performance is a fundamental quality of software systems, affecting all underlying layers of systems. Therefore, in order

[1] Study developed by the Research Group of the PDTI 001/2012, financed by Dell Computers of Brazil Ltd. with resources of Law 8.248/91.

[2] The order of the authors is merely alphabetical.

to improve the software quality of applications, Software Performance Engineering (SPE) is used to describe and provide means to improve the performance through two distinct approaches: early cycles studies based on predictive models and late cycles inspections based on measurements [18]. One SPE activity is performance testing, responsible to perform tests for part or for the entire system, under normal load and/or stress load. Nowadays, there are some techniques, e.g., Record/Playback [14] and Model-based Testing, that are used by some tools to automate the performance test case generation and results analysis.

MBT [4] [8] [17] is a technique to test systems in which models are designed to create testing models suitable mapping a given reality a model is a useful way for into analyzing quality properties and performance attributes as it eases understanding and promotes divide-and-conquer rationale [1]. Thus, a SUT model can be constructed anticipating problems that may or may not arise in early design. However, the main usage of MBT has been directed to the test of functional aspects (defects) of software and only lately researchers are applying some of its techniques towards non-functional testing.

Therefore, lately, an increasing number of works [2] [5] [6] [15] discuss how to apply MBT and UML models to automatic generate performance test cases and scripts. Most of them apply MBT in conjunction with MARTE Profile [10], an extension of UML for modeling performance requirements. Its purpose is to provide modeling patterns with clear semantics, allowing automated performance model analysis. Based on the UML models, one can extract information for the tests and then analyse the performance counters (e.g. throughput, transactions per second and response time). Such information is distributed throughout several UML diagrams, e.g., use cases (UC), activity (AD) and sequence diagrams. An approach that depicts how this diagrams are used to represent the information of performance testing is presented by [6], in which a model designed in MARTE and composed by UML diagrams was tagged with specific properties such as probabilities on the incidence of use cases, response times for each activity, resource availability, etc.

III. UML Diagrams and Test Scenarios

The common first step adopted by the industry when testing applications for non-functional properties is to choose a given load engine among the various available tools. The decision on which tool to best test an application involves several factors such as familiarity (has been used before), pricing or target software platform.

However, every workload manager has its own peculiarities, configurations and tweaks as well as common operations to be performed when mimicking user behavior within the SUT. Thus, the test scenarios and scripts generated for some tool can not be reused to another.

Figure 1 shows our approach, in which the main idea is to convert high-level UML diagrams to an abstract intermediate model suitable for the derivation of abstract test scenarios and scripts to different workload manager. The main difference between our approach and classic MBT abstract test suites is the fact that we are interested in creating intermediate models to use for performance testing rather than functional testing. Another contribution is that we apply MBT to construct a performance abstract test scenario. Based on that, can be generated test scenarios and scripts for a wide range of workload manager.

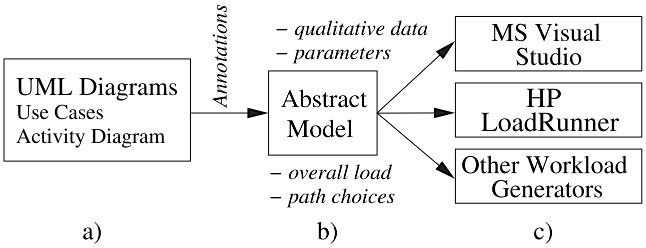


Fig. 1. Model transformation engine

A. Abstract Intermediate Models

Our focus in this paper is generate a technology-independent testing description from a MBT approach. Thus, we reuse the applications models to generate an abstract intermediate model that is a description of the test scenarios. It contains information needed to instantiate the performance scripts, such as loads, ramp-up time, total execution time etc., which can be derived from the model, while abstracting script details such as machine locations, API calls etc. An important issue when using MBT to generate performance test scenarios and scripts is define which UML diagram and which UML profile will be used to annotate the needed information.

B. Transformation Methodology

This section describes how to convert UML models (mostly UC and AD) into general purpose abstract intermediate models. Before describing the overall methodology, we discuss the UML annotation provided by other parties. Our approach relies on the assumption that the UML models are carefully annotated with high-quality information. We are trusting that all stakeholders are interested in the process of discovering the application's problems and bottlenecks and also that all involved personnel is committed to create comprehensive test scenarios, allowing further reliable analysis.

The process of generating test scenarios from annotated UML models must encompass a set of operations that must be performed prior to the test execution. Our methodology involves basically three steps:

1) UML Annotations Verification (Figure 1-a): We focus our attention to two broadly used diagrams present in UML: Use Cases (UC) and Activity Diagrams (AD). This step entails the evaluation if the models were annotated with information that can be used to the generation of the abstract model in the next step. Thus, to annotate these information on the UML models we defined some UML tags based on the UML profile to Schedulability Performance and Time (SPT) [18]: a) <<**TDtime**>>: limits the performance test duration(s) for the scenario(s); b) <<**TDpopulation**>>: maps the number of virtual users that populates the system; c) <<**TDhost**>>: describes the name of the host to connect to; d) <<**TDaction**>>: specifies an action that must be taken, e.g., the execution of a script; e) <<**TDparameters**>>: represents two information: name and value, that must be filled out to proceed, e.g., a login screen, or a search form; f) <<**TDprob**>>: indicates the probability to decide the next activity to be executed. It is used in the decision element model within the ADs or annotated in the association element model between actor and use case within the use cases diagram; g) <<**TDthinkTime**>>: defines the amount of time units each virtual user waits before taking another action. This tag is used by a large set of workload generators to mimic user behavior.

After this step, the UML models should satisfy the following conditions: a) every UC have at least one or several ADs counterparts; b) the AD is well-formed, i.e., contains an initial and an end state; c) test plan with information regarding the test itself, e.g., the type of test, the number of virtual users; d) every activity is annotated with context information, i.e., user form and query string parameters and user think time information.

2) Intermediate Model Generation (Figure 1-b): Our abstract intermediate model is designed to be extensible and modular, using hierarchical textual structures to represent activities readily available in UML diagrams or other similar representations of business processes (e.g. Business Process Model Notation (BPMN) [7]). In fact, it is straightforward to take any high-level business process representation and directly transform it to a test scenario, however, the test must be instantiated having a specific workload generator in mind.

This step is split in two abstract models: abstract test scenarios and abstract test cases. Each abstract models is a hierarchical structure that is used to map the AD to a textual abstract representation retaining the order of events and the parallel activities that must take place. This structure uses a sequential numbering convention with appended activities, e.g., 1, 1.1, 2.1 and so forth. Each activity is mapped to a number, here defined as an *activity number*. The numbered mapping is a visual aid to inspect sequential and parallel relations within the abstract intermediate model.

Note that for the abstract model to function according to our specification, it should contain only the fundamental information to instantiate different test scenarios. The abstract format suggested here is extensible and could be enhanced to contain more information regarding performance tests for a more complex test suite.

3) Workload Generator Instantiation (Figure 1-c): This step is tool-oriented because it generates specific test scenarios that must be created for each abstract intermediate model. We explain how this is performed in the following sections. Our approach shifts the concern on the performance test execution itself to the description of an abstract model that captures the needed test information looking up only to high-level (UML models). Next, we present how to apply our approach to generate scripts to two workload generators.

IV. Example of Use: Generating Scripts Based on Abstract Test Scenarios

This section describes an application scenario which our approach is applied to generate abstract models. For this purpose, we used the TPC-W benchmark [16] as an application example and develop a tool to generate automatically the abstract scenarios and then generate scripts for MS Visual Studio and LoadRunner. The TPC-W is a transactional web service benchmark that implements a benchmark and an e-commerce application that is used by the benchmark (or by any other workload generator).

To support the generation of scripts to test the TPC-W application, we developed an application to parse the information annoted in the UML models and generate the abstract models (following the guidelines described in Section III). This application is derived from the Software Product Line called PLeTs [5] [15].

A. TPC-W UML Diagrams

The first step to apply our approach (Section III) to test the TPC-W application is to annotate the TPC-W models. For this task, we have created several UML based tags to represent the performance information needed to generate the abstract intermediate models. As a starting point we have annotated three different use cases (shown in Figure 2): a) `Browser`: the users performs browsing interactions; b) `Registration`: the user fulfill a registration form; and c) `Shop`: the users performs purchase interactions.

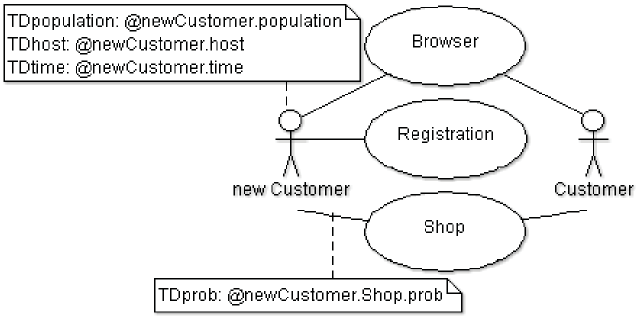

TDpopulation: @newCustomer.population
TDhost: @newCustomer.host
TDtime: @newCustomer.time
Browser
Registration
new Customer
Customer
Shop
TDprob: @newCustomer.Shop.prob

Fig. 2. TPC-W Use Case Diagram

Each actor present in the UC diagram contains information for a specific test scenario. Thus, we define two different test scenarios interactions for actors `Customer` and `New Customer`. The test scenario for `New Customer` is a set of interactions that will be performed over the set of use case elements (`Browser, Registration` and `Shop`). It is important to notice that each UC is related to a specific AD, e.g., the AD present in the Figure 3 is related to the `shop` use case (Figure 2). Basically, each AD represents the sequence of activities performed by an actor over the application. As depicted in Figure 3, the first activity is `Home Page`, which is the TPC-W home page. After that, the user could

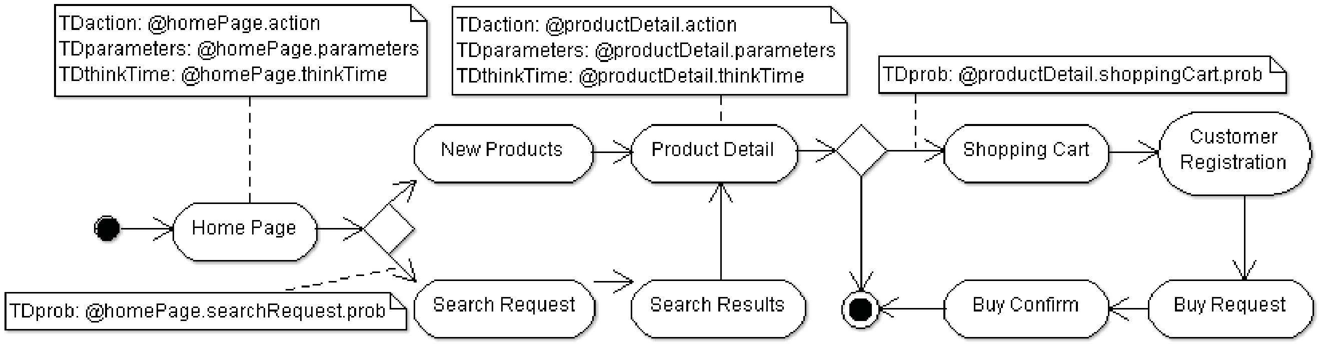

TDaction: @homePage.action
TDparameters: @homePage.parameters
TDthinkTime: @homePage.thinkTime
TDaction: @productDetail.action
TDparameters: @productDetail.parameters
TDthinkTime: @productDetail.thinkTime
TDprob: @productDetail.shoppingCart.prob
New Products
Product Detail
Shopping Cart
Customer Registration
Home Page
TDprob: @homePage.searchRequest.prob
Search Request
Search Results
Buy Confirm
Buy Request

Fig. 3. TPC-W Activity Diagram

perform the following actions: selects a specific book category (`New Products`) or performs a search for a particular book (`Search Request` and `Search Results`). The activity `Search Request` shows as a result a list of books to the user. Hence, when selecting a book from this list (`Search Results`), several information of the selected book are displayed to the user (`Product Detail`). After that, the user must perform one of the following activities: finish his access to the application or continue on the website and make a purchase (`Shopping Cart`). If the user decided for the latter option, the next step is related to the registration of customer (`Customer Registration`) in the application. Then, the user fills some information of purchase such as financial information (e.g. credit card number) and delivery date (`Buy Request`). The last step checks if all information is correct, then the purchase is confirmed (`Buy Confirm`) and finishes the access to the application. This diagram has also two decision elements in its flow, that represent the probability of executing different paths in the system, e.g., the tag `@homePage.searchRequest.prob` between activities `Home Page` and `Search Request` in the Figure 3.

As described in Section III, the UML diagrams can be initially annotated with seven tags in our approach. The UC shown in Figure 2, has four tags: `TDpopulation`, `TDhost`, `TDtime` and `TDprob`. Each one has its respective parameter, `@customer.population`, `@customer.host`, `@customer.time` and `@newCustomer.BrowsingMix.prob`. In relation to our AD we have also included four different tags and their respective parameter (see Figure 3).

B. Abstract Test Scenarios Generation

Once all the UML diagrams (see Figures 2 and 3) were annotated with performance informations, we apply our approach to generate abstract test scenarios for the TPC-W application. The creation of abstract test scenarios allows a later definition of a workload manager and its test script templates.

Basically, an abstract test scenario defines the amount of users that will interact with the SUT and also specifies the users behavior. The abstract test case contains the information regarding the necessary tasks to be performed by the user.

Figure 4 shows an example of an abstract test scenario generated for the actor `Customer`. The amount of abstract test scenarios generated based on a UC diagram is directly related to the amount of actors modeled in the UC model, e.g., for the TPC-W example there are two abstract test scenarios.

```
Abstract Test Scenario: Customer
## Test Setting
Virtual Users : <<TDpopulation: @customer.population>>
Host of SUT : <<TDhost: @customer.host>>
Test Duration : <<TDtime: @customer.time>>
## Test Cases Distribution:
<<TDprob: @customer.Shop.prob>>
1. Shop
1.1. Shop 1
1.2. Shop 2
1.3. Shop 3
1.4. Shop 4
<<TDprob: @customer.Browser.prob>>
2. Browser...
<<TDprob: @customer.Registration.prob>>
3. Registration
3.1. Registration 1...
3.4. Registration 4
```

Fig. 4. Abstract test scenario of the actor `Customer`

As presented in Section III, the abstract test scenario has the information related to the test context and the definition of the abstract test cases that must be instantiated, including the distribution of the number of virtual users for each abstract test case. Thus, our annotation approach is divided in two groups: 1) *Test Setting* – describes the general characteristics that are applied to the test context as a whole (extracted from the UC); 2) *Test Cases Distribution* – represents information specific to the abstract test cases generated from each AD. In order to accomplish that, each test case represent a user path in the SUT. It is important to notice that the header of each abstract test case contains probability information (see Figure 4) .

As show in Figure 5, the abstract test cases are built in a hierarchical approach, in which activities are listed and organized according to the dependency between the AD activities (represented by *activity number*). Figure 5 shows the abstract test case based on one test sequence derived from the AD (Figure 3). Furthermore, in the description of abstract test cases there is a variation of parameters added to each activity, which are composed by the name and the value of each tag, showing the flexibility of the configuration models. A parameter is the concatenation of two pieces of information: activity name and tag name, preceded by the delimiter `@`. Although, the tag `TDprob` has three pieces of information. The first of these information is the tag name preceded by the name of two UML elements. An example of that is presented in Figure 3 where the tag `TDprob @homePage.searchRequest.prob` is tagged in the UML association element between the UML decision node and the target activity (`Search Request`). The same notation is applied in the UC diagrams, where the tag is applied between the UC `Shop` and the actor `new Customer`.

```
#Abstract Test Case: Shop 3
1. Home Page
   <<TDmethod : @HomePage.method>>
   <<TDaction : @HomePage.action>>
   <<TDparameters : @HomePage.parameters>>
   <<TDthinkTime : @HomePage.thinkTime>>
2. New Products...
3. Product Detail...
4. Shopping Cart...
5. Customer Registration...
6. Buy Request...
7. Buy Confirm
   <<TDmethod : @BuyConfirm.method>>
   <<TDaction : @BuyConfirm.action>>
   <<TDparameters : @BuyConfirm.parameters>>
   <<TDthinkTime : @BuyConfirm.thinkTime>>
```

Fig. 5. Example of abstract test case generated from the `Shop` use case

It is important to notice that each tag parameter refers to a data file (Figure 6), that is automatically generated. Thus, when abstract test scenario and scripts are instantiated to a concrete test scenario and scripts for a specific workload generator, the tag parameter is replaced by a value extract from a file.

```
@BuyConfirm.method:"POST"
@BuyConfirm.action:"http://localhost/tpcw/buy_confirm"
@BuyConfirm.parameters:[$ADDRESS.CITY, $ADDRESS.STATE]
@BuyConfirm.thinkTime:5
```

Fig. 6. Example of data file containing some parameters

C. Test Scenarios and Scripts Instantiation

Based on the abstract test scenarios and test cases presented in Section IV-B, the next step is to generate concrete instances (scripts and scenarios). This is a technology dependent step, because the concrete scenarios and test cases are strongly related to a specific workload generator that will directly execute the test cases.

This is an important step on the automation of the performance testing, because it allows the flexibility of choice a workload generator or technology only in the execution stage of the test cases. However, it is necessary an advanced tool knowledgement to create scripts and scenarios. Therefore, to demonstrate how our approach could be valuable to a performance testing team, we presents how to generate test scenarios and scripts for two workload manager: Visual Studio (VS) and LoadRunner (LR). Basically, the VS and the LR structure its test scenarios and scripts in two files. One of them is a scenario file that is used to store the test configuration, workload profile distribution among test scripts and the performance counters that will be monitored by the tool. The other file is a script configuration file that is used to store the information about users' interaction with the application, including HTTP requests, as well as its parameters and transactions defined between requests. Figure 7 shows the VS scenarios file that was generated to test TPC-W. In this case, the `MaxUser` property corresponds to the parameter `@customer.population`. Another tag that has changed was the `RunConfiguration` with attribute `RunDuration` that is related to the tag `@customer.time`. The process to instrument a test scenario is based on a template. Hence, the further information within the scenario, that are not from the abstract test scenario are those standard information present in any test scenario generated by the workload generator.

```
<LoadTest ...>
  <Scenarios>
    <Scenario Name="Customer" ...>
      <ThinkProfile Value="0" Pattern="On"/>
      <LoadProfile Pattern="Step" InitialUsers="0"
           MaxUsers="50" StepUsers="10" StepDuration="0"
           StepRampTime="60"/>
      <BrowserMix>...</BrowserMix><TestMix>...</TestMix>
      <NetworkMix>...</NetworkMix>
    </Scenario>
  </Scenarios>
  <CounterSets>...</CounterSets>
  <RunConfigurations>
    <RunConfiguration RunDuration="7200" WarmupTime="300"
         TestIterations="100"...>...</RunConfiguration>
  </RunConfigurations>
</LoadTest>
```

Fig. 7. XML of test scenario generated for the Visual Studio (*.LoadTest)

Figure 8 presents a snippet of VS test scripts that was generated to test TPC-W. In turn, Figure 9 shows a snippet for LR. These test scripts were instantiated based on abstract test case presented in Figure 5. Basically, the test scripts are a set of several HTTP requests. Among the features correlated in the set of test artifacts, we highlight the following example: 1) `Tag Request` - in the VS the attribute `Url` and the attribute `web_submit_data` in the LoadRunner are related to the parameter `@BuyConfirm.action`; the VS attribute `ThinkTime` and the parameter `lr_think_time` LoadRunner are correlated to the parameter `@BuyConfirm.thinkTime`; 2) `Tag QueryStringParameter` - the VS and LoadRunner attributes `Name` and `Value` are related to the parameter `@BuyConfirm.parameters`.

V. Final considerations

The present work proposed a common format to define abstract intermediate models suitable for the instantiation of

```
<WebTest Name="Shop 3" ...>
 <Items>
  <TransactionTimer>...</TransactionTimer>
  <TransactionTimer Name="Buy Confirm">
   <Items>
    <Request Url="http://localhost:8080/tpcw/
          TPCW_buy_confirm_servlet" ThinkTime="5" ...>
     <QueryStringParameters>
       <QueryStringParameter Value="{{$ADDRESS.CITY}}"
            Name="CITY" .../>
       <QueryStringParameter Value="{{$ADDRESS.STATE}}"
            Name="STATE" .../>...
     </QueryStringParameters>
    </Request>
   </Items>
  </TransactionTimer>
 </Items>
 <ValidationRules>...</ValidationRules>
</WebTest>
```

Fig. 8. Test script generated for the Visual Studio

```
Action()
{
  ...
  lr_think_time(5);
  web_submit_data("buy_confirm.jsp",
  "Action=http://localhost:8080/tpcw/TPCW_buy_confirm_servlet
       ",
  "Method=POST",
  "RecContentType=text/html",
  "Referer=",
  "Mode=HTML",
  ITEMDATA,
    "Name=CITY", "Value={{$ADDRESS.CITY}}", ENDITEM,
    "Name=STATE", "Value={{$ADDRESS.STATE}}", ENDITEM,
  LAST);
  ...
}
```

Fig. 9. Test script generated for the LoadRunner

test scenarios for different workload managers. Throughout the paper we have discussed some important aspects on how to use annotated UML models to derive an intermediate textual format having the most important primitives that are needed to construct comprehensive test scenarios. We also show how to transform the abstract test scenarios in test script for two workload manager.

Our technique provides an indication that generating abstract models is a powerful means to derive effective technology-independent test scenarios. It is important to highlight that the creation of an abstract model for later definition of a test script and scenario using a chosen workload manager needs only to annotate a few selected data in the UML models. Translating UML models to this representation is also more comprehensible for end-users when they are tracking bugs or trying to understand the flow of operations for a functionality.

We envision several future works to consider following the present proposition. One could, for example, seamlessly translate a different UML model (e.g. Sequence Diagram) using our abstract model to generate scripts to some tool that is based on a different testing technique, e.g, structural testing. Another concern that has come to our attention is directed to the description of an abstract model to relate more architectural information in terms of the underline infrastructure of the SUT, for instance, the use of virtualized environments or cloud computing.

Acknowledgments. We thank CNPq/Brazil, CAPES/Brazil, INCT-SEC, and DELL for the support in the development of this work.

REFERENCES

[1] S. Balsamo, A. D. Marco, P. Inverardi, and M. Simeoni. Model-Based Performance Prediction in Software Development: A Survey. *IEEE Transactions on Software Engineering*, 30:295–310, May 2004.

[2] C. Barna, M. Litoiu, and H. Ghanbari. Model-based performance testing (nier track). In *Proceedings of the 33rd International Conference on Software Engineering*, pages 872–875, New York, NY, USA, 2011. ACM.

[3] G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language User Guide (2nd Edition)*. Addison-Wesley Professional, 2005.

[4] S. R. Dalal, A. Jain, N. Karunanithi, J. M. Leaton, C. M. Lott, G. C. Patton, and B. M. Horowitz. Model-based testing in practice. In *Proceedings of the 21st International Conference on Software engineering*, pages 285–294, New York, NY, USA, 1999. ACM.

[5] E. de M. Rodrigues, L. D. Viccari, and A. F. Zorzo. Plets-test automation using software product lines and model based testing. In *22th International Conference on Software Engineering and Knowledge Engineering (SEKE)*, pages 483–488, 2010.

[6] S. Demathieu, F. Thomas, C. Andre, S. Gerard, and F. Terrier. First experiments using the uml profile for marte. In *Object Oriented Real-Time Distributed Computing (ISORC), 2008 11th IEEE International Symposium on*, pages 50–57, may 2008.

[7] R. M. Dijkman, M. Dumas, and C. Ouyang. Semantics and analysis of business process models in BPMN. *Information and Software Technology*, 50(12):1281–1294, 2008.

[8] I. K. El-Far and J. A. Whittaker. *Model-based Software Testing*. Wiley, New York, 2001.

[9] R. Ferreira, J. Faria, and A. Paiva. Test Coverage Analysis of UML State Machines. In *Proceedings of the 3rd International Conference on Software Testing, Verification, and Validation Workshops*, pages 284 –289, april 2010.

[10] O. M. Group. UML Profile for Modeling and Analysis of Real-Time and Embedded Systems (MARTE). *MARTE specification version 1.0. OMG, 2009. OMG document number formal/2009-11-02.*, 2009.

[11] Hewlett Packard - HP. *Software* HP LoadRunner, Sep. 2010. URL: https://h10078.www1.hp.com/cda/hpms/display/main/hpms_content.jsp?zn=bto&cp=1-11-126-17\^8_4000_100.

[12] A. Kerbrat, T. Jéron, and R. Groz. Automated test generation from sdl specifications. In *Proceedings of the 6th SDL Forum*, pages 135–152, 1999.

[13] J. Levinson. *Software Testing With Visual Studio 2010*. Pearson Education, 2011.

[14] G. Meszaros. Agile regression testing using record & playback. In *Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 353–360, New York, NY, USA, 2003. ACM.

[15] M. B. Silveira, E. M. Rodrigues, A. F. Zorzo, L. T. Costa, H. V. Vieira, and F. M. Oliveira. Generation of Scripts for Performance Testing Based on UML Models. In *23rd International Conference on Software Engineering and Knowledge Engineering (SEKE)*, pages 1–6, 2011.

[16] TPC-W Org. Benchmark TPC-W, Feb. 2012. URL: http://http://www.tpc.org/tpcw.

[17] M. Utting and B. Legeard. *Practical Model-Based Testing: A Tools Approach*. Morgan Kaufmann, San Francisco, 2006.

[18] C. Woodside and D. Petriu. Capabilities of the UML Profile for Schedulability Performance and Time (SPT). In *Workshop SIVOES-SPT RTAS'2004*, 2004.

A Catalog of Patterns for Concept Lattice Interpretation in Software Reengineering

Muhammad U.Bhatti*, Nicolas Anquetil*, Marianne Huchard†, and Stéphane Ducasse*
*RMoD Project-Team INRIA - Lille Nord Europe USTL - CNRS UMR 8022, Lille, France
Email: {firstname.lastname}@inria.fr
†LIRMM, CNRS and Université de Montpellier-2, Montpellier, cedex 5, France
huchard@lirmm.fr

Abstract—**Formal Concept Analysis (FCA) provides an important approach in software reengineering for software understanding, design anomalies detection and correction. However, FCA-based approaches have two problems: (i) they produce lattices that must be interpreted by the user according to his/her understanding of the technique and different elements of the graph; and, (ii) the lattice can rapidly become so big that one is overwhelmed by the mass of information and possibilities. In this paper, we present a catalogue of important patterns in concept lattices, which can allow automating the task of lattice interpretation. The approach helps the reengineer to concentrate on the task of reengineering rather than understanding a complex lattice. We provide interpretation of these patterns in a generalized manner and illustrate them on various contexts constructed from program information of different open-source systems. We also present a tool that allows automated extraction of the patterns from concept lattices.**

I. Introduction

Formal Concept Analysis (FCA) is a mathematical technique to discover significant groupings of objects having similar attributes [14]. FCA can be applied to program entities, which helps in generating high-level views of a program for program understanding and identifying and correcting some anomalies in software systems. For example, FCA is used to identify objects in procedural code by looking at functions accessing global variables [26], [9]. Equally, it is applied to object-oriented systems by analyzing software components (*e.g.,*, classes, attributes or methods) and their relationships (*e.g.,*, belongs-to, uses or defines). Some studies aim at understanding object-oriented systems [3], [8], [23], others use program information for reengineering classes [28], [29], [5]. The extracted program information is used to construct concept lattices that are presented to the reengineers for analysis. The reengineer then explores a lattice to uncover important information.

The problem with the FCA techniques is that they produce concept lattices that are not readily comprehensible for developers who need to spend a good amount of effort to interpret these lattices. A possible solution is to break the information being fed into FCA into small chunks [5]. However, the approach is not scalable as the chunks need to be individually analyzed, which requires generating several lattices. Another issue is that concept lattices extract many more concepts than the number of objects they are given as input [1], thus adding complexity over the semantic complexity of the lattice.

The objective of the paper is to reduce concept lattices to a few interesting node and subgraph patterns such that the user does not have to interpret each node and its relationships nor to analyze the entire lattice in detail. We don't pretend that all useful findings are captured by these patterns, but they simplify the task of lattice analysis by proposing some accepted semantics. We look for the lattice patterns in an automated way, which could open the path for semi-automated reengineering actions.

In this paper, we define the patterns that proved to be useful in many cases of FCA applied to software engineering domain. We provide an interpretation in a generalized way, explaining their interest. Later, we exemplify them using different information from various open-source systems. These patterns are implemented in a prototype and we give some figures on the reduction in information to process for a concrete example.

This paper is organized as follows: Section II presents existing work that uses FCA in software reengineering and motivates our approach. Section III presents nodes and subgraphs patterns. Section IV proposes a validation on open-source systems and Section V describes the prototype for pattern identification. Section VI concludes the paper.

II. Motivating a generic interpretation tool for concept lattices

In this section, we present different studies that have been performed to understand and restructure software systems through FCA. Then we provide motivation for our proposed approach.

A. Existing work

a) Module and Object Identification with FCA: Sahraoui *et al.* [26] present a method that extracts objects from procedural code using FCA. Important concepts are looked for in the resultant lattices using heuristics. Another approach compares the object identification capability of clustering and FCA techniques [9]. A few heuristics are described to interpret the concepts of the lattice. An approach to transform a COBOL legacy system to a distributed component system is proposed by Canfora *et al.* [6]. Siff and Reps explore the relationship between program variables and procedures through FCA for restructuring of C programs [27].

b) Object-Oriented Reengineering and FCA: Godin *et al.* [15] proposed in 1993 to analyze classes by their member methods to evaluate and refactor OO class hierarchies using FCA. Dicky *et al.* [10] define an incremental algorithm and study the method specialization in the FCA framework. Falleri *et al.* [13] compare several FCA configurations [7]. Leblanc *et al.* [17] apply FCA to extract Java interface hierarchies from Java classes.

Snelting and Tip [28] present a framework for detecting and remediating design problems in a class hierarchy; it is used in [29] for refactoring Java class hierarchies. Arévalo, Ducasse and Nierstrasz [3] use FCA to understand how methods and classes in an object-oriented inheritance hierarchy are coupled by means of the inheritance and interfaces relationships. Lienhard, Ducasse and Arévalo [21] use FCA to identify traits in inheritance hierarchies. Dekel and Gil [8] use FCA to visualize the structure of Java classes and to select an effective order for reading the methods. In [5] a tool-assisted technique is presented to identify useful abstractions and class hierarchies in procedural object-oriented code.

FCA is used to detect the presence of bad smells and design patterns [2], and to suggest appropriate refactorings to correct certain design defects [24]. FCA is also used to understand a system's source code for relevant concepts of interest [23] and to cluster words for feature location [25].

FCA is also used to examine information generated from program execution to reconstruct control flow graphs [4] and feature location [12]. FCA is used in aspect mining [20], detection of design-level class attributes from code [30], and searching for code patterns and their violations [22].

c) FCA Filtering: Some studies, applying FCA to software reengineering, have suggested concept filtering techniques to resolve the problem of lattice complexity. The notion of concept partitions is used to filter the concepts that are not interesting for creating modules [27]. Mens *et al.* filter concepts according to some properties[1] of the concepts in the lattices constructed by their approach [23]. Arévalo *et al.* describe a few heuristics to reduce the search for concepts that describe important class hierarchy schemas [2]. Joshi and Joshi [19] provide a few patterns that may emerge when context is based upon methods and attributes of a class.

There is a plethora of FCA-based techniques that aim to analyze software systems for reengineering purposes. The diversity of these approaches shows the interest in FCA as a tool to certain kinds of software analysis. Some approaches apply context-specific heuristics for filtering non-interesting concepts. However, these heuristics remain tied to a specific context and cannot be generalized. Therefore, there is a need to define a unifying framework for lattice interpretation. This paper goes in that direction by describing a few patterns of nodes and subgraphs in concept lattices.

B. Synthetic view of FCA in Software Reengineering

One of the strengths of FCA for program comprehension and software reengineering is the wide range of contexts that can be used. For each different context, the method provides different insights into reengineering opportunities.

OO systems consist of different entities such as packages, classes, methods, attributes, or variables. In Table I we summarized the main entities and relationships that could be used as context (other ones could be invented). In the table, rows are formal objects, and columns formal attributes. In each cell of the upper half, a blank means there is no possible relationship (we found no interesting relationship to link a method formal object to packages formal attributes), U stands for use (a method uses another method by calling it, or a class uses another class through inheritance), and C stands for contains (or declare, a class contains an identifier, a method contains a variable). The relationships in boldface (upper half) denote contexts that we found already studied in the literature. In that case, a representative reference is given in the lower half of the table. The "×"s in the lower half denote possible contexts that we identified and for which we know of no prior published research. They represent unexplored areas of FCA and software reengineering research.

TABLE I
MAIN POSSIBLE CONTEXTS FOR FCA; FORMAL OBJECTS ARE IN ROWS, FORMAL ATTRIBUTES ARE IN COLUMNS, RELATIONSHIPS ARE USE AND CONTAIN; IN LOWER HALF, REFERENCES INDICATE THE CONTEXTS THAT WERE ALREADY STUDIED (ALSO IN BOLD FACE IN UPPER HALF) AND "×"S MARK POSSIBLE CONTEXTS THAT HAVE NOT BEEN EXPLORED YET

		Formal attributes				
		pckg	class	meth.	att.	var.
Formal objects	pckg	C,U	C,U	C,U	C,U	C,U
	class	U	C,**U**	**C**,U	**C**,U	C,U
	meth.		U	**U**	**U**	C,U
	att.		U			
	var.		U	**U**	**U**	U
	pckg	×	×	×	×	×
	class	×	[5], [32]	[21], [15] [18]	[13]	×
	meth.		×	[3], [5] [22]	[3], [8] [30]	×
	att.		×			
	var.		×	[28]	[28]	×

Table I considers only the main formal objects and formal attributes in software reengineering for OO systems. A comprehensive survey of the contexts studied in software engineering is provided by Tilley *et al.* [31].

C. Concept lattice interpretation

FCA is a flexible technique that may be used on a wide range of contexts. The number of unexplored contexts, even for the simple enumeration of possibilities proposed in the Section II-B, gives an idea of its potential. However before it gains larger use, we believe two problems need to be solved.

First most of the existing approaches, leave the analysis work to the user. Concept lattices are complex abstract constructs that may prove difficult to interpret. Sahraoui *et al.* [26] recognized this problem and proposed a tool to help analyzing the lattices.

Another issue with FCA is the size of the lattices they produce [1]. This again points toward the need to provide an

[1] size of the concept's "intent" and "extent".

assisted solution for lattice analysis. Some filtering techniques exist in the literature to remove unwanted concepts from lattices. However, these techniques are dependent upon their contexts because these are geared towards unrevealing specific code patterns. The solution of decomposing a context into smaller chunks requires a lot of effort to generate lattices representing each chunk. A natural solution lies in automating the interpretation of lattices. This automation will be more useful if it can be applied indifferently to lattices built on any formal context. The user would not be required to analyze each node in the lattice; he will search for patterns in the lattices and useful nodes will be identified without spending too much time on the lattice. Moreover, an automated technique can extract these patterns without requiring the user to understand the complexities of FCA. We hope to define a dictionary of such patterns that along with a definition of the meaning of a concept for each possible context, could propose to users an interpretation for any lattice built from a known context.

In the next section, we present a catalogue of patterns that represent interesting constructs in concept lattices from the software engineer point of view. These patterns help the user in two ways. First, they help to reduce the work of understanding a complex lattice for interpreting a few node and graph patterns. Hence, the work is reduced to look for these patterns and understand their interpretation. Second, because we consider generic subgraph patterns, a tool can be built to extract these patterns from the lattices, which will greatly simplify the work of analyzing these lattices.

III. Patterns in Concept Lattices

In this section, we present concept lattice patterns intended to help software engineers analyze the result of FCA. These patterns are sufficiently generic to be applied to a large set of possible contexts.

A. Nodes in Concept Lattices

The patterns we will present depend on typical arrangements of nodes and vertices, but also on the specific nodes that concept lattices may/should contain. Therefore, before going to the patterns, we will take a look at the four types of nodes that a concept lattice may contain. In the following, we borrow from the Conexp tool the way these nodes are displayed. The tool displays each node as a circle. A black lower semi-circle in the node indicates the presence of formal objects introduced by the node (*i.e.,* that no sub-concept has). A grey upper semi-circle in the node indicates the presence of formal attributes introduced by the node (*i.e.,* that no super-concept has). Four types of nodes are possible :

- **Full (black and grey):** The node introduces formal attributes that its super-concepts don't have and has formal objects that its sub-concepts don't have.
- **Grey:** The node introduces formal attributes that its super-concepts don't have, but all its formal objects (if it has any) appear in a sub-concept. Such node must have more than one direct sub-concept.
- **Black:** The node has formal objects of its own (that no sub-concept has), but "inherits" all its formal attributes (if it has any) from its super-concepts. Such node must have more than one direct super-concept.
- **Empty:** The node has no formal attribute or object of its own. It must have more than one direct super-concept and more than one direct sub-concept.

B. A Catalogue of Patterns

We now describe some subgraph patterns that we identified as useful to help analyze a concept lattice. The criteria that we used to choose these patterns are:

- They have a clear meaning that can be interpreted for any formal context; Hence, the patterns are *generic* enough to extract meaningful information;
- They represent important groups of formal objects and formal attributes, *useful* for the user;
- They produce very few false positives: If the patterns identify too many false-positives in the results, the user is required to look at the lattice to filter the false-positives. Hence, the effectiveness of the patterns would be lost. One can *rely on* the patterns and not analyze the lattice. We prefer limiting the false positives at the expense of having many false negatives.

The patterns are defined as specific topology of nodes and edges, sometimes accompanied by a specific type of node for one or more of the members of the pattern. Some of these patterns are illustrated in Figure 1 to help the reader visualize them.

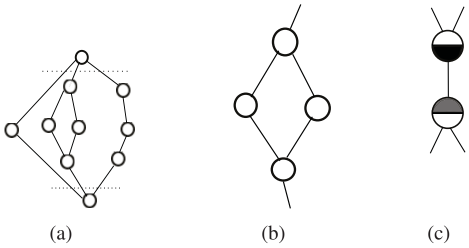


Fig. 1. Some of the patterns in concept lattices

1) Top node (⊤): We identified two cases of interest for the top node:

Top-black The pattern reveals the fact that the formal objects attached to the top node don't have relationship to any of the formal attributes present in the context. It indicates formal objects that are not relevant to the context being studied.

Top-grey or Top-full The existence of a grey or full node at the top of a lattice marks a set of formal attributes (the grey part) that are in relationship to all the formal objects contained in the context. The pattern represents an important set of formal attributes that form the very basis of the context.

2) Bottom node (⊥): We may identify two cases:

Bottom-grey The pattern reveals the fact that the formal attributes attached to the node are not in relationship with any of the formal objects present in the context. Likewise, the pattern is important as it depicts attributes that are not relevant to the context being studied.

Bottom-black or Bottom-full The existence of the black or full node at the bottom of a lattice illustrates a set of formal objects that are related to all the formal attributes present in the formal context.

3) Horizontal Decomposition: Horizontal decomposition (see Figure 1(a)) is a pattern that appears when one can identify disjoint subgraphs when removing the top and bottom of the lattice. In the illustrating figure, there are three disjoint subgraphs. Horizontal decomposition points to the presence of disjoint sets of relationships between formal objects and formal attributes. This is somehow similar to having several disjoint contexts. The different subgraphs may actually have formal attributes or formal objects in common in the case of Top-grey/Top-full and Bottom-black/Bottom-full, but assuming we ignore these (see discussion above), then each subgraph may be considered independently from all the others. Snelting and Tip mention horizontal decomposition in [28]. A less constrained pattern would be to find a horizontal decomposition, with non trivial subgraphs, between two nodes that are not the top or the bottom of the lattice. This is also related to the Module pattern that will be discussed immediately.

4) Module: In partially ordered set theory, a module represents a group of nodes that are connected to the rest of the graph through a unique top node and a unique bottom node [16]. The supremum and infimum of the module are part of it. Figure 1(b) illustrates a simple example of module in a concept lattice. A module represents an important pattern because it can be considered as a sublattice inside a concept lattice. One could imagine collapsing the entire module into one "composite node" without changing the semantics of the lattice (just make it a bit more abstract). Also, all the patterns applicable to the concept lattice can also be applied to the module. The module in itself can be seen as a smaller individual lattice that can be analyzed independently of the rest.

5) Irreducible specialization: Specializations are the basis of a lattice and one finds them everywhere: Every arc in a concept lattice marks a specialization relationship. Yet the pattern is interesting when the specialization occurs between a black (or full) super-concept and a grey (or full) sub-concept (illustrated in Figure 1(c)). Irreducible specialization patterns depicts two nodes that should not be merged. The upper node (in Figure 1(c)) needs to exist because it has its own formal object(s) and the lower node needs to exist because it has its own formal attribute(s). That is to the say specialization pattern represents two entities in a system that are relevant in the system they belong to. When the super-concept (sub-concept) in the specialization pattern is the top node (resp. bottom node), we further require that it is a full node, to ensure that it represents a complete concept with formal objects and formal attributes. When we are not dealing with the top or bottom nodes in the lattice, we don't need this restriction because the nodes in the pattern can inherit formal attributes (formal objects) from their super-concepts (resp. sub concepts).

IV. Concrete Examples

In this section, we present a validation of these patterns to evaluate that the patterns do appear in the concept lattices, and these reveal important information about the underlying program. For the examples, we experiment with various formal contexts from different program entities of open-source systems. These contexts are similar to existing contexts of the literature. The contexts will be presented as triplet: O-R-A that is to say: formal Object, Relation, and formal Attribute. Program information for the lattices is extracted through FCAParser[2].

A. Member use

This formal context is based on the following information:

- O = All methods within a class except getters and setters
- A = Attributes of the class
- R = The method accesses an attribute directly or through a getter method

The concepts resulting from this context represent different features that a class may implement, assuming that each feature will be composed of methods accessing a particular set of attributes. The example used will be the class Main of JMemorize[3]. Figure 2 shows the lattice resulting from this example. We may identify the following patterns in the concept lattice:

- **Top black:** The two methods in the top-black node do not access any attribute of the class. Code analysis shows that copyFile is a utility method whereas onProgramEnd is an empty method in the class.
- **Horizontal Decomposition:** Removing the top and bottom nodes of the lattice, we are left with 6 independent subgraphs: (i) a set of three disjoint subgraphs on the far right having methods main, run, and startStats, (ii) the bulk of the nodes in the centre, (iii) one node with methods exit, rmPrgObs and addPrgsObs on the lower left, and (iv) two nodes with methods clrThrw, logThrw. The subgraphs identify independent concerns of the class (convertible to traits). For example, the log (on the far left) or the handling of observers when the program ends (lower left).
- **Irreducible specialization:** There is one instance of this pattern on the left of the lattice. As explained, it indicates that we have here two features that could not be easily merged.
- **Module:** The same two nodes on the far left illustrate a case of the simplest possible module. The two nodes could be grouped in one to simplify the lattice. This is not incompatible with them being an irreducible specialization, as this merging into one composite node is only a proposition to simplify the lattice itself, and not something that should impact the underlying source code.

[2] http://fcaparser.googlecode.com/

[3] http://jmemorize.sourceforge.net/

The complete lattice of the class shows that the class implements several concerns: observers, logging, and application startup. The application startup concern is illustrated by the presence of main, run, and startStats methods. In the code, main calls run to start the application, which in turn calls startStats to collect program statistics during program execution. The bulk of the nodes in the center shows that these nodes implement some coherent functionality, which is revealed by their interconnections. The disjoint branches of the lattice propose to decompose the class to encapsulate each concern into a separate class.

On a single class with few members, the FCA technique and the patterns we propose may seem like an overkill, but one must understand that, in practice, such tools would be used for all the classes (thousands) of a system and a user would not have the possibility to analyze each one independently. The patterns would be a help to point out the classes that offer the best opportunities for design improvement.

B. Class Coupling

Class coupling explores the relationship amongst different classes, by the way each one uses the members of the other classes. Similarly to attribute uses (Section IV-A) for classes, concepts resulting from this context represent high level features in packages by identifying common access to other class members.

- O = Classes;
- A = Class members (attributes and methods);
- R = The class Uses a member of another class.

The example will use the classes of package org.jhotdraw.geom of JHotDraw. The resulting lattice is presented in Figure 3. We may find the following patterns:

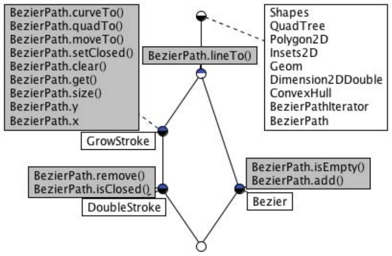


Fig. 3. Class Coupling lattice for `org.jhotdraw.geom` package in JHotDraw

- **Top black:** The pattern in this context shows the existence of nine classes in the package that do not access members of any of the classes in the package. In this specific case, the lack of communication is due to the fact that this package is an intermediary layer between java 2D graphics (java.awt.geom) and the rest of the JHotDraw framework.
- **Irreducible specialization:** There is a case of irreducible specialization that is similar to the one observed in Section IV-A.
- **Module:** There are two simple modules in this lattice. The first one consists of the lattice itself minus its top node. It gives us all the classes that interact together. The second one, two nodes on the left, for which we can draw the same conclusions that in example Section IV-A. Of course in such a small case, the simplifications that node aggregation would bring are not really needed.
- **Horizontal decomposition:** If we focus on the largest non-trivial module (whole lattice without the top black node), we can detect the presence of a horizontal decomposition pattern with two independent branches. They suggest two features: one that works with BezierPath s as containers (isEmpty(), add()) and the other one that sees them as graphical elements (moveTo(), isClosed(), . . .). This shows that the class BezierPath implements two different concerns and requires refactoring different concerns in separate classes.

V. Prototype for Pattern Identification

We have provided concrete examples that show that the catalogue of patterns does reveal some important spots in concept lattices. We developed a tool in Moose [11] to automate the task of pattern identification in concept lattices. The tool supports the identification of all the patterns. We used the tool to extract the patterns from the lattice constructed from all classes in OpenBravo[4] using the following formal context:

- O =All classes;
- A = Method names;
- R = The class locally defines the method name

Table II provides a summary of the results produced by the tool. The resultant concept lattice is composed of 1053 nodes and 2260 connections. Of those nodes and connections, the user needs to study 154 nodes and 150 connections, if (s)he wants to explore all the patterns. This represents a reduction of 85% in the number of nodes to analyze and 93% in the number of connections.

TABLE II
Patterns in OpenBravo

# of formal objects	756
# of formal attributes	6658
# of nodes	1053
# of node connections	2260
Top	Top Black (1 node, 0 connections)
Bottom	-
Irreducible specializations	45 (90 nodes, 45 connections)
Horizontal decomposition	28 (28 nodes, 56 connections)
Modules	14 (49 nodes, 49 connections)
Total Nodes and connections in Patterns	154 nodes, 150 connections

The patterns in the lattice reveal: classes without methods or empty classes (Top black); hierarchies of classes having similar methods (Irreducible specialization); classes that do not have common methods with other classes (Horizontal decomposition). Modules in the lattice represent different methods that are common (duplicated) amongst the classes of

[4] http://www.openbravo.com/

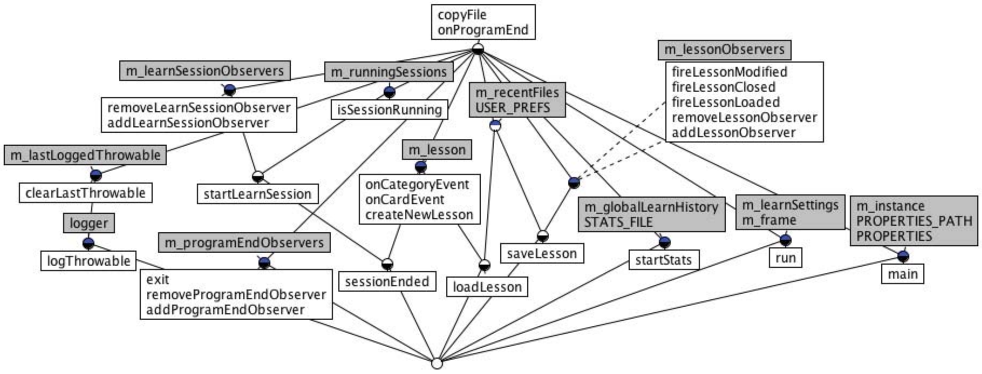


Fig. 2. Lattice for `Main` in JMemorize

the system. Hence, the modules nodes can be used to refactor these classes toward a better design.

VI. Conclusion

An inconvenience of using FCA is that lattices are complex and one needs to be knowledgeable about FCA to extract useful information. The paper presents patterns in concept lattices that can ease the task of lattice interpretation. We demonstrate that: The patterns reveal important information for the underlying context; The patterns identify pertinent results regardless of the context used for creating the formal concept; All mined patterns in the examples are pertinent, thus false-positive results are absent. A prototype is presented that searches for the patterns in lattices. In software reengineering, such a tool could support automated identification of problem areas. For our future work, we are interested in expanding the catalogue of patterns and mix our approach with filtering techniques.

References

[1] N. Anquetil. A comparison of graphs of concept for reverse engineering. IWPC, 2000.

[2] G. Arévalo, S. Ducasse, S. Gordillo, and O. Nierstrasz. Generating a catalog of unanticipated schemas in class hierarchies using formal concept analysis. *Inf. Softw. Technol.*, 52:1167–1187, November 2010.

[3] G. Arévalo, S. Ducasse, and O. Nierstrasz. Understanding classes using X-Ray views. In *MASPEGHI 2003 (ASE 2003)*, pages 9–18, Oct. 2003.

[4] T. Ball. The concept of dynamic analysis. In *ESEC/FSC'99*, number 1687 in LNCS, pages 216–234, Heidelberg, sep 1999. Springer Verlag.

[5] M. U. Bhatti, S. Ducasse, and M. Huchard. Reconsidering classes in procedural object-oriented code. In *WCRE*, 2008.

[6] G. Canfora, A. Cimitile, A. De Lucia, and G. A. Di Lucca. A Case Study of Applying an Eclectic Approach to Identify Objects in Code. In *Proceedings of IWPC '99*, pages 136–143. IEEE, May 1999.

[7] M. Dao, M. Huchard, M. R. Hacene, C. Roume, and P. Valtchev. Improving generalization level in uml models iterative cross generalization in practice. In K. E. Wolff, H. D. Pfeiffer, and H. S. Delugach, editors, *ICCS*, volume 3127 of *Lecture Notes in Computer Science*, pages 346–360. Springer, 2004.

[8] U. Dekel and Y. Gil. Revealing class structure with concept lattices. In *WCRE*, pages 353–362. IEEE Press, Nov. 2003.

[9] A. Deursen and T. Kuipers. Identifying objects using cluster and concept analysis. In *Proceedings of ICSE '99*, pages 246–255. ACM Press, 1999.

[10] H. Dicky, C. Dony, M. Huchard, and T. Libourel. On automatic class insertion with overloading. In *OOPSLA*, pages 251–267, 1996.

[11] S. Ducasse, T. Gîrba, and O. Nierstrasz. Moose: an agile reengineering environment. In *Proceedings of ESEC/FSE 2005*, pages 99–102, Sept. 2005. Tool demo.

[12] T. Eisenbarth, R. Koschke, and D. Simon. Locating features in source code. *IEEE Computer*, 29(3):210–224, Mar. 2003.

[13] J.-R. Falleri, M. Huchard, and C. Nebut. A generic approach for class model normalization. In *ASE*, pages 431–434. IEEE, 2008.

[14] B. Ganter and R. Wille. *Formal Concept Analysis: Mathematical Foundations*. Springer Verlag, 1999.

[15] R. Godin and H. Mili. Building and Maintaining Analysis-Level Class Hierarchies using Galois Lattices. In *Proceedings OOPSLA '93*, volume 28, pages 394–410, Oct. 1993.

[16] M. Habib, M. Huchard, and J. Spinrad. A linear algorithm to decompose inheritance graphs into modules. *Algorithmica*, 13(6):573–591, 1995.

[17] M. Huchard and H. Leblanc. Computing Interfaces in JAVA. In *Proceedings of ASE '2000*, pages 317–320, 2000.

[18] M. Huchard and H. Leblanc. Computing interfaces in java. In *ASE*, pages 317–320, 2000.

[19] P. Joshi and R. K. Joshi. Concept analysis for class cohesion. In *Proceedings CSMR 2009*, pages 237–240.

[20] A. Kellens, K. Mens, and P. Tonella. A survey of automated code-level aspect mining techniques. *Transactions on Aspect-Oriented Software Development*, 4(4640):143–162, 2007.

[21] A. Lienhard, S. Ducasse, and G. Arévalo. Identifying traits with formal concept analysis. In *ASE'05*, pages 66–75, Nov. 2005.

[22] C. Lindig. Mining patterns and violations using concept analysis. Technical report, Saarland University, Germany, 2007.

[23] K. Mens and T. Tourwé. Delving source code with formal concept analysis. *Comput. Lang. Syst. Struct.*, 31:183–197, October 2005.

[24] N. Moha, A. M. R. Hacene, P. Valtchev, and Y.-G. Guéhéneuc. Refactorings of design defects using relational concept analysis. ICFCA'08, pages 289–304. Springer-Verlag, 2008.

[25] D. Poshyvanyk and A. Marcus. Combining formal concept analysis with information retrieval for concept location in source code. In *ICPC '07*, pages 37–48, Washington, DC, USA, 2007. IEEE Computer Society.

[26] H. A. Sahraoui, H. Lounis, W. Melo, and H. Mili. A concept formation based approach to object identification in procedural code. *Automated Software Engineering Journal*, 6(4):387–410, 1999.

[27] M. Siff and T. Reps. Identifying modules via concept analysis. *Transactions on Software Engineering*, 25(6):749–768, Nov. 1999.

[28] G. Snelting and F. Tip. Understanding Class Hierarchies Using Concept Analysis. *ACM TOPLAS*, pages 540–582, May 2000.

[29] M. Streckenbach and G. Snelting. Refactoring class hierarchies with KABA. In *OOPSLA '04*, pages 315–330, New York, NY, USA, 2004.

[30] A. Sutton and J. Maletic. Recovering uml class models from c++: A detailed explanation. *Inf. Softw. Technol.*, pages 212–229, March 2007.

[31] T. Tilley, R. Cole, P. Becker, and P. Eklund. A survey of formal concept analysis support for software engineering activities. In G. Stumme, editor, *Proceedings of ICFCA '03*. Springer-Verlag, Feb. 2003.

[32] P. Tonella and G. Antoniol. Object oriented design pattern inference. In *Proceedings of ICSM '99*, pages 230–238. IEEE Computer Society Press, Oct. 1999.

Client-Side Rendering Mechanism: A Double-Edged Sword for Browser-Based Web Applications

Hao Han*, Yinxing Xue† and Keizo Oyama*‡
*National Institute of Informatics, Japan
{han,oyama}@nii.ac.jp
†National University of Singapore, Singapore
yinxing@comp.nus.edu.sg
‡The Graduate University for Advanced Studies (SOKENDAI), Japan

***Abstract*—Rendering mechanism plays an indispensable role in browser-based Web application. It generates active webpages dynamically and provides human-readable layout of information. Client-side rendering system brings various flexibilities but also new problems. In this paper, we give a comprehensive analysis of possible advantages and disadvantages brought by client-side rendering mechanism in viewpoints of both developers and users based on practice and experience.**

***Index Terms*—Rendering Mechanism, Template Engine, Web Application, Web UI, Practice and Experience**

I. Introduction

Browser-based Web application is one of often used computer software applications designed for information interactivity on the Web. It is reliant on a common Web browser to render the application executable, and usually coded in browser-rendered/supported languages such as HTML (HyperText Markup Language) and JavaScript. At server side, rendering mechanism is indispensable for human-readable presentation and generates webpages based on template engine. Figure 1 describes the basic architecture of a typical server-side rendering system. In general, rendering system processes templates and content to generate the output Web documents. A template engine specifies a template and fills in the template with the assigned values to obtain the output page. Templates are frequently used to separate the webpage presentation from the underlying business logic and data computations in the context of Web application development. In the case of Web application development, this means that “no logic computation in HTML and no HTML in logic computation” roughly. Server responses may be determined by requests sent from client side such as data in a posted HTML form, parameters in the URL, or the type of Web browser being used. Server-side scripting (program running on the server) is used to change the web content on various webpages. Such webpages are often created with the help of server-side languages such as ASP, Perl, and PHP.

Web applications become more and more complicated with the advancement of diverse Web technologies in order to realize the demands of new techniques for the different kinds of users. The users prefer the Web applications containing the user-friendly interfaces and easy operations. For example, Ajax (Asynchronous JavaScript and XML) dynamically interchanges content which sends a request to the server for data. The server returns the requested data subsequently formatted by a client side script, which reduces server load time because the client does not request the entire webpage regenerated by the server-side rendering. This trend accelerates the development of interactive and animated websites. Many websites use the DHTML (Dynamic HTML) or DOM scripting technology to create the interactive Web pages. It contains HTML, client-side scripts (such as JavaScript), DOM (Document Object Model), etc. The scripts change variables programmatically in an HTML document, and affect the look and function of static content.

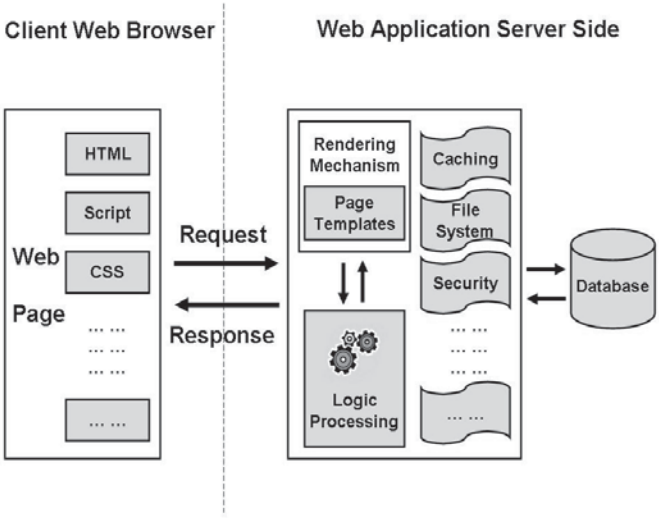


Fig. 1. Server-side rendering system

Client-side rendering mechanism is one of proposed approaches responding to this trend. Instead of a fully rendered HTML page, client-side rendering system produces skeletal segments, which are combined and transformed into an HTML page at client-side Web browser. XML (Extensible Markup Language), XSLT (XML Stylesheet Language Transformations [26]), JSON (JavaScript Object Notation) and JavaScript are usually used to generate or update the dynamic content (interactive and animated parts) at client side. Each segment presents an individual topic or functional region of webpage and only the segment that subject to changes is transmitted. These interactive webpages bring the users various visual effects and consume fewer resources on the server.

As a developing technology always has its faults, client-side rendering mechanism also has both merits and drawbacks. In this paper, we propose a developing framework as a simple implementation of client-side rendering mechanism, and give a comprehensive analysis of possible advantages and disadvantages in different viewpoints such as accessibility, caching, personalization and speed. Our analysis emphasizes on issues like performance, practice and experience at client side rather than the business/processing logic at server side, and the specific Web browser plug-ins are not discussed. Some experimental evaluations about rendering cost and development time are also discussed.

The organization of the rest of this paper is as follows. In Section II, we present a developing client-side rendering framework and introduce some related approaches/systems. From Section III we give analysis of performance/functionality at client-side and application development based on various viewpoints. Finally, the conclusion and the future work are given in Section IX.

II. An Example of Client-Side Rendering System

There is no uniform and mature structure of client-side rendering system widely accepted by general Web applications. In order to provide an intuitive and visible image of client-side rendering for the further analysis and explanation, we present an example based on XML and XSLT technology, which is generally employed as a stylesheet processor used to transform XML data into HTML or XHTML documents.

We are developing an XML-based framework for generating flexible and extensible Web applications based on client-side rendering mechanism as shown in Figure 2. Here, each webpage is divided as a set of static segments and dynamic segments. Static segment always comprises and displays the same information in response to all requests from all users and in all contexts. (e.g. navigation bar and site information placed in page footer). Dynamic segment presents dynamically generated content that can change in response to different contexts or conditions. There are two ways to create this kind of effect/change. Generally, server-side program is used to change the page source determined by requests from client side. At the client, client-side scripting is used to change interface behaviors within a specific webpage in response to mouse or keyboard actions or specified timing events. In this case the dynamic behavior occurs within the page presentation. Figure 2 describes the workflow and data flow of framework as follows.

1) The server responds to the request of client with the segment container file. A container is a layout page that surrounds or references static segments, XML data, XSLT files, or external files like JavaScript and CSS (Cascading Style Sheets) files in its page body. The XML data is encapsulated into a single XML document in order to reduce the connection time before it is sent to client side. The container file also contains the mapping information, which reflects the one-to-one mapping between the XML data and XSLT file.
2) The dynamic segment is generated at client-side browser. The XSLT processor transforms the corresponding XML data into HTML or XHTML document based on the one-to-one mapping. The static segments and generated dynamic segments comprise the main source of an entire client-side webpage.
3) When the end-users trigger the mouse/keyboard events defined in XSLT processors of dynamic segments, the variables in corresponding XML data are changed by JavaScript functions, and the XSLT processors retransform the updated XML data into the new HTML segments.

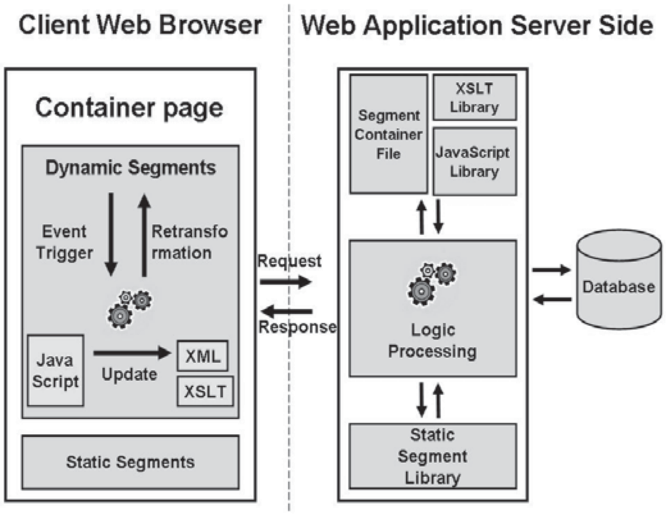


Fig. 2. An example of client-side rendering system

A more detailed description of this system can be found in [9].

As a widely used technology, the open source community has created a huge number of client-side templating solutions. Logic-less templating technologies follow the strict model-view separation rule that there should be little or no logic in templates. They usually provide a clean separation between presentation and logic without sacrificing ease of use, and are particularly well-suited for asynchronous and streaming applications. Twitter [1] uses Mustache [16] and JSON to move significant pieces of functionality to the browser, and Google plus [2] employs Java-friendly Closure [1]. Linkedln [3] moves from server-side templates to client-side JavaScript templates powered by dust.js [3].

Besides these logic-less templates, there are embedded templating options, which allow developers to embed regular

[1] https://twitter.com/
[2] https://plus.google.com/
[3] http://www.linkedin.com/

JavaScript code directly within the template. Underscore.js [24] is based on microtemplating and provides a lot of the functional programming support usually expected in Prototype.js [20]. jQuery [13] simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid Web development. Jade [11] is influenced by Haml [8] syntax and implemented with JavaScript for node.js [17].

Many related studies have given the discussion about “rendering mechanism and rich application” with “component-based” (Fiz [6]), “data-centric” (Hilda [27]), “strict model-view separation” ([19]), “flexible model-view separation” ([7]), “skeletal script” (FlyingTemplate [22]), “interactive behaviors learning” ([18]), “accessibility evaluation” ([5]) and others described in following sections.

III. Accessibility

RIAs (Rich Internet Applications or we call them rich client applications) and supporting technology reflect an implementation of rendering mechanism in the user agent or in the browser at the client side. They provide more dynamic Web content and more attractive and interactive websites. However, not all end-users benefit from this interactivity. For example, users with weak or disabled eyesight access the Web using assistive technologies, such as a screen reader (e.g. IBM Home Page Reader [4]) that delivers audio content. If a client-side rendering/programming technique is used by a webpage, the screen reader cannot read the page-based operation (page modification), which is particularly problematic for the keyboard navigation essential to accessibility. With the meta data or HTML5 [10], interface developers should be able to adapt interfaces to meet specific needs partially. However, meta data is recommended but not necessary in HTML page, and until now most of developers do not use HTML5 tags widely.

Rich defined XML is better for screen reader, not only in reading but also page-based operation [15]. Figure 3 shows an example of XML document employed in Figure 2. It adds semantics to webpage components and content so that assistive technologies can interpret their operation. Regional landmark roles provide a navigable structure within a webpage, and node names and attributes present a guarantee of accessibility of controls. For example, “writable” represents the dynamic content-update notifications: update, remove, and add (0: cannot, 1: can).

```
<contact role="contact list">
 <personal role="personal contact">
  <person role="main contact">
   <name writable="000">Nakata</name>
    <address writable="100">Yokohama, Japan</address>
    <mainemail writable="100">nakata@example1.com
    </mainemail>
    <otheremail writable="111">nakata@example2.com
    </otheremail>
    <tel writable="101">81-3-1234-5678</tel>
     ... ...

  </person>
 </personal>
 <business>
  ... ...

 </business>
 <others>
  ... ...

 </others>
</contact>
```

Fig. 3. Names and attributes (embedded in XML) add semantics to webpage

IV. Data Caching

A Web data caching is a mechanism for the temporary storage of Web documents, such as HTML pages, to reduce bandwidth usage, server load, and perceived lag. For Web applications distributed by HTTP (Hypertext Transfer Protocol), freshness, validation, and invalidation mechanisms are used for controlling caches. Dynamic webpages are often cached when there are few or no changes expected and the page is anticipated to receive considerable amount of Web traffic that would create slow load time for the server if it had to generate the pages on the fly for each request. Despite the popularization of broadband networks, page or page segment is still a most basic and often used data transfer unit for Web applications. So, it would lease the load of server if we reduce the data transfer of page.

Page segment and client-side rendering mechanism bring more efficient caching functionality at server side and client-side Web browser. Compared with the traditional page-based caching system, static segments would be widely cached and independent of changeable dynamic segments. Moreover, the XSLT templates of dynamic segments could be reused or cached at client Web browsers and the layout at client side does not affect the cache. This reuse/caching could reduce the data transfer between the server and client. As a experimental example, we reconstructed the response Web page (search result page) of Google Search into XML and XSLT data. The experimental results show that the transferred data size is much reduced as shown in Table I if page is rendered at client side.

TABLE I
Response data size comparison

Structure	Data Format	Size (bytes)
Server-side rendering	HTML	20,916
Client XSLT caching	XML	6,794

For the Web browsing at mobile devices, the page segments can be simplified for better presentation at mobile Web browsers. Android developers also recommend upgrading the Web UI to an XML layout [25]. Moreover, mobile devices could partially share the XML data in server side caching

[4] http://www-03.ibm.com/able/

system with the general desktop Web browsing like XML-based message exchange between different platforms.

V. Layout Customization

A configurable page is a webpage designed with built-in layout personalization and content selection capabilities. Each piece of content, also referred to as a resource displayed within a cell of a layout region, can be rearranged, hidden or shown. Client-side XSLT customization is more flexible and powerful than CSS personalization, which is often used in blog pages. For example, it is easy to rearrange the layout or control the visibility by changing the attribute values of nodes.

- Layout arrangement (Figure 4): End-users can move segments by dragging and dropping operations to adjust the locations (e.g. object.style.left = event.clientX - event.offsetX + document.body.scrollLeft; object.style.top = event.clientY - event.offsetY + document.body.scrollTop;), which is more compact or suitable for user browsing habits than the default layout arrangement of segments.

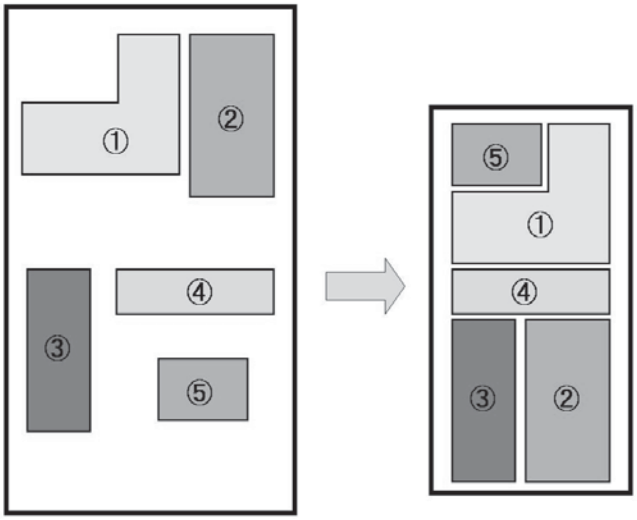


Fig. 4. Layout rearrangement

- Visibility control (Figure 5): End-users can hide/undisplay the undesired segments by setting the property “display” of attribute “style” to “none” (object.style.display = “none”;). If the hidden segments are deleted, the original execution environment of JavaScript would be broken and the JavaScript could not run normally if the hidden segments (XML data) are used/updated in JavaScript programmatically.

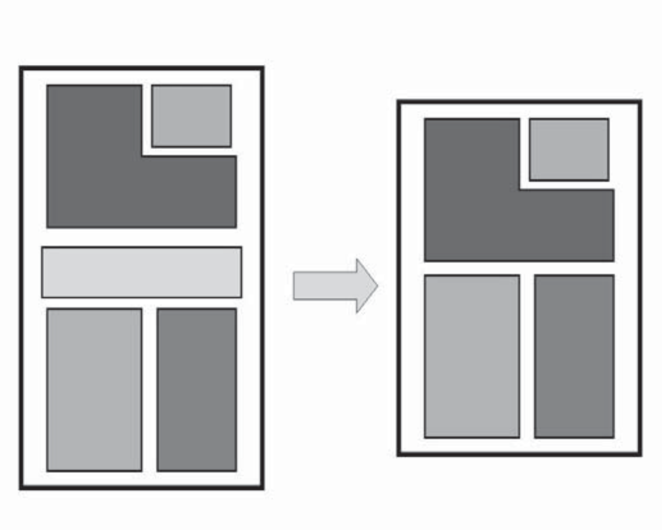


Fig. 5. Hide segment(s)

VI. Speed and Delay

If server-side load is reduced and rendering tasks are moved to client-side, tasks of client-side are unavoidably increased. Client-side rendering and rich client applications usually bring users a “image” that they are executed heavily and high CPU/memory-consuming, which leads to delay of page loading or refreshing at client Web browser. Actually, this delay is caused by large data streaming and plug-in such as JSON text stream in map applications, Flash player and Silverlight, or JavaScript loading. Client-side rendering itself is not a high CPU-consuming process. We (OS: Windows 7, Browser: Internet Explorer 9, CPU: Intel Core i7 2.93GHz, RAM: 4.00 GB) captured performance data of examples given in Section IV by Windows Performance Analyzer Tools [5]. As shown in Figure 6 and 7, there are not many differences in CPU usage percentage (e.g. maximum and average value) between presenting different types of webpages. A similar statistical result of main memory usage is also learned.

As clients become increasingly sophisticated, there is more for a browser to do. For websites that rely heavily on client-side rendering, it is essential to include this delay. Too much JavaScript on the client side makes the browser slow unless the user has a powerful computer. Many large websites use one common set of JavaScript files and one common set of CSS files in every template and webpage. Those files change over time and often contain elements that are no longer used on the webpage or anywhere on the website. We need to keep track of what elements are being loaded on each webpage. client side uses JavaScript libraries that often contain massive amounts of functions, but only a handful of those functions are used actually. If we want to speed up presenting webpage, we should remove any JavaScript or CSS that are not being used on the webpage.

Moving JavaScript to the bottom of webpage also enables all other requests to be processed quickly, which would make webpage appear to load more quickly. The browser can begin rendering faster and do not wait for all of JavaScript to load at the top of webpage. However, not all JavaScript can be moved to the bottom of a webpage. If the JavaScript is a

[5] http://msdn.microsoft.com/en-us/performance

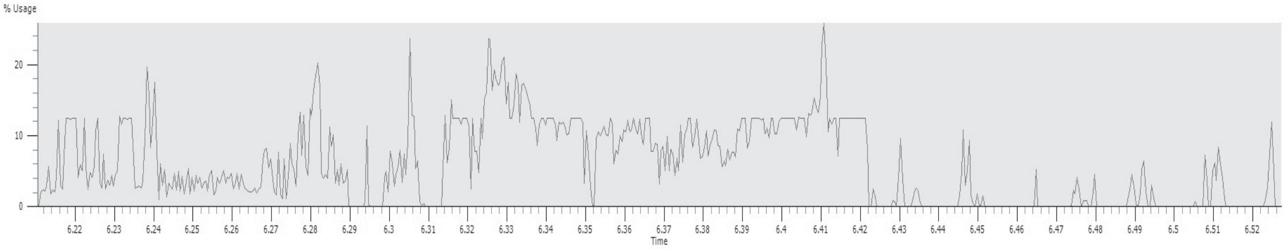


Fig. 6. CPU usage percentage of presenting a webpage rendered at server side

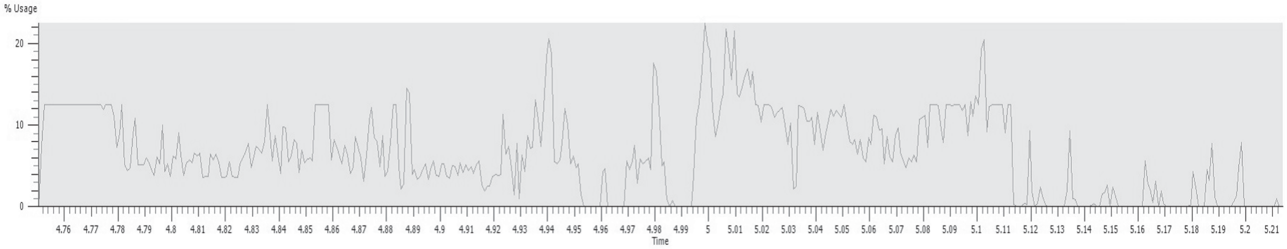


Fig. 7. CPU usage percentage of presenting a webpage rendered at client side

library required for other components on the page to render, it must be loaded early in the page lifecycle.

More discussions about bridging the gap between the browser view of a UI and its JavaScript implementation are given in [14].

VII. Development and Reuse

We have to consider development efficiency and programming skill requirements if we employ client-side rendering mechanism. It is based on the separation of topic and functionality region of webpage. We implement not only the model-view separation in template but also the static-dynamic separation in segment. In the model-view separation, XSLT serves as client-side template and offers a great expressiveness, allowing a translator to produce complex HTML documents from XML data. This separation leads to the division of labor. The webpage designer can adjust the presentation or layout without having to change the program logic, which is always much riskier. However, XSLT files development requires mastering a quite different language compared with JSP/ASP/PHP and high proficiency with JavaScript frameworks/libraries (e.g. jQuery [13] and Prototype.js [20]). Therefore, this XSLT-XML separation methodology and client-side rendering mechanism are not suitable for small personal webpages. It would waste the time by analyzing and has no big effect.

Here, we give two experiments to prove the abovementioned opinions and compare development time cost.

1) Experiment 1: Developers are not proficient in XSLT and write original JavaScript functions for rendering. We developed a CIM (Customer Information Management) system, which provides the *add*, *search*, *show*, *update*, and *delete* functions mainly. By the traditional method, it needs five basic functionality pages (and other pages such as listing page): add a new customer (input the customer information), search for a customer (input the search keyword), show the customer information (detailed information such as name, address, telephone, and statistic of purchase history), update a customer information (e.g. change the address, add a new contact), and delete a customer. By using dynamic page, the *show* page and *update* page are merged into a *show+update* page. The users can update the information in *show* page by clicking the corresponding value area (trigger *Onclick* event) without the page jumping/transition. The updated information includes changed user information and calculated new purchase statistic (e.g. sum, average price, and graphic statistic). Compared with the two standalone pages *show* and *update*, the *show+update* page needed much more time in programming and test as shown in Table II (same developing engineers and quality/test engineers). The developing engineers had to write large quantities of JavaScript to deal with the DOM and hidden values programmatically, and face up to the fact that XML syntax is far more restrictive than HTML. Moreover, browsers' debugging support is still very poor compared with server-side debugging support. Therefore, although the sum of pages is reduced, the cost of programming and test becomes higher.

TABLE II
Development time cost comparison

Page (Function)	Programming (hours)	Testing (hours)
show	6	8
update	12	16
show+update	40	64

2) Experiment 2: Developers are proficient in XSLT and

JavaScript library. We upgrade a Web UI (User Interface) system, which is an online information processing system and has two versions. The UI generator of old one is developed by JSP (JavaServer Pages) and the new one is developed by XSLT. In a version upgrade, a new functionality needs to be added for personalizing webpage layout based on user access authorization. A developing engineer did the code update for two versions. Table III shows that the XML-XSLT architecture is more efficient. The programming job could be analyzed and divided according to the abilities or proficiencies of developing engineers. One engineer does not need to write an entire template file of a webpage, and the webpage development can be subdivided into functionality-oriented development. For the dynamic segment, the JavaScript library is used to access or update client XML data. Compared with the traditional dynamically updating client HTML source, the XML data access/update/transformation process is independent of the layout of webpage, and avoids using the HTML tags and HTML-oriented functions such as *innerHTML* or *innerText*, which are lack of the possibility of code reuse. XML-oriented JavaScript functions are customized more easily than HTML parsing functions.

TABLE III
UPGRADE TIME COST COMPARISON

Version	Language	Time (hours)
Old	JSP	184
New	XSLT	80

VIII. DISCUSSION

The design philosophy behind our approach/opinion aims at achieving the following goals.

1) To extricate the server side from the overloaded rendering work requested simultaneously by multiple client ends.
2) To further separate the concerns of the Web application and distinguish the handling of various types of data.
3) To facilitate the personalization and customization of the displayed output webpages for the different users and types of client ends.

For the goal 1, in the client-server model, there is one common controversy about the choice of the thin-client architecture or the thin-server (full-client [12]) architecture. Yang et al. [28] found that thin-client systems can perform well on Web and multimedia applications in LAN (Local Area Network) environments. This shows that the rendering mechanism of the traditional Web application worked fine when it was usually at the server side. However, nowadays with the better and better graphical quality displayed by Web applications and more various requirements from users, the rendering tasks become more and more computationally costly. Thus, keeping the rendering mechanism still at server side becomes an obsolete design. Now there is a trend that TSA (Thin Server Architecture) is advocated recently [4][23]. Note that in our approach, we are not at the extreme to have most functions of Web applications exclusively limited at the server side or at the client side. The idea of our approach is not to use server-side templates to create and transmit the webpage, but to separate concerns using protocol between client and server and to get a much more efficient and less costly development model. Using TSA leads to following three potential advantages.

- The server side can only focus on the business logics.
- The client side can focus on the presentations.
- The communications between server and clients just exchange the raw data such as XML.

As shown in Table I, the exchanged data between server and client is reduced in our approach, which improves the usage of data caching. Besides, moving the rendering from server to client also facilitates the concurrency of presentation. The XSLT used in our approach, as a functional programming language, can easily enable concurrency, since the order of execution of side-effect free functions in functional programming is not important. Thus, the client can run the multiple same XSLT threads simultaneously to speed up the data presentation.

For the goal 2, after separation of presentation and content, for the ease of reusability, maintainability and extensibility, it is desired to further separate the concerns of the Web application on multiple dimensions [21]. The accessibility is one of the dimensions we try to separate concerns on. By adopting the RIAs or rich defined XML, the client side can provide navigable structure and guarantee of accessibility of controls (see Section III). Another dimension of concern important to the performance is the different types of content. For example, the static and dynamic content in the different segments in the output webpages should be differentiated. In the original server-side rendering mechanism, the data exchanged between server and client is usually webpages. In our approach, with the skeletal segments generated by XML, XSLT, JSON and JavaScript, the exchanged data between server and client can be narrowed down to the segment level. And distinguishing the static and dynamic segments also improves the efficiency of data caching (see Section IV). Except the different handling for static and dynamic content, the similar idea can be applied to yet some other types of content.

For the goal 3, as the user-adaptive and context-aware characteristic is the trend for the future Web applications [2], Web application should have the different rendering strategies for the different types of client ends and the various user requirements. Layout arrangement and visibility control (see Section V) is just one type of user customization and personalization. In some more sophisticated client ends like smart-phone, the gesture control in response to the users' operations on the screen is better to be handled by the client end, otherwise waiting for the rendering results from server-side will cause the extra delay.

Certainly, the above merits of client-side rendering do not come without any compromise. As shown in Table II and Table III, using our approach together with the involved techniques may introduce the extra learning curve and initial investment for the early separations than the commonly used server-side rendering mechanism (taking almost more than double time, see Table II). However, in the long term, the developers will benefit from this in the evolution and extension of the Web applications.

IX. Conclusion

We gave a comprehensive analysis of advantages and disadvantages of client-side rendering mechanism in different viewpoints of practice and experience. The experimental evaluations proved a proposed example framework supports the demands of users well and could bring the diverse new opportunities about the extensible reuse of Web applications.

There is still a lot of work to be done before it becomes a more mature technology since it currently cannot arrive at a solution satisfactory to both users and developers. The dynamic visual effects are realized by client script functions, which bring the flexible operation but increase the development cost for some developers or development scenarios. Therefore, as shown in Figure 8, the developers need to make a proper balance between the opinions and options of users and themselves usually.

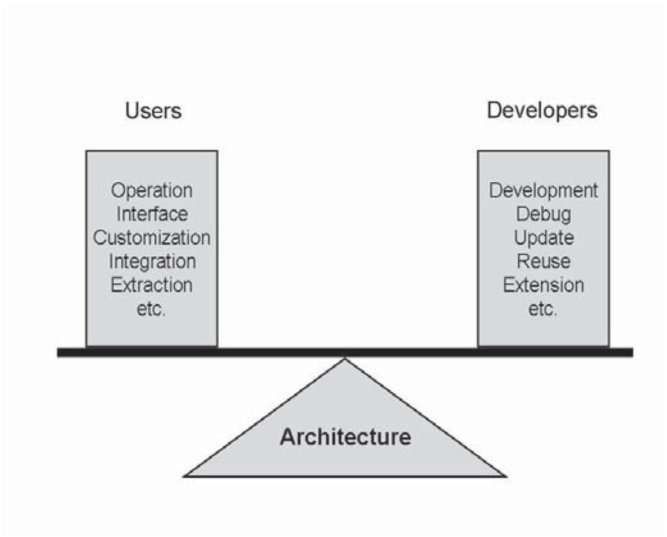


Fig. 8. Balance between Web application users and developers

As future work, we will explore further the problems of system security, scalability, and server-side database update of client-side rendering mechanism. Additionally, besides the currently developing JavaScript libraries and frameworks, we will develop more various supporting technologies for efficient client rendering in future.

X. Acknowledgement

We gratefully acknowledge the advice and experiment support from Bo Liu (Fuji Xerox, Japan). This work was supported by a Grant-in-Aid for Scientific Research A (No.22240007) from the Japan Society for the Promotion of Science (JSPS).

References

[1] Closure. http://code.google.com/closure/.

[2] Peter Dolog. Designing adaptive Web applications. In *The Proceedings of the 34th Conference on Current Trends in Theory and Practice of Computer Science*, pages 23–33, 2008.

[3] Dust. http://akdubya.github.com/dustjs/.

[4] Extreme Scale: Thin Server Achitecture. http://www.slideshare.net/spacemonkeylabs/thin-server-architecture.

[5] Nadia Fernandes, Daniel Costa, Sergio Neves, Carlos Duarte, and Luis Carrico. Evaluating the accessibility of rich internet applications. In *The Proceedings of the 9th International Cross-Disciplinary Conference on Web Accessibility*, number 13, 2012.

[6] Fiz. http://fiz.stanford.edu/home/home.

[7] Francisco J. Garcia, Raul Izquierdo Castanedo, and Aquilino A. Juan Fuente. A double-model approach to achieve effective model-view separation in template based Web applications. In *The Proceedings of the 7th International Conference on Web Engineering*, pages 442–456, 2007.

[8] Haml. http://haml-lang.com/.

[9] Hao Han and Bo Liu. Problems, solutions and new opportunities: Using pagelet-based templates in development of flexible and extensible Web applications. In *The Proceedings of 12th International Conference on Information Integration and Web-based Applications and Services*, pages 677–680, 2010.

[10] HTML5. http://www.w3.org/TR/html5/.

[11] Jade. http://jade-lang.com/.

[12] Jin Jing, Abdelsalam Sumi Helal, and Ahmed Elmagarmid. Client-server computing in mobile environments. *ACM Computing Surveys*, 31(2):117–157, 1999.

[13] jQuery. http://jquery.com/.

[14] Peng Li and Eric Wohlstadter. Script InSight: Using models to explore JavaScript code from the browser view. In *The Proceedings of the 9th International Conference on Web Engineering*, pages 260–274, 2009.

[15] Lourdes Moreno, Paloma Martinez, Belen Ruiz, and Ana Iglesias. Toward an equal opportunity web: Applications, standards, and tools that increase accessibility. *Computer*, 44(5):18–26, 2011.

[16] Mustache. http://mustache.github.com.

[17] node.js. http://nodejs.org/.

[18] Stephen Oney and Brad Myers. Firecrystal: Understanding interactive behaviors in dynamic Web pages. In *The Proceedings of the 2009 IEEE Symposium on Visual Languages and Human-Centric Computing*, pages 105–108, 2009.

[19] Terence John Parr. Enforcing strict model-view separation in template engines. In *The Proceedings of the 13th International Conference on World Wide Web*, pages 224–233, 2004.

[20] Prototype. http://www.prototypejs.org/.

[21] Peri Tarr, Harold Ossher, William Harrison, and Jr. Stanley M. Sutton. N degrees of separation: multi-dimensional separation of concerns. In *The Proceedings of the 21st International Conference on Software Engineering*, pages 107–119, 1999.

[22] Michiaki Tatsubori and Toyotaro Suzumura. HTML templates that fly: a template engine approach to automated offloading from server to client. In *The Proceedings of the 18th International Conference on World Wide Web*, pages 951–960, 2009.

[23] Thin Server Achitecture. http://www.thinserverarchitecture.com/.

[24] underscore.js. http://documentcloud.github.com/underscore/.

[25] Upgrade the UI to an XML Layout. http://developer.android.com/guide/tutorials/hello-world.html.

[26] XSL Transformations. http://www.w3.org/TR/xslt20/.

[27] Fan Yang, Nitin Gupta, Nicholas Gerner, Xin Qi, Alan Demers, Johannes Gehrke, and Jayavel Shanmugasundaram. A unified platform for data driven Web applications with automatic client-server partitioning. In *The Proceedings of the 16th International Conference on World Wide Web*, pages 341–350, 2007.

[28] S. Jae Yang, Jason Nieh, Matt Selsky, and Nikhil Tiwari. The performance of remote display mechanisms for thin-client computing. In *The Proceedings of the General Track of the Annual Conference on USENIX Annual Technical Conference*, pages 131–146, 2002.

An Empirical Study on Improving Trust among GSD Teams Using KMR

Mamoona Humayun
Department of computer Science and Technology
Harbin Institute of Technology,Harbin, China
mamoona@hit.edu.cn

Cui Gang
Department of computer Science and Technology
Harbin Institute of Technology,Harbin, China
cg@hit.edu.cn

Abstract—Trust is one of the key factors in successful software development. However, achieving and maintaining trust in global software development (GSD) projects is really hard because of the inherent challenges in global software development; these challenges include cultural diversity, inadequate communication, temporal difference and knowledge management (KM). Many studies have explored the role that trust plays in knowledge seeking and acceptance, but very few have explored the role knowledge management plays in building trust. In order to minimize the problem of trust, we propose the use of Knowledge Management Repository (KMR) that helps in building trusting working relationships among GSD team members. This paper reports the results of a controlled experiment that was conducted in an academic setting with two groups of students to test the impact of KMR on trust. The results indicate that applying KMR in GSD projects positively affects the trusting working relationship among GSD teams. Study findings provide a substantial understanding of trust, and the role KMR plays in building and maintaining trust. In this paper, we discuss these findings and their implications in GSD organizations.

Keywords-*Global software Development, Knowledge management, Knowledge Management Repository, Trust*

I. INTRODUCTION

Global software development is becoming a norm in software industry and organizations are now rapidly shifting from a traditional form of collocated development to global software development [1, 2]. There are a number of benefits and business reasons that motivate companies to shift from in-house development to GSD; these reasons include the latest technologies, availability of resources and methodologies, being closer to emerging markets, low cost, etc [3]. However these benefits come with associated costs and challenges that result in poor communication, lack of trust and coordination [4, 5, 6, 7, 8, 9, 10, 11].

Trust is defined as the ‘‘the willingness of a party to be vulnerable to the actions of another party based on the expectation that the other will perform a particular action important to the trustor, irrespective of the ability to monitor or control that other party” [28]. Trust is considered as one of the most important factors in successful software development and is essential within an organization for improving performance, efficiency, productivity, creativity and the overall results achieved. In GSD, working in a trust oriented environment facilitates collaboration and co operation and encourages the team members to work with a common purpose and shared goals and thus achieve the desired results [4, 5, 6, 9, 10, 12].

Improving communication and trust in GSD is important and KM is considered as the best means towards this end as KM holds a central role in the success of GSD projects [6, 14]. Trust has a direct relationship with knowledge and could not occur without it [6, 13, 15, 16]. There is a need to use the latest information and communication technologies of knowledge sharing for building trust [9, 16, 17, 18]. Sometimes a small amount of missing knowledge causes a delay and even the failure of the software project [6].

Therefore, keeping in view the importance of KM and Trust for GSD based organizations; in this paper we will study the impact of KMR on building trust. KMR is an information technology tool that promotes knowledge sharing among GSD teams and it is one of the commonly used practices of KM as discussed in literature [17, 18, 19, 20, 21, 22, 23, 24].

In the next section, we define the relation between knowledge management and trust in the light of literature, and we elaborate on the importance of KM and trust in GSD projects. In sections III and IV the project overview and research methodology are described. In section V, we present the findings and assessments of our data.Section VI summarizes the main lessons learned from this research and finally we conclude with suggestions for future research.

II. BACKGROUND AND RELATED WORK

GSD team members are unlikely to meet face-to-face due to the nature of collaboration in a multinational organization and cost saving strategies. Establishing and maintaining trust in GSD teams improve their effectiveness because team members are generally not able to spend much time in cross checking and monitoring others.Positive interpersonal relationships and shared experience help in building trust [4, 13, 16].

Knowledge management holds a central place in the development of trust among GSD team members. To develop trust in GSD, there is a need to promote affective communication, adopting knowledge sharing practices, developing strong leadership and knowing and building relationships between individuals [10, 16, 25, 26]. Missing trust sometimes leads to the termination of further co-operation and relationships[25].

A survey was performed by NUS in 2007 in which more than 30 nations were involved and the results obtained from this survey point to an opportunity for governments from developing nations to use KM as a key driver towards increasing public sector productivity and building trust in government [14].

Information and communication technologies are considered as the best means for improving communication and building trust among GSD teams [9]. An intra-organizational tool named “trusty” is suggested for the development and improvement of trust among GSD teams in

[9] but no validation is provided in this study. The study claims that this tool facilitates communication and cooperation among GSD team members by providing them with a knowledge exchange platform.

GSD teams require intensive communication and collaboration for the development of trust. KMR is an easy way of managing and sharing knowledge across the organization and allowing stakeholders to know when, how or by whom knowledge is done. KMR thus help in improving the coordination and trusting working relationship among team members[23, 26].

III. RESEARCH METHODOLOGY

Research question addressed in this study is

Q:How KMR helps in building and maintaining trust among GSD team members?

In order to study the impact of KMR in building trust, we have carried out a controlled experiment. The selected context of our experiment was an academic environment. We conducted our study with two groups of students each consisting of six members, these students belong to two different universities located in Pakistan and China. The reason for choosing the students from these two universities is due to the fact that this was the first experiment that we carried out, and the authors work at these universities. So, it was easy to carry out experiment with these two universities instead of others, due to regulations and difficulties involved in obtaining permission. Moreover the variables of this choice were suitable for our experiment as we need two GSD teams where the cultural, linguistic and temporal difference is involved. Each group consists of three Pakistani and three Chinese students. Both groups had to complete the similar project of evaluating and redesigning a website and the duration of the project was three months for each group. All the students who participated in this experiment were the student of BS computer science, so the age and experience of the participants was almost same.

Existing wiki software was used as KMR after making few modifications into it. The information that students consider to be important with respect to their remote colleagues were added into this KMR so that they may know about their colleagues. This KM repository provides a space to collaborate and share projects, documents, messages, schedules, tasks and contacts within the group and many other features that help in project management and coordination.

However the access of this KMR was given to only one group of students and not the other so that the impact of KMR in building trust may be studied. We use the name **Group A (just for ease of use)** for the team who was using KMR and **Group B** for the team who was not using KMR.

In order to measure trust four indicator/measures of trust were used based on prior deconstruction of team trust in literature [27]. We describe each measure briefly in turn

A. Propensity to trust

It is the willingness of one or more person in a group to trust others. Propensity to trust is affected by many factors like team culture, lifestyle, experience, education etc. it's a general personality attribute that leads towards the general expectations about the credibility and trustworthiness of other person which remains stable across many situations. [28].

B. Perceived trustworthiness

It refers to the extent to which an individual expect others to behave according to their commitments. It exists when the team members behave according to the expectations of their colleagues; they are loyal and honest with their team members and no body takes advantage of the other [28].

C. Cooperative behaviors

It refers to the environment in which team members work with collaboration, help others in difficult situations and share their experiences and knowledge. A team with cooperative behavior works efficiently towards a common goal.

D. Monitoring behavior

It refers to the extent to which team members monitor and check the actions of their team mates. Literature argues that monitoring is associated with lack of trust. This behavior decrease the efficiency and performance of the team so it should be avoided especially in GSD teams where a huge geographical distance is involved and monitoring affects not only trust and performance of GSD teams but it also causes budget overrun[28].

Based on the above four measures, literature was studied and 28 items were selected that were related to these four measures of trust. These items were checked by two independent subject matter experts. These experts evaluated these items according to the criteria of understandability, length and redundancy. Few items were discarded because of the redundancy and some more items were added, so finally after evaluation 21 items were selected. From these 21 items; 6 items were related to the propensity to trust, 6 items were related to the perceived trustworthiness, 6 items were related to the cooperative behavior and 3 items were related to the monitoring behaviors. From these four measures, the first three measures propensity to trust, perceived trustworthiness and cooperative behaviorwhich have a positive impact on trust while the monitoring behaviorwhich have a negative impact on trust. Using these 21 items a questionnaire was prepared. Responses on the trust scales were given on a 5-point scaling ranging from 5= “strongly agree” to 1= “strongly disagree”.

As trust takes time in building and as it changes with the passage of time, so during the three months duration of the project, ***Group A*** and ***Group B*** team members were asked to fill in the questionnaire three times each after the gap of one month based on their mutual understanding about their team mates. However, the strict confidentiality of their responses was ensured before giving them a questionnaire. The six Chinese students (three from G***roup A*** and three from G***roup B***) filled the questionnaire in the presence offirst researcher while the students in Pakistani University were asked to email the questionnaire to the researcher.

IV. DATA COLLECTION AND RESULTS

After the completion of first questionnaire filling exercise, 12 questionnaires were received, six from each group A and B respectively. An aggregated questionnaire was prepared in which against each question the total was calculated (where total= number of participants who are strongly agree or agree with the statement of the questionnaire). Then these scores were again aggregated on the basis of four measures of trust and the results obtained are shown in Table1.

TABLE 1: AGGREGATE LEVEL OF TRUST FOR BOTH GROUP A AND GROUP B AT FIRST STAGE (WHRE PK :PAKSTANI STUDENTS AND CN: CHINESE STUDENTS)

Measurement Factors	**Agreement percentage** (strongly agree+ agree)			
	Group A		**Group B**	
	PK	**CN**	**PK**	**CN**
Propensity to trust	13	14	11	12
Perceived trustworthiness	13	15	12	12
Cooperative behavior	14	15	10	12
Monitoring behavior	5	5	7	6

The same procedure was applied for combining the results of exercise two and three. The overall results in allthree stages of this experiment were obtained and are shown in Table2

Table2 shows that the values of the first three measures namely propensity to trust, perceived trustworthiness and cooperative behavior are comparatively high in ***Group A*** as compared to ***Group B*** while the monitoring behavior in ***Group A*** is low as compared to ***Group B***. This indicates that the level of trust among team members of***Group A*** is comparatively high as compared to ***Group B.***

Figure1-4 shows us the values of four measures of trust during three stages of the projects A and B. In these figures three questionnaire filling exercises are shown along X-axis from one to three and the value of trust indicator is shown along Y-axis. As the total students in each group were six therefore maximum value of Y can be 30 in case if all the students are strongly agree about some questionnaire statement and minimum value can be 5 if all the students are disagree on a point. Therefore the scale range is from 5 to 30 for each measure of trust.

TABLE 2: OVERALL RESULTS OBTAINED FROM BOTH GROUP A AND GROUP B (WHERE SA: STRONGLY AGREE, A: AGREE, PK: PAKISTANI STUDENTS, CN: CHINESE STUDENT & GRP: GROUP)

Measurement Factors			Propensity to trust	Perceived trustworthiness	Cooperative behavior	Monitoring behavior
Agreement %age of Trust (SA+A) Stage 1	GRP **A**	PK	13	13	14	5
		CN	14	15	15	5
	GRP **B**	PK	11	12	10	7
		CN	12	12	12	6
Agreement %age of Trust (SA+A) Stage 2	GRP **A**	PK	14	15	16	5
		CN	15	16	15	4
	GRP **B**	PK	12	13	12	6
		CN	12	14	13	6
Agreement %age of Trust (SA+A) Stage 3	GRP **A**	PK	16	16	17	4
		CN	17	16	16	3
	GRP **B**	PK	13	14	14	6
		CN	14	12	13	5

Results displayed in Fig.1-4 reflect the positive impact of KMR in building trust among GSD team members. We further validated our results by conducting an open ended discussion with both Groups after the end of the project. Team members from ***Group A*** were asked that how much this KMR was helpful for them. Reply of almost 67% students waspositive. A student from ***Group A*** said that " *KMR provide us a platform through which we can discuss everything and even when we use this KMR it doesn't seem to us that a huge geographical distance is involved between our team mates"*. Moreover, when these team mates were asked about the role of KMR in building trust the answer of almost every participant was positive.

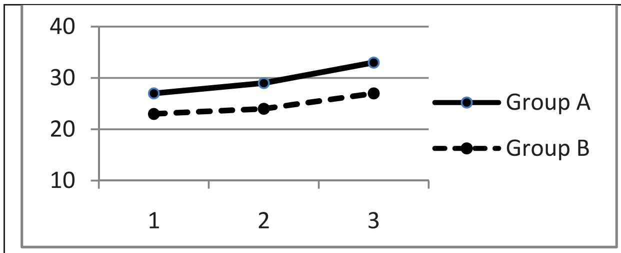


Figure 1: Results of Propensity to trust for Group A and Group B during three stages of the projects

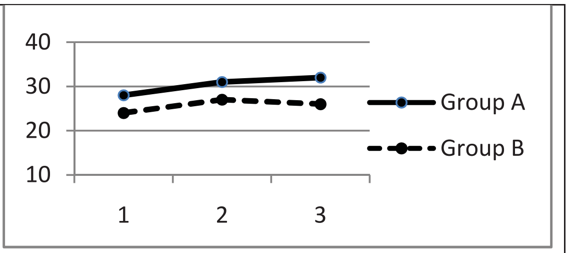


Figure 2: Results of Perceived trustworthiness for Group A and Group B during three stages of the project

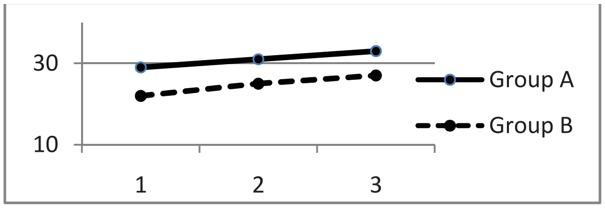


Figure 3: Results of Cooperative behavior for Group A and Group B during three stages of the project

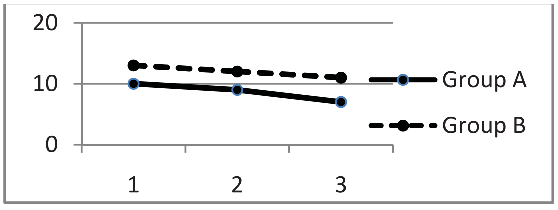


Figure 4: Results of Monitoring Behavior for Group A and Group B during three stages of the project

One of the students from ***Group B*** told us that although there exist many communication software but we cannot monitor and control our projects through them, there must be some common software which provides us a platform for every kind of formal and informal communication. Further he said that change in management and project tracking in a globally distributed team environment is especially difficult to manage without the presence of such a platform.

V. CONCLUSION AND FUTURE WORK

To conclude, trust is one of the important factors in the success of GSD projects. The 21- item measures indicated in this study provides an insight into the organization about the level of trust existing among the teams. Moreover, the major result of the study shows us that there must be knowledge management and a sharing mechanism like the KMR in this experiment. This KMR helps the teams in building and maintaining trust by providing them with a platform for communication and discussion. The KMR if implemented and used properly helps the team members in resolving conflicts, propagating the changes among teams and maintaining informal communication. As a results of all these, trusting working relationship among team members increase and the projects are completed successfully.

It was a controlled experiment performed with two Groups of students. In the future, there is a need to implement this KMR in a real GSD organization to study the impact of this KMR on trust by using the same 21-items measures of trust.

ACKNOWLEDGEMENETS

We would like to thank all the students who participated in our experiment and made this experiment successful.

REFERENCES

[1] Herbsleb. James. Global software engineering: the future of socio-technical coordination. *Future of Software Engineering*. 2007, pp.23-25.

[2] Eoin. Conchuir, Helena. Holmstrom, Par. Agerfalk and Brian. Fitzgerland. Exploring the Assumed Benefits of Global Software Development. *ICGSE'06.* 2006, pp. 159-168.

[3] Damian. D,and Moitra. D. Global Software Development: How Far Have We Come? *IEEE software.2006, 23*(5)*,* pp. 17-19.

[4] Ani, Al. Ban, Wilensky. H, Redmiles. D,Simmons. E. An Understanding of the Role of Trust in Knowledge Seeking and Acceptance Practices in Distributed Development Teams. . *ICGSE'11.* 2011, pp. 25-34.

[5] Sami. Haq. Mushtaq. Raza, Asraf. Zia and Ahmed. Khan. Issues in Global Software Development: A Critical Review,*J. Software Engineering & Applications,* 2011, 4, pp. 590-595.

[6] Paivi. Parviainen, Maarit. Tihinen. Knowledge-related challenges and solutions in GSD. *Expert Systems a Journal of Knowledge Engineering*. 2011.

[7] Emam, hossain, Paul. Bannerman and D. Jaffery. Scrum Practices in Global Software Development: A Research Framework, D. Caivano et al. (Eds.): PROFES 2011, LNCS 6759, pp. 88–102.

[8] Ángel. García-Crespo, Ricardo. Colomo-Palacios, Pedro. Soto-Acosta and Marcos. Ruano-Mayoral . Qualitative Study of Hard Decision Making in Managing Global Software Development Teams, Information Systems Management.2010, 27:3, pp. 247-252.

[9] Gabriela. Aranda, Aurora. Vizcaíno, José. Luís-Hernández, Ramón. Palacio and Alberto. Morán. Trusty: A Tool to Improve Communication and Collaboration in DSD. A.S. Vivacqua, C. Gutwin, and M.R.S. Borges (Eds.): CRIWG 2011, LNCS 6969, pp. 224–231.

[10] Casey. Valentine. Developing Trust in Virtual Software Development Teams. *Journal of Theoretical and applied commerce Research*.2010, 5(2), pp.41-58.

[11] John. Noll, Saarah. Beecham and Ita. Richardson. Global Software Development and Collaboration: Barriers and Solutions. 2010. ACM Inroads, pp.66-78.

[12] Samireh. Jalali, Cigdem. Gencel and Darja. Smite. Trust Dynamics in Global Software Engineering. *ESEM'10, September 2010,* pp. 16-17 Bolzano-Bozen, Italy.

[13] Nils. Brede and Darja. Smite. Understanding a Lack of Trust in Global Software Teams: A Multiple -case Study. 2008. *Software Process Improvement and Practice.* 2008, **13(3)**, pp. 217–231

[14] Yuen, Hui. Yum. Overview of the knowledge management in the public sector. *7th Global Forum on Reinventing Government*: *Building Trust in Government Workshop on managing Knowledge to Build Trust in Government*.2008.

[15] Fares. Anwar, Rozilawati. Razali and Ahmad. Kamsuriah. Achieving Effective Communication during requirements Elicitation - A Conceptual Framework. *ICSECS* .2011. Part III, CCIS 181, pp. 600–610.

[16] Julia. Kotlarsky, Paul c. van Fenemaand Lesle P. willcocks. Developing a knowledge-based perspective on coordination: The case of global software projects. *Information & Management 45.*2008, pp. 96–108.

[17] Muneera. Bano&Naveed. Ikram. KM-SORE: Knowledge Management for Service Oriented Requirements Engineering. Copyright (c) IARIA, 2011.

[18] V.Clerc. Towards Architectural Knowledge Management Practices for Global Software Development. *Third ICSE Workshop on Sharing and Reusing architectural Knowledge (SHARK'08), Leipzig, Germany.2008*, pp. 23-28.

[19] Daniela. Damian and Didar. Zowghi. Requirements Engineering challenges in multi-site software development organizations*. Requirements Engineering Journal*, *8.*2003, pp. 149-160.

[20] Christof. Ebert and Philip. DeNeve. Surviving Global Software Development. *IEEE Software*, 18(2) .2001, pp. 62-69.

[21] Clerc. Viktor, Lago. Patricia and Vliet. Hans. The Usefulness of Architectural Knowledge Management Practices in GSD. Fourth *IEEE International Conference on Global Software Engineering*.2009, pp. 73-82.

[22] Nguyen. Tracey, Smyth. Robert & Gable. Guy. Knowledge Management Issues and Practices: A Case Study of a Professional services Firm. Fifteenth Australian conference on information system. 2004.

[23] Nour. Ali, Sarah. Beechmanand Mistrik. Ivan. Architectural Knowledge Management in Global Software Development: A Review. *International Conference on Global Software Engineering* .2010, pp.347-352.

[24] Sarah. Beechman, John. Noll, Ita. Richardson and Nour. Ali. Crafting a Global Teaming Model for Architectural Knowledge. *International Conference on Global Software Engineering* .2010, pp.55-63.

[25] Darja. Smite, Nils Brede. Moe and Richard Torkard. Pitfalls in Remote Team Coordination: Lessons Learned from a Case Study. Lecture Note in computer Science,2008, volume 5089/2008, pp. 345–359.

[26] Wei. Xiao and Qing-qi. Wei. A Study on Virtual Team Trust Mechanism and Its Construction Strategies. *International Conference on Information Management, Innovation Management and Industrial Engineering*.2008, pp. 315-319.

[27] Ana Cristina. Costa and Neil. Anderson. Measuring trust in teams: Development and validation of a multifaceted measure of formative and reflective indicators of team trust. European Journal of Work and Organizational Psychology. 20(1). 2001, pp. 119-154.

[28] Roger C. Mayer and James H. Davis and David. Schoorman. An integrative model of organizational trust. Academy of Management Review, 20(3). 1995, pp. 709–734.

Modeling and Analysis of Switched Fuzzy Systems

Zuohua Ding, Jiaying Ma
Lab of Scientific Computing and Software Engineering
Zhejiang Sci-Tech University
Hangzhou, Zhejiang 310018, P.R. China

Abstract

Switched fuzzy systems can be used to describe the hybrid systems with fuzziness. However, the languages to describe the switching logic and the fuzzy subsystems are in general different, and this difference makes the system analysis hard. In this paper we use Differential Petri Net (DPN) as a unified model to represent both the discrete logic and fuzzy dynamic processes. We then apply model checking technique to DPN to check the correctness of the requirements.

1 Introduction

Switched systems have been widely used in a variety of industrial processes. If a system is too complex or ill-defined, i.e. the system contains fuzzyness, switched fuzzy system has been adopted to model such systems[1].

Compared with conventional switched system modeling, switched fuzzy system modeling is essentially a multi-model approach in which simple sub-models (typically linear models) are fuzzily combined to describe the global behavior of a nonlinear system. A typical sub-model is Takagi and Sugeno (T-S) fuzzy model, which consists of a set of If-Then rules and a set of ordinary differential equations[5].

This model raises some issues in the software engineering. For example, in the T-S fuzzy model, the languages to describe the switching logic and the fuzzy subsystems are in general different, and this difference makes the system analysis and implementation hard. So far, the efforts to address this issue are only for the deterministic switched systems.

It is our attempt to solve this issue for switched fuzzy systems. We use Differential Petri net defined by Demongodin and Koussoulas[2] as a unified behavior model to represent switched fuzzy systems. We then use model checking tool HyTech to check the DPN, and use an enhanced version of the tool Visual Object Net++ to simulate DPN. We employ the the Differential-Drive Two-Wheeled Mobile Robots as the running example to illustrate our method.

2 Switched Fuzzy Systems

A Switched Fuzzy System (SFS) consists of a family of T-S fuzzy models and a set of rules that orchestrates the switching among them. Assume that the Switched Fuzzy System has m subsystems, each is described by a Takagi-Sugeno fuzzy model[5], and $\sigma : R^+ \to M = \{1, 2, \ldots, m\}$ is a piecewise constant function that represents the switching signal.

2.1 Takagi-Sugeno Fuzzy Model

Let $N_{\sigma(t)}$ be the number of inference rules. Then the T-S fuzzy model is described as follows:

$$\begin{array}{rcl} & & \text{[Local Plant Rule]} \\ R^l_{\sigma(t)} & : & \text{if } \xi_1 \text{ is } M^l_{\sigma(t)1} \wedge \cdots \wedge \xi_p \text{ is } M^l_{\sigma(t)p}, \\ & & \text{then } x'(t) = A_{\sigma(t)l}x(t) + B_{\sigma(t)l}u_{\sigma(t)}(t), \\ & & l = 1, 2, \ldots, N_{\sigma(t)}. \end{array}$$

In this model, $R^l_{\sigma(t)}$ denotes the l^{th} inference rule, $u_{\sigma(t)}$ is the input variable, vector $x(t) = [x_1(t), x_2(t), \ldots, x_n(t)]^T \in R^n$ represents the state variables, vector $\xi = [\xi_1, \xi_2, \ldots, \xi_p]$ represents the vector of rule antecedents (premises) variables, and matrices $A_{\sigma(t)l} \in R^{n\times n}$ and $B_{\sigma(t)l} \in R^{n\times p}$.

Hence, the i^{th} subsystem can be represented as follows:

$$\text{subsystem } i : \left\{ \begin{array}{l} R^1_i\text{: if } \xi_1 \text{ is } M^1_{i1} \wedge \cdots \wedge \xi_p \text{ is } M^1_{ip} \\ \qquad \text{then } x'(t) = A_{i1}x(t) + B_{i1}u_i(t) \\ R^2_i\text{: if } \xi_1 \text{ is } M^2_{i1} \wedge \cdots \wedge \xi_p \text{ is } M^2_{ip} \\ \qquad \text{then } x'(t) = A_{i2}x(t) + B_{i2}u_i(t) \\ \vdots \\ R^{N_i}_i\text{: if } \xi_1 \text{ is } M^{N_i}_{i1} \wedge \cdots \wedge \xi_p \text{ is } M^{N_i}_{ip} \\ \qquad \text{then } x'(t) = A_{iN_i}x(t) + B_{iN_i}u_i(t) \end{array} \right.$$

By using the center of gravity method for defuzzification, the global model of the i^{th} fuzzy subsystem can be described by the equation:

$$x'(t) = \sum_{l=1}^{N_i} \eta_{il}(\xi(t))(A_{il}x(t) + B_{il}u_i(t)),$$

where

$$\eta_{il}(t) = \frac{\prod_{\rho=1}^{n} \mu_{M_\rho^l(t)}}{\sum_{l=1}^{N_i}\prod_{\rho=1}^{n} \mu_{M_\rho^l(t)}}, 0 \le \eta_{il} \le 1, \sum_{l=1}^{N_i} \eta_{il}(t) = 1,$$

and $\mu_{M_\rho^l(t)}$ denotes the membership function of the fuzzy state variable x_ρ that belongs to the fuzzy set M_ρ^l.

2.2 Switching Logic For $\sigma(t)$

Let X be the universe of discourse. Assume that X is partitioned to m parts, i.e. there are $X_i \in R^{1\times n}, i = 1, 2, \ldots, m$ such that $X = X_1 \cup X_2 \cup \cdots \cup X_m$. We also assume that X_i is associated with subsystem i. The switching is based on the *Region Rule*, which is defined as the following:

[Region Rule]
If ξ_1 is Ω_{i1} and ξ_2 is Ω_{i2} and ... and ξ_p is Ω_{ip}
Then [Local Plant Rule]

where Ω_{ij} are crisp sets, $X_i = \Omega_{i1} \times \Omega_{i2} \times \cdots \times \Omega_{ip}$, and

$$\Omega_{ij}(\xi_j) = \begin{cases} 1, & \xi_j \in \Omega_{ij}; \\ 0, & \text{otherwise.} \end{cases}$$

Thus at time t, if the state variable x has value $x(t) = (x_1(t), x_2(t), \ldots, x_p(t), \ldots, x_n(t))$ such that $x_1(t) \in \Omega_{i1}$ and $x_2(t) \in \Omega_{i2}$ and ... and $x_p(t) \in \Omega_{ip}$, then $\sigma(t) = i$.

If we regard that $x(\in X_i)$ as a state, then our switching is actually a Finite State Machine (FSM) based switching.

2.3 A Running Example

We employ the the differential-drive two-wheeled mobile robots (TWMR) as the example to illustrate our method. Based on the design control, TWMR can move on a reference trajectory. Figure 1 pictures the movement. In the figure, variable y represents hight of the rear axle and the variable θ specifies the angle of the robot orientation in a reference frame. Both are the functions of time t.

In order to simply and effectively control the nonlinear dynamics, the authors in paper[4] introduced the switched T-S fuzzy model. Based on the values d of the premise variable, the premise variable space are partitioned into three regions. In each region, the local nonlinear dynamic is represented by a T-S fuzzy model. The switched T-S fuzzy model is described as follows:

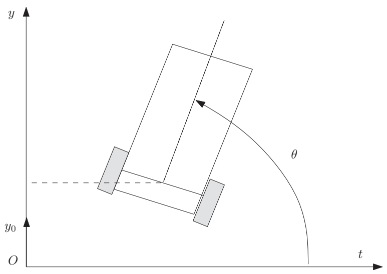


Figure 1. The model for TWMR.

[**Region Rule 1**]: If $d < \theta(t) \le 3.131$, then
Local Plant Rule 1 : If $\theta(t)$ is $h_{11}(\theta(t))$,
then $x'(t) = A_{11}x(t) + B_{11}u(t)$;
Local Plant Rule 2 : If $\theta(t)$ is $h_{12}(\theta(t))$,
then $x'(t) = A_{12}x(t) + B_{12}u(t)$.
[**Region Rule 2**]: If $-d \le \theta(t) \le d$, then
Local Plant Rule 1 : If $\theta(t)$ is $h_{21}(\theta(t))$,
then $x'(t) = A_{21}x(t) + B_{21}u(t)$;
Local Plant Rule 2 : If $\theta(t)$ is $h_{22}(\theta(t))$,
then $x'(t) = A_{22}x(t) + B_{22}u(t)$.
[**Region Rule 3**]: If $-3.131 \le \theta(t) < -d$, then
Local Plant Rule 1 : If $\theta(t)$ is $h_{31}(\theta(t))$,
then $x'(t) = A_{31}x(t) + B_{31}u(t)$;
Local Plant Rule 2 : If $\theta(t)$ is $h_{32}(\theta(t))$,
then $x'(t) = A_{32}x(t) + B_{32}u(t)$.

3 Mapping From SFS To Petri net

Without specifying, the basic symbols used for the Petri net construction come from Reisig[3]. We will map SFS to Differential Petri Net (DPN) defined by Demongodin and Koussoulas[2]. Through the introduction of the differential place, the differential transition, and suitable evolution rules, DPN is possible to model concurrently discrete-event processes and continuous-time dynamic processes, represented by systems of linear ordinary differential equations. Full development of the DPN model can be found in [2].

3.1 Mapping Rules For Linear Systems

Without loss of generality, we assume that the linear systems are $x'(t) = Ax(t) + Bu(t)$, where

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}, B = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix},$$

and $u(t)$ is the input to the system.

The Petri net design of the system is shown in Figure 2. There are two parts in the net: The first part is for the continuous part(left) and the second part is the input part(right).

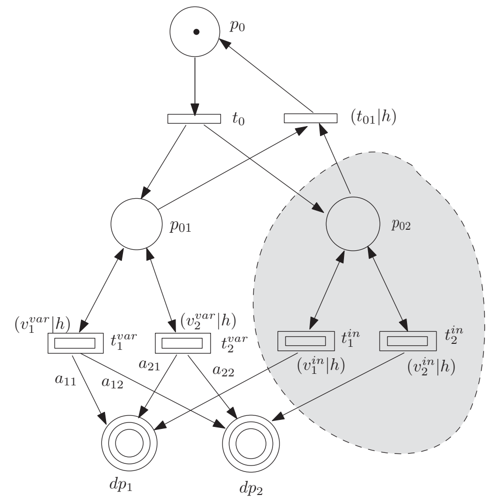


Figure 2. The Petri net of $x'(t) = Ax(t) + Bu(t)$**.**

In the figure, places dp_1 and dp_2 are two differential places that represent the state variables x_1 and x_2, respectively. Transitions t_1^{var} and t_2^{var} are two differential transitions, both are associated with a delay h through a discrete transition that connected to it. The firing speeds of differential transitions t_1^{var} and t_2^{var} depend on the values of x_1 and x_2 and can be represented as the follows:

$$\begin{aligned} v_1^{var}(t) &= a_{11}x_1(t) + a_{12}x_2(t), \\ v_2(t)^{var} &= a_{21}x_1(t) + a_{22}x_2(t). \end{aligned}$$

3.2 Mapping Rules For The Switching Logic

Without activity coordination, a Petri net is the same as a finite state machine. Thus the finite state machine can be regarded as a special case of Petri net. Accordingly, the mapping rules from finite state machine to Petri net are straightforward, and the mapping rules are listed in the Figure 3. In the figure, the second column displays the structures of the FSM and the third column display the structures of the corresponding Petri net structures.

3.3 Petri Net Representation of TWMR

Based on the above rules, we build the DPN for TWMR. The net is built by an enhanced version of the tool Visual Object Net++[1], which is shown in Figure 4. In the figure, the state variables y and θ are represented by places dp_1 and dp_2 respectively. Three regions Region 1, Region 2 and Region 3 are represented by places p_1, p_2 and p_3, respectively. Transition t_1 has condition $d < \theta(t) \leq 3.131$, transition t_2 has condition $-d \leq \theta(t) \leq d$, and transition t_3 has condition $-3.131 \leq \theta(t) < -d$.

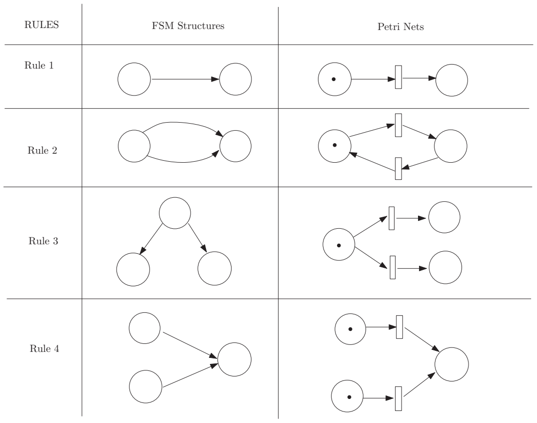


Figure 3. From FSM to Petri nets.

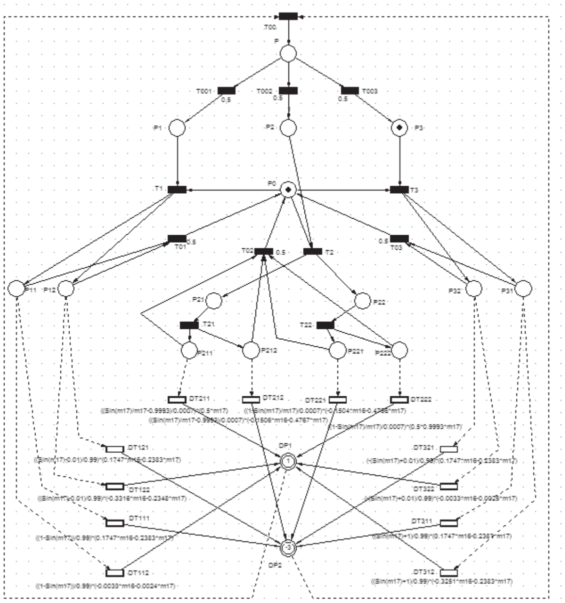


Figure 4. DPN for two-wheeled mobile robots.

For the simulation, we choose three different initial values for variable x, $x(0) = (y_0, \theta_0)$, and they are $(y_0, \theta_0) = (1, \frac{\pi}{52})$ which is located in Region 2, $(y_0, \theta_0) = (1, \frac{\pi}{2})$ which is located in Region 1, $(y_0, \theta_0) = (1, -3)$ which is located in Region 3. The simulation shows that in each case, we have that $\lim_{t\to\infty} y(t) = 0$ and $\lim_{t\to\infty} \theta(t) = 0$, which indicate that the control purpose is reached.

[1] http://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/db.html

4 Model Checking DPN

We use HyTech[2] to check the DPN model. HyTech is a symbolic model checker for linear hybrid automata.

4.1 From DPN To Hybrid Automata

In order to perform model checking with HyTech, we need to transform the Differential Petri net to a Hybrid Automaton (HA). The transformation rules are omitted here. After transformation, the Hybrid Automaton of DPN of TWMR, which is nonlinear, is shown in Figure 5.

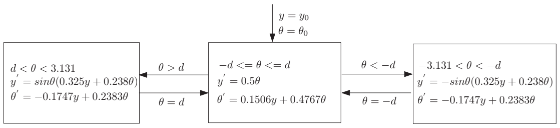

$y = y_0$
$\theta = \theta_0$
$d < \theta < 3.131$
$y' = sin\theta(0.325y + 0.238\theta)$
$\theta' = -0.1747y + 0.2383\theta$
$\theta > d$
$\theta = d$
$-d <= \theta <= d$
$y' = 0.5\theta$
$\theta' = 0.1506y + 0.4767\theta$
$\theta < -d$
$\theta = -d$
$-3.131 < \theta < -d$
$y' = -sin\theta(0.325y + 0.238\theta)$
$\theta' = -0.1747y + 0.2383\theta$

Figure 5. Nonlinear hybrid automaton of TWMR.

4.2 Linearizing Nonlinear Hybrid Automata

We need to linearize nonlinear hybrid automata to linear hybrid automata (LHA). In this paper, we adopt *linear phase portrait approximation* replacing nonlinear predicates by more relaxed linear predicates. For the automaton of TWMR, we split each space of state variable into smaller intervals in which the time derivatives are bounded, and the resulting LHA model is shown in Figure 6.

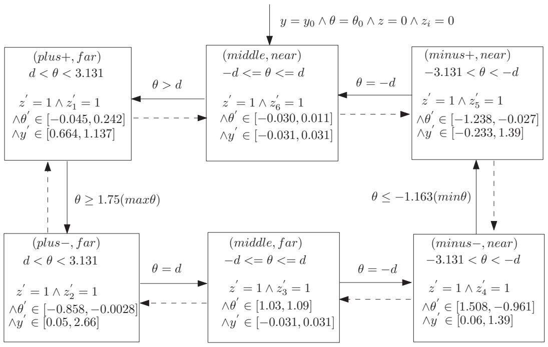

$y = y_0 \wedge \theta = \theta_0 \wedge z = 0 \wedge z_i = 0$
$(plus+, far)$
$d < \theta < 3.131$
$z' = 1 \wedge z_1' = 1$
$\wedge\theta' \in [-0.045, 0.242]$
$\wedge y' \in [0.664, 1.137]$
$\theta > d$
$(middle, near)$
$-d <= \theta <= d$
$z' = 1 \wedge z_6' = 1$
$\wedge\theta' \in [-0.030, 0.011]$
$\wedge y' \in [-0.031, 0.031]$
$\theta = -d$
$(minus+, near)$
$-3.131 < \theta < -d$
$z' = 1 \wedge z_5' = 1$
$\wedge\theta' \in [-1.238, -0.027]$
$\wedge y' \in [-0.233, 1.39]$
$\theta \geq 1.75(max\theta)$
$\theta \leq -1.163(min\theta)$
$(plus-, far)$
$d < \theta < 3.131$
$z' = 1 \wedge z_2' = 1$
$\wedge\theta' \in [-0.858, -0.0028]$
$\wedge y' \in [0.05, 2.66]$
$\theta = d$
$(middle, far)$
$-d <= \theta <= d$
$z' = 1 \wedge z_3' = 1$
$\wedge\theta' \in [1.03, 1.09]$
$\wedge y' \in [-0.031, 0.031]$
$\theta = -d$
$(minus-, near)$
$-3.131 < \theta < -d$
$z' = 1 \wedge z_4' = 1$
$\wedge\theta' \in [1.508, -0.961]$
$\wedge y' \in [0.06, 1.39]$

Figure 6. Linear hybrid automaton of TWMR.

4.3 Requirement Specification

Reachable requirement can be specified by state assertion *reach*. If a system fails to satisfy a correctness requirement, then HyTech generates an error trajectory, which illustrates a time-stamped sequence of events that leads to a violation of the requirement.

One of the specification requirements of TWMR is specified as: TWMR can arrive at the position $y = 0$, $\theta = 0$. The output of the execution of Hytech is displayed in the follows:

```
Number of iterations required for reachability: 5
Location: s6
  100y+7>=0 & 2500x<=157 & 2500x+157>=0 & 100y<=7
Location: s5
  100y>=7 & 1000x+1163>=0 & 2500x+157<=0 & 2y<=11
Location: s4
  20y<=121 & 1000x+1163>=0 & 2500x+157<=0 & 2y>=11
Location: s3
  2500x<=157 & 2500x+157>=0 & 20y=121
Location: s2
  y = 3 & 2x = 3
can reach
```

From the output, we see that the trajectory of the TWMR is $s_2 \rightarrow s_3 \rightarrow s_4 \rightarrow s_5 \rightarrow s_6$, and the last line is 'can reach'. Thus the required specification is satisfied.

5 Conclusion

In this paper Differential Petri Net has been used to model switched fuzzy systems. This model combines fuzzy dynamic and discrete logic events in a single net. Thus we can use tool Visual Object Net++ to simulate the switched fuzzy system, and use tool HyTech to model check the correctness of the system requirements. In the future, we will consider to represent a hybrid system with a unified model and apply model checking technique to the unified model.

References

[1] J. B. Coulaud, G. Campion, G. Bastin, and M. De Wan, Stability Analysis of a Vision-Based Control Design for an Autonomous Mobile Robot, *IEEE Transactions on Robotics*, vol.22, no.5, pp.1062-1069, 2006.

[2] I. Demongodin and N. T. Koussoulas, Differential Petri nets: Representing continuous systems in a discrete-event world, *IEEE Transactions on Automatic Control*, vol.43, no.4, pp.573-579, 1998.

[3] W. Reisig, Petri Nets, Springer-Verlag, 1985.

[4] K. Tanaka, M. Iwasaki, H. O. Wang, Stable switching fuzzy control and its aplication to a hovercrsft type vehicle, *Proceedings of* 9^{th} *International Conference on Fuzzy Systems*, vol.2, pp.804-809, 2000.

[5] T. Takagi, M. Sugeno, Fuzzy identification of systems and its applycations to modeling and control, *IEEE Transactions on System Man and Cybernetics*, vol.15, no.1, pp.116-132, 1985.

[2] http://embedded.eecs.berkeley.edu/research/hytech/

An Empirical Study on Recommendation Methods for Vertical B2C E-commerce

Chengfeng Hui, Jia Liu*, Zhenyu Chen, Xingzhong Du, Weiyun Ma
State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China
Software Insititute, Nanjing University, Nanjing, China
*liujia@software.nju.edu.cn

Abstract

Recommender systems have been already performing well in large comprehensive E-commerce sites. Another trend emerging in E-commerce area is vertical B2C sites. The vertical B2C sites sell only one or a few of categories of goods to end users, and most of the users are new users. There may be not sufficient history data to generate high-quality recommendation because the traffic of these sites are low. To analyze the feasibility and usability of popular recommendation methods (e.g. collaborative filtering, content-based, etc.) in vertical B2C sites, we have been working in collaboration with an E-commerce site to gather real data from an actually running vertical B2C site. In this paper, we evaluate the performance of different recommendation methods over half a year period from December 2010 to June 2011. We analyze both the performance and cost of these recommendation methods, and experimental results show that we should apply suitable methods based on the available data.

Keywords: Vertical E-commerce, Collaborative Filtering, Content-based Recommendation, Recommender System

1. Introduction

E-commerce [6] has been widely used to perform business transactions. One fast growing subcategory of E-commerce is vertical B2C, and it is becoming more and more attractive to end users. Unlike mass online merchants (e.g. Amazon, Walmart), vertical B2C sites focus on one single or a small number of categories of products (e.g. Newegg sells computers/electronics, Netflix sells Books/Music/Movies), they can provide more various and comprehensive choices in a particular field.

Vertical B2C E-commerce sites are becoming very popular, but there is little, if any publication directly evaluating the feasibility of popular recommendation algorithms on vertical B2C sites, so we want to analyze the performance of different recommendation methods in such systems. Fortunately we have the opportunity to collaborate with an E-commerce site to gather real data from currently running site and conduct our case study.Vertical B2C E-commerce sites do have some features that will limit the application of popular recommendation methods. For content-based method, the most obvious limitation is items must be capable of being described as features, if there is no or limited feature information about items, it is hard for content-based techniques to get a good performance. For collaborative filtering, one precondition is there are sufficient ratings to items given by users or something equivalent. But unlike data sets (e.g. Movielens, Eachmovie, etc.) which we usually evaluate collaborative filtering on, it is very likely that there are no rating information on vertical B2C sites. Another problem is most users of vertical B2C E-commerce sites are new users, which will lead to the lack of sufficient history behavior data of one user to generate his or her profile.

2. Recommender System

The origin of recommender systems can be traced back to approximation theory, information retrieval and forecasting theories [1]. The appearance of first papers about collaborative filtering in the mid-1990s [3] make recommender systems become an popular research direction. And recent explosively development of Internet further increases the interest in this domain both in the industry and academia. The most famous recommender system applied in industry is probably the book recommender system of Amazon (www.amazon.com). This system records users' purchase, explore, comment and rating data to recognize users' preference and then recommends products to users. The biggest news in academia is maybe in 2009 Netflix (www.netflix.com) which is a large DVD rental service company announced to award 1 million dollar prize to a team that could increase the accuracy of rating prediction by 10%.The BellKor's Pragmatic Chaos team eventually won the prize with their BigChaos solution [8].

Recommender systems are usually classified into two categories. The first category is content-based recommendations [2], and the second category is collaborative filtering recommendations [7]. Content-based methods utilize the content information of items to generate recommendations. Content information means the intrinsic features of items. In movie recommendation scenario, the content information of a movie is mainly its title, director, actors, genre, plot summary, keywords etc. The first step of content-based methods is to figure out the commonalities among the movies user has given high rating. Then the system will generate the user's preference with these commonalities, and the recommended movies will be apparently those ones similar to the user's preference. Collaborative filtering methods utilize users' rating information to generate recommendation. According to the way of using the rating information, collaborative filtering methods can be classified into two categories: Memory-based CF and Model-based CF. Memory-based CF directly makes use of entire or part of rating information to generate recommendation. Model-based CF do not directly use rating information to generate recommendation, these methods first use rating information to train a model, which is then used to make recommendation.

3. Experimental Method

The E-commerce site used in this paper has been one of the most popular online bag retailers, and most of its customers are from North America. We collaborate with the site for over half a year to gather real data from its site.

3.1. Data Set

The data was gathered during January 2011 to June 2011. This data set contains more than 363000 times visits from over 63000 users.We get bag features from the site's database directly, these features to describe bags are color, size, price, discount, type, brand, etc. The feature database contains about 1300 bags. About 70 percent of the users are new user, and they come to visit our site for the first time and on average every user visits only about 5 product pages.

3.2. Evaluated Algorithm

We explore four categories of recommendation methods. The first category is content-based method [2]. And the second category is collaborative filtering method, this category contains two methods. One is item-to-item collaborative filtering method [5], and the other is traditional user-based collaborative filtering method [4]. The third category is method based on simple statistical result, here we use the simplest most popular visited N products. The last category is about some normal business sense, we select the cheapest and newest products to recommend to users.

Content-based: Content-based recommendation system tries to find the products that are similar to the items which user liked in the past. We recommend the products most similar to current visited product. Similarity between product a and b is calculated as follow:

$$similarity_{a,b} = \frac{|F_a \cap F_b|}{|F_a \cup F_b|} \tag{1}$$

where F_a is the feature set of product *a*, and F_b is the feature set of product *b*. $|F_a \cap F_b|$ denotes number of features product *a* and *b* both have, and $|F_a \cup F_b|$ denotes the entire number of features product *a* and *b* have.

Item-to-item-CF: Item-to-item collaborative filtering method defines similarity between items by the tendency of users often visit or purchase these items together. We apply the method similar to the algorithm proposed in [5]. Similarity between product a and b is calculated as follow:

$$similarity_{a,b} = \frac{CV(a,b)}{V(a) + V(b)} \tag{2}$$

where CV(*a*,*b*) denotes times of products *a* and *b* are co-visited, V(*a*) is the times of product *a* has been visited and V(*b*) is the times of product *b* has been visited.

User-based-CF: User-based-CF method first figure out similar users of the target user. And then generate recommendation result to target user according to her similar users' historical preference. Generally, we recommend products visited most often by target user's similar users. Because we have no rating information in the site, we replace rating by times of visit to products in user vectors. The similarity between two user vectors can be measured by Pearson correlation.

Most-popular: Most-popular method may be the most widely used strategy to recommend products without personalization. We recommend most popular visited products to users. This method does not need complex computation and get a relatively good performance at a very low cost.

Cheapest: Cheapest method is simple, we choose the cheapest products to recommend. One important feature of E-commerce is online shops can reduce fees compared to bricks-and-mortar stores so that they can provide a lower price. The performance of cheapest method will tell us whether low price is one incentive for users to shop online.

Newest: Just as simple as Cheapest method, we recommend the newest products to users. New products will arrive on shelves at online shop earlier than that in bricks-and-mortar store, so this method will detect whether a significant percent of consumers shop online for this reason.

3.3. Evaluation Method

We use top-N precision to evaluate the performance of algorithms in recommendation task. In order to measure precision, we first separate the original gathered data set which is mentioned in section 3.1 into two parts. The training set contains 80 percent of data, and the testing set 20 percent. To satisfy the different recommendation algorithms mentioned above, we apply two methods to separate the data set. For content-based, item-to-item-CF and user-based-CF, we cannot do any recommendation when the user comes to visit our site for the first time. So when we randomly select 20 percent of the data from the original gathered data set, we exclude those records of the products user visits when they come to our site at first. For the last three algorithms, the recommendation is based on the overall data and not specific to each user. So we select the 20 percent testing data completely randomly.

The computation of precision proceeds as follows. For any single test, if the test product is one of the recommended *N* products it is a hit case. Then the overall precision can be defined by follow formula:

$$precision(N) = \frac{hit - times}{N * |U_T|} \tag{3}$$

where *hit-times* is in all test cases the times of recommended *N* products contain test product, *T* is the test set and $|U_T|$ is the number of users *T* contains.

Except for precision, we also evaluate the cost of each recommendation method according to following five criterions:1. require item feature;2. require rating information;3. require user behavior history data;4. require statistical information of the data set;5. require relatively complex calculation; Each criterion adds 1 point to the total cost.

4. Result and Discussion

4.1. Experimental Result

In this section, we present the experimental results of the recommender algorithms mentioned in above section on our data set.We apply the evaluation method described in section 3.3 to evaluate six recommender algorithms.

Figure 1 shows that item-to-item-CF gets the best performance with precision of 19.50%, it outperforms other methods quite a lot, and its 3 points cost is relatively high but not the highest. The most-popular as benchmark method gets precision of 8.86% which is the second best performance, and its 2 points cost is relatively low compared with other methods with similar performance. It shows that most-popular has really high cost performance. Content-based with a 6.85% precision and user-based-CF getting a 7.39% precision do not outperform most-popular which we select as the benchmark. Also, these two methods have the highest 4 points cost. This does not indicate that these two main recommendation methods are not effective, but maybe they have some limitations under vertical B2C E-commerce cased. Cheapest and newest have not brought us any surprise. Although their cost is really low with 1 point, but the performances of cheapest with a 6.12% precision and newest with a 3.23% precision are not acceptable.

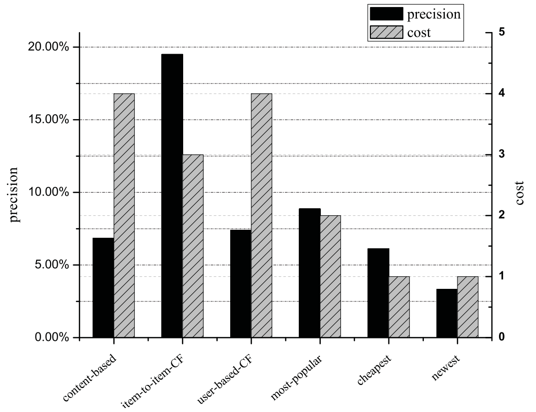


Figure 1. Top-5 precision and cost of methods

4.2. Discussion

Content-based: Content-based do not outperform the benchmark in our experiment, and we think two main reasons are the low quality of product feature information and lack of item ratings to detect users' preference. The low quality feature information of bags mainly comes from two reasons, one is some suppliers do not give all feature information and the other is different suppliers use different words as features. To preprocess the feature information from different suppliers, we need to fill in some missing features and unify different words which express the same meaning. In our experiments, we assume all visited items are of the same interest to user. But if we know the degree of users' interest in each item, we can weight most preferred items' features more.

Item-to-item-CF: Item-to-item-CF gets the best performance, this is an algorithm successfully applied in Amazon's recommender system. In this algorithm, the main cost is that we need to count all item pairs' co-visited or co-purchased times. Although the average number of items visited by one single user is only about 5, we have sufficient visiting history of different users to one single item, and it is exactly what this algorithm needs to figure out similar items from the item perspective. Also this method do not need item feature information, it is one advantage to

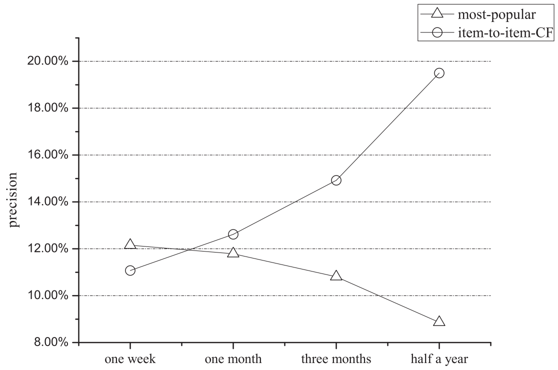


Figure 2. Top-5 precision of different time period

Content-based method.

User-based-CF: user-based-CF performs much worse than item-to-item-CF mainly because the key point of this algorithm is accurately computation of the user similarity. As mentioned in Data Set section, on average every user visits only 5 products in the site, so it is severely lack of co-visited products of users to compute user similarity. Another factor of user-based-CF's bad performance is the same as content-based method, we have no ratings to know users' preference to items. To get rating information needs users' extra effort. Because the number of user is much large than that of item, computing user similarity costs more than computing item similarity, that is why User-based-CF costs more than Item-to-item-CF.

Most-popular: This simple benchmark does a good job in our experiment because of its high cost performance. This method needs not much information such as item feature information, user rating and calculation cost is also low. It only needs the statical information of users' visiting or purchasing information.

Cheapest&Newest: These two methods derived from common business sense do not surprise us, but we still recommend to have a try on these simplest methods because sometimes simple does not mean ineffective. To make the recommendation process complex and obscure is not the objective of recommendation.

What we also want to discuss is the time value of the data. We choose item-to-item-CF and most popular which get relatively better performance to test how time factor affect results. Figure 2 shows the result of these two methods in different time period.

We can figure out data from different time period will affect result a lot, and also the effect to different methods is different. Most-popular perform better in a short period such as one week and gradually deteriorate as statistical period goes long, this may indicate that users' preference varies fast, the most popular products this week will be different from those of next week. Item-to-item-CF performs better when time goes by because it really needs some time to collect users' visiting or purchasing data for computing item similarity.

5. Acknowledgment

The work described in this article was partially supported by the Humanities and Social Sciences Foundation of Ministry of Education of China (10YJC870020, 10YJC630283), the National Natural Science Foundation of China(11171148, 61003024). The authors would like to thank the industrial partner for sharing data.

References

[1] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. Knowl. Data Eng.*, 17(6):734–749, 2005.

[2] M. Balabanovic and Y. Shoham. Fab: Content-based, collaborative recommendation. *Commun. ACM*, 40(3):66–72, November 1997.

[3] W. Hill, L. Stead, M. Rosenstein, and G. Furnas. Recommending and evaluating choices in a virtual community of use. CHI '95, pages 194–201, New York, NY, USA, 1995.

[4] J. A. Konstan, B. N. Miller, D. Maltz, J. L. Herlocker, and L. R. G. J. Riedl. Grouplens: Applying collaborative filtering to usenet news. *Commun. ACM*, 40(3):77–87, 1997.

[5] G. Linden, B. Smith, and J. York. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE INTERNET COMPUTING*, 7(1):76–80, 2003.

[6] E. Ngai, L. Xiu, and D. Chau. Application of data mining techniques in customer relationship management: A literature review and classification. *Expert Syst. Appl.*, 36(2):2592–2602, 2009.

[7] X. Su and T. M. Khoshgoftaar. A survey of collaborative filtering techniques. *Adv. in Artif. Intell.*, January 2009.

[8] A. Toscher and M. Jahrer. The bigchaos solution to the netflix grand prize. 2001.

Automated Approaches to Support Secondary Study Processes: a Systematic Review

Jefferson Seide Molléri
UNIVALI – Universidade do Vale do Itajaí
Itajaí, Brazil
jefferson.molleri@univali.br

Fabiane Barreto Vavassori Benitti
UNIVALI – Universidade do Vale do Itajaí
Itajaí, Brazil
fabiane.benitti@univali.br

Abstract

Context: *There is the need to identify automated methodologies for supporting the systematic review process.*
Objectives: *To conduct a systematic review of the literature searching for automated approaches used by researchers to support the systematic review process.*
Method: *We undertook a systematic review following the guidelines set out in Kitchenham and Charters', analyzed the relevant studies, and compared our findings with previous studies.*
Results: *508 papers have been identified and reviewed according to inclusion and exclusion criteria, resulting in 31 relevant studies.*
Conclusions: *A wide variation in the approaches were identified, in the concentrate context of selection on of primary studies, data extraction and monitoring and data synthesis from the conducting the review phase.*

I. Introduction

The Systematic Literature Review (SLR) is defined as a specific methodology of research, developed in order to gather and evaluate the available evidence pertaining to a focused topic [1]. Systematic reviews are gaining popularity in software engineering, with reviews published on diverse topics, such as software engineering experiments. SLRs are a key tool for enabling evidence-based practice as they combine the findings from multiple studies. Such reviews are important, as the volume of research that needs to be considered by software engineering (SE) researchers is constantly expanding. [2]

Despite its importance, the systematic literature process is not a easy task, as it uses specific concepts usually unknown to researchers familiar with traditional (unsystematic) literature reviews. Even when they are conducted according to their corresponding 'good practice' rules, they suffer from lack of scientific rigor in performing its different steps. The development of a systematic approach of research review aims to establish a more formal and controlled process of conducting this type of investigation, avoiding the introduction of the biases of the unsystematic review. [1]

Moreover, the systematic reviews require considerably more effort than traditional literature reviews, since provide additional information on variations in the primary studies in a wide variety of empirical methods [3]. The accomplishment of experimental studies in Software Engineering is time consuming, hard task, and produces great volume of information and knowledge with complex management [4]. Hence, some studies strongly depend on a computerized infrastructure to support its processes [5].

In the face of difficulties encountered during the execution of systematic reviews, there is the need to invest efforts in research methodologies for planning and carrying out systematic reviews [1]. The support of a automated tool is essential to provide higher quality and facilitate the systematic literature review process [6]. This short report outlines the results of a systematic review aiming to identify the current state of the art of automated approaches to support the systematic review process.

II. Methods

This research follows the guidelines set out in Kitchenham and Charters' [3] guide for SLRs in software engineering. This research method provides a verifiable method of summarizing existing studies as well as identifying gaps in the current research.

A. Research questions

The review process begins with the construction of a protocol that contains the general scope of the study. The scope can contain a Population, Intervention, Comparison, Outcomes and Context criteria to frame research questions according to the PICOC method, referred per Kitchenham and Charters' guide [3].

- **Population:** researchers
- **Intervention:** automated approaches to support
- **Comparison:** phases and stages of the systematic review process
- **Outcomes:** productivity and reliability
- **Context:** secondary studies

Every systematic review has at least one primary research question with the possibility of more secondary questions. The following are the primary (RQ1) and secondary research questions (RQ2, RQ3 and RQ4) for this study:

- RQ1: Which automated approaches to support the secondary studies processes are used by researchers?
- RQ2: Which the phases and stages are supported by the identified approaches?
- RQ3: What limitations, phases and stages are not covered by the identified approaches?

- RQ4: What are the impacts of using an automated approach into productivity and reliability criteria the secondary studies process?

B. Data sources

For this SLR, the electronic databases in Table I were searched as these are the primary sources for software engineering research publications.

TABLE I. DATA SOURCES

Database	*URL*
ACM Digital Library	http://portal.acm.org
CiteSeer	http://citeseerx.ist.psu.edu
IEEE Explore	http://ieeexplore.ieee.org
IET – The Institution of Engineering and Technology	http://www.theiet.org/
ScienceDirect	http://www.sciencedirect.com
SpringerLink	http://www.springerlink.com

C. Search string

The PICOC method gives us a list of search terms to use in the electronic databases. The search string used the logical operator OR to include synonyms for each search term, and the logical operator AND to link together each set of synonyms. When concatenated using the appropriate boolean expressions the following generic search string was produced:

('secondary study' OR 'systematic review' OR 'literature review' OR 'mapping review') AND (software OR application) AND (planning OR conducting OR reporting OR 'research question' OR 'review protocol' OR 'study selection' OR 'quality assessment' OR 'data extraction' OR 'data synthesis' OR 'meta analysis' OR meta-analysis) AND (productivity OR reliability OR 'effort reduction')

The generic search string was adapted to match the individual requirements of each of the electronic database on our data sources list above.

D. Study selection

In order to determine whether or not a study should be included, the methods section or similar section of the primary study was evaluated in search for citations of the automated approaches to support the review. Studies that were selected for inclusion in this systematic review were identified from on-line electronic databases within the following criteria:

- Primary studies that present automated approaches to support the conduct of systematic reviews; and
- Secondary studies (systematic reviews and mappings, etc.) that refer the automated support tools used.

This review excluded studies based on the exclusion criteria, which have been built in order to exclude irrelevant publications while maintaining the studies of interest.

- Are related to non-automated approaches;
- Do not address the approaches used;
- Non-English studies;
- Duplicated studies; and
- Was unable to access the full text.

Although it was not possible to check every single study by a secondary reviewer, an inter-rater reliability test was performed in order to reduce the researcher's bias. The secondary reviewer selected 5 studies randomly from the list of relevant primary studies and performed the study selection process. The results between reviewers were compared and no differences were found.

E. Data extraction

The data extraction process was carried out by reading each of the 31 selected studies thoroughly and extracting relevant data, which were managed through an Microsoft Excel spreadsheet. In order to keep information consistent the data extraction for each selected study was driven by the following:

TABLE II. DATA EXTRACTION FOR EACH STUDY

Extracted Data	**Description**	**Research questions**
Bibliographic references	Author, year of publication, title and source of publication	RQ1
Study classification	Primary or secondary study	RQ1
Focus of the study	Main topic area, and objective of the study	RQ1
Application	Automated approach referenced	RQ1
Application context	Phase and stages of review process supported	RQ2, RQ3
Application results	Productivity and reliability data	RQ4

F. Data synthesis

For the data synthesis, we inspected the extracted data for similarities in order to define how results could be encapsulated. The results of the synthesis will be described in the subsequent sections. Extracted data were tabulated to demonstrate the basic information of each study. With respect to qualitative assessment of the studies, the following criteria were summarized:

- Bibliographic references and main topic area;
- Number of selected studies per database;
- Classification and studies by year of publication;
- Process phases and stages most widely covered; and
- Automated approaches most frequently cited.

III. RESULTS

After the literature search, 508 potentially relevant primary studies have been obtained. The larger sets of results came from ScienceDirect (305 papers) and SpringerLink (95). Studies were then reviewed according to the predefined inclusion and exclusion criteria, focused on the methods section that references the review methodology and the approaches to support the review process, reducing the number to 31 selected studies, as shown in Table III.

TABLE III. BIBLIOGRAPHIC REFERENCES FROM SELECTED STUDIES

Authors	Main topic area	Data source
Ali et al. (2010) [6]	Computer Science	IEEE Xplore
Bailey et al. (2007) [7]	Computer Science	IEEE Xplore
Dieste and Juristo, (2011) [8]	Computer Science	IEEE Xplore
Svensson et al. (2010) [9]	Computer Science	IEEE Xplore
Salleh et al. (2010) [10]	Computer Science	IEEE Xplore
Šmite et al. (2010) [11]	Computer Science	SpringerLink
Campbell et al. (2011) [12]	Medicine	SpringerLink
Ivarsson et al. (2011) [13]	Computer Science	SpringerLink
Gu and Lago (2009) [14]	Computer Science	SpringerLink
Roberts et al. (2008) [15]	Medicine	SpringerLink
Umoquit et al. (2011) [16]	Medicine	SpringerLink
Schwappach et al. (2007) [17]	Medicine	SpringerLink
Lane and Richardson (2010) [18]	Computer Science	ScienceDirect
Beecham et al. (2007) [19]	Computer Science	ScienceDirect
Dybå and Dingsøyr (2008) [20]	Computer Science	ScienceDirect
Breivold et al. (2011) [21]	Computer Science	ScienceDirect
Eadie et al. (2011) [22]	Medicine	ScienceDirect
Hannay et al. (2009) [23]	Computer Science	ScienceDirect
Bastani and Jaberzadeh (2011) [24]	Medicine	ScienceDirect
Mellado et al.(2010) [25]	Computer Science	ScienceDirect
Liu et al. (2010) [26]	Medicine	ScienceDirect
Kitchenham (2009) [27]	Computer Science	ScienceDirect
Klainin-Yobas et al. (2011) [28]	Medicine	ScienceDirect
Chen et al. (2009) [29]	Computer Science	ScienceDirect
Wojtusiak et al. (2009) [30]	Medicine	ScienceDirect
Zhang et al. (2010) [31]	Computer Science	ScienceDirect
Talaei-Khoei et al. (2011) [32]	Computer Science	ScienceDirect
Watt et al. (2009) [33]	Medicine	ScienceDirect
Jahangirian et al. (2010) [34]	Computer Science	ScienceDirect
Benchimol et al. (2010) [35]	Medicine	ScienceDirect
Belanger (1997) [36]	Envtl. Science	ScienceDirect

A. *Selected Studies*

Among the electronic databases, IEEE Xplore provide most relevant studies: about 23,8% of the 21 searched studies were selected, as Table IV. ScienceDirect and SpringerLink databases had a higher amount of selected studies, but with a lower rate of selected per searched papers. One single study was obtained in ACM Digital database, however this was a duplicated paper already found in IEEE Xplore. Neither CiteSeer or EIT had selected studies among searched ones.

TABLE IV. SELECTED STUDIES PER DATABASE

Database	Studies	
	Searched	***Selected***
IEEE Xplore	21	5 (23.8)*
SpringerLink	95	7 (7.36%)
ScienceDirect	305	19 (6.22%)
ACM Digital	39	1 (2.56%)*
CiteSeer	4	-
IET	2	-
	508	***31 (6.1%)***

* IEEE Xplore and ACM Digital had a duplicated study

B. *Studies Classification*

From the 31 papers selected there were 30 secondary studies and only one primary study, which suggest that the support of automated approaches in systematic review process is lacking in primary research. A good indication of the maturity of an area is the type of its publications. Journal articles are often more mature than conference papers [18]. Table V shows the number of each type of publication by year. Journal articles make up the majority (93,54%) of publications. These results likely to show that the more mature articles references best automated approaches used than studies in conferences and symposium papers.

TABLE V. TYPES OF PUBLICATION BY YEAR

Classification	Year of Publication						
	1997	***2007***	***2008***	***2009***	***2010***	***2011***	***2012***
Primary Studies							
Journal				1			
Secondary Studies							
Conference					1		
Journal	1	1	3	2	7	9	5
Symposium		1					
Total	***1***	***2***	***3***	***3***	***8***	***9***	***5***

C. *Application Context*

A systematic literature review involves several tasks and activities. Kitchenham summarises the stages in a systematic review into three main phases: Planning the Review, Conducting the Review and Reporting the Review [3]. These three phases represent an overview of the systematic review process that could benefit form support of the automated approaches to grant productivity and reliability of execution.

Throughout this SLR, all selected studies focused on the Conducting the Review stage, although they are distributed among some discrete activities covered by the five stages shown in Table VI. About 30,77% of the automated approaches address the selection of primary studies, 25,64% the data extraction and monitoring and 43,59% the data synthesis stage. These results show a lack of comprehensiveness of the approaches on the process stages and phases.

TABLE VI. APPLICATION CONTEXT

Context (Phase / Stage)	Automated approaches*
Conducting the Review	
Identification of research	0
Selection of primary studies	12 (30,77%)
Data extraction and monitoring	10 (25,64%)
Study quality assessment	0
Data synthesis	17 (43,59%)
Total	***39***

* In total 39 references to automated approaches were found in 31 studies selected

D. *Automated Approaches*

From the selected papers there was a wide variation in the approaches described, as well the context of their uses. Table VII shows the list of automated approaches referenced in selected studies. Microsoft Excel is the most common software identified in our study, i.e. 17,95%, but EndNote (15,38%), RevMan (12,82%) and Reference Manager (7,69%) were also well referenced. Two studies quote also a tool database that is not appointed in the article.

Some of the identified automated approaches covers particularly the selection of primary studies data stage: Refman, Zotero, SCOPUS and WordStat. Others concentrated in the extraction of data, such as Zotero, MS Access, and MIX. A number of other tools also support data synthesis: RevMan, Comprehensive Meta-Analysis, MySQL database, AQ21, Nvivo, PASW, PROCITE, SAS and STATA. Microsoft Excel was identified as a support tool to all the three stages listed, especially on data extraction and monitoring. EndNote was also referred as an approach capable to address both the selection of studies and data extraction stages of SLR process.

TABLE VII. AUTOMATED APPROACHES REFERENCED

Automated approach	**Selection of primary studies**	**Data extraction and monitoring**	**Data synthesis**	**Total**
Microsoft Excel [6], [11], [20], [21], [34], [35], [36]	1	4	2	7
EndNote [14], [19], [20], [21], [25], [36]	4	2		6
RevMan - Review Manager [12], [24], [26], [29], [33],			5	5
Reference Manager (Refman) [15], [16], [20]	3			3
Comprehensive Meta-Analysis [23], [28]			2	2
MySQL database [7], [13]			2	2
unnamed database tool [9], [17]	1	1		2
Zotero [18], [32]	1	1		2
AQ21 attributional rule learning program [30]			1	1
Microsoft Access [18]		1		1
MIX: Comprehensive Free Software for Meta-Analysis of Causal Research Data [10]		1		1
Nvivo [20]			1	1
PASW statistic [28]			1	1
PROCITE [8]			1	1
SAS [36]			1	1
SCOPUS (search facilities) [27]	1			1
STATA [17]			1	1
WordStat [31]	1			1
Total	12	10	17	**39**

* In total 39 references to automated approaches were found in 31 studies selected

E. *Productivity and Reliability*

In order to answer RQ4 we aimed to obtain the results concerning productivity and reliability in application of the identified automated approaches. Productivity criteria should be scored by the reduction of effort in the SLR, and reliability would be given by the completion of discrete activities in accordance with process guidelines. However, no article had demonstrated productivity and reliability data that could be collected.

IV. DISCUSSION

This systematic review's overall goal, addressed by RQ1, is to identify existing automated approaches to support the systematic review process. Among the approaches identified, Microsoft Excel was most often cited, at three different stages of the process. This fact suggests that spreadsheets are the prime choice of researchers to support the process of conducting a systematic review. Other reference management tools were also well cited, such as EndNote, Reference Manager and RevMan.

In order to answer RQ2 and RQ3, the application contexts of the identified approaches were extracted and summarized, showing that most approaches focus on Conducting the Review phase. This implies a lack of adequate automated procedures to researchers in Planning and Reporting phases, or ignorance of researchers about these approaches. These phases covers important activities of the SLR process, such as defining the research questions, producing a review protocol and writing up the results.

Even the entire Conducting the Review stage was not supported by the tools identified in this study. It became clear the lack of automated approaches to support the Identification of research and Study quality assessment stages. It is important to recognize that many activities are initiated during a preliminary stage and refined when subsequent stages takes place, involving iteration methods [3] not showed by the identified approaches.

With respect to the impacts of using an automated approach into productivity and reliability criteria, the review found no clear evidences in the studies that could be summarized. A single primary study [30] discussed application methods of the approach in detail, but lacked an evaluation process to provide evidences on the reduction of effort and quality assurance of the SLR process.

A. *Threats to Validity*

As with SLRs in general there is often a lot of subjectivity involved in the selection of primary studies as well as the data extraction stages. In order to minimize the subjectivity involved in this study the reviewers strictly adhered to the instructions set out in the review protocol. The majority of the SLR activities were conducted by the primary reviewer. In order to reduce the single reviewer bias, an inter-rater reliability test was performed by the secondary reviewer. There were no difference between the results of the two reviewers.

Due to lack of the productivity and reliability data in results, the impacts of using an automated approach to support the systematic review process could not be observed. We notice that this issue may have two interpretations: the search string may be inaccurately constructed (although supported by PICOC method); or this topic of research has not yet reported results. We consider this issue a gap in the state of the art, and recommend this research question to be addressed by further studies.

V. CONCLUSION

In order to enhance the comprehension of the state of the art, this paper reports the results of a secondary study on searching relevant automated approaches to support the systematic review process. A wide variation in the approaches were identified, supporting particularly some discrete activities of the Conducting the Review phase. Since no relevant study provided adequate results concerning productivity and reliability of the approaches, this paper serves well as a basis for further research efforts in the search for evidences of improvements in SLR process through the support of automated approaches.

REFERENCES

[1] J. Biolchini, P. G. Mian, A. C. Natali, and G. H. Travassos, "Systematic Review in Software Engineering: Relevance and Utility," Rio de Janeiro, Brazil: Systems Engineering and Computer Science Department, UFRJ.

[2] T. Dybå, and T. Dingsøyr, "Strength of evidence in systematic reviews in software engineering," Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement, Kaiserslautern, Germany, pp. 178—187, October 2008.

[3] B. Kitchenham and S. Charters. "Guidelines for performing systematic literature reviews in software engineering (version 2.3)," Technical report, Keele University and University of Durham, 2007.

[4] F. Shull, J. Carver, and G. H. Travassos, "An empirical methodology for introducing software processes," SIGSOFT Softw. Eng. Notes}, vol. 26, no. 5, pp. 288—296, September 2001.

[5] G. H. Travassos, P. S. M. dos Santos, P. G. M. Neto, and J. Biolchini, "An Environment to Support Large Scale Experimentation in Software Engineering," Engineering of Complex Computer Systems, 2008. ICECCS 2008. 13th IEEE International Conference on, vol., no., pp.193—202, Mar.-Apr. 2008.

[6] A. Zamboni, A. Thommazo, E. C. M. Hernandes, and S. C. P. F. Fabbri, "StArt – Uma Ferramenta Computacional de Apoio à Revisão Sistemática," in CBSOFT – Congresso Brasileiro de Software: Teoria e Prática, 2010, Salvador, Anais do Congresso Brasileiro de Software, 2010, vol. 1, no., pp. 1—6, Setembro 2010.

[7] S. Ali, L. C. Briand, H. Hemmati, and R. K. Panesar-Walawege. "A Systematic Review of the Application and Empirical Investigation of Search-Based Test Case Generation," Software Engineering, IEEE Transactions on , vol. 36, no. 6, pp. 742-762, Nov.-Dec. 2010.

[8] J. Bailey, D. Budgen, M. Turner, B. Kitchenham, P. Brereton, S. Linkman. "Evidence relating to Object-Oriented software design: A survey," Empirical Software Engineering and Measurement, 2007. ESEM 2007. First International Symposium on , vol., no., pp. 482-484, 20-21 Sept. 2007.

[9] O. Dieste and N. Juristo. "Systematic review and aggregation of empirical studies on elicitation techniques," Software Engineering, IEEE Transactions on , vol. 37, no. 2, pp. 283-304, March-April 2011.

[10] R. B. Svensson, M. Höst and B. Regnell. "Managing Quality Requirements: A Systematic Review," Software Engineering and Advanced Applications (SEAA), 2010 36th EUROMICRO Conference on , vol., no., pp. 261-268, 1-3 Sept. 2010.

[11] N. Salleh, E. Mendes and J. Grundy. "Empirical Studies of Pair Programming for CS/SE Teaching in Higher Education: A Systematic Literature Review," Software Engineering, IEEE Transactions on , vol. 37, no. 4, pp. 509-525, July-Aug. 2011.

[12] D. Šmite, C. Wohlin, T. Gorschek, and R. Feldt, "Empirical evidence in global software engineering: A systematic review", Empirical Software Engineering, vol. 15, no. 1, pp. 91-118, February 2010.

[13] F. Campbell, M. Johnson, J. Messina, L. Guillaume and E. Goyder, "Behavioural interventions for weight management in pregnancy: a systematic review of quantitative and qualitative data," BMC Public Health, vol 11, no. 1, pp., December 2011.

[14] M. Ivarsson and T. Gorschek, "A method for evaluating rigor and industrial relevance of technology evaluations," Empirical Software Engineering, vol. 16, no. 3, pp. 365-395, June 2011.

[15] Q. Gu and P. Lago, "Exploring service-oriented system engineering challenges: A systematic literature review," Service Oriented Computing and Application, vol. 3, no. 3, pp. 171-188, September 2009.

[16] J. Roberts, A. Huissoon, J. Dretzke, D. Wang, and C. Hyde, "A systematic review of the clinical effectiveness of acupuncture for allergic rhinitis," BMC Complement Altern Med, vol. 8, no. 1, pp., December 2008.

[17] M. J .Umoquit, P. Tso, H. E. D. Burchett, and M. J. Dobrow, "A multidisciplinary systematic review of the use of diagrams as a means of collecting data from research subjects: application, benefits and recommendations," BMC Medical Research Methodology, vol. 11, no. 1, pp., December 2011.

[18] D. Schwappach, and T. Boluarte, "HEE-GER: a systematic review of German economic evaluations of health care published 1990–2004", BMC Health Services Research, vol. 7, no. 1, December 2007.

[19] S. Lane, and I. Richardson, "Process models for service-based applications: A systematic literature review, Information and Software Technology, vol. 53, no. 5, pp. 424–439, May 2011.

[20] S. Beecham, N. Baddoo, T. Hall, H. Robinson, and H. Sharp, "Motivation in Software Engineering: A systematic literature review," Information and Software Technology, vol. 50, no. 9–10, pp. 860–878, August 2008.

[21] T. Dybå, and T. Dingsøyr, "Empirical studies of agile software development: A systematic review," Information and Software Technology, vol. 50, no. 9–10, pp. 833–859, August 2008.

[22] H. P. Breivold, I. Crnkovic, and M. Larsson, "A systematic review of software architecture evolution research," Information and Software Technology, vol. 54, no. 1, pp. 16–40, January 2012.

[23] L. H. Eadie, P. Taylor, and A. P. Gibson, "A systematic review of computer-assisted diagnosis in diagnostic cancer imaging," European Journal of Radiology, vol. 81, no. 1, pp. e70–e76, January 2012.

[24] J. E. Hannay, T. Dybå, E. Arisholm, and D. I. K. Sjøberg, "The effectiveness of pair programming: A meta-analysis," Information and Software Technology, vol. 51, no. 7, pp. 1110–1122, July 2009.

[25] A. Bastani, and S. Jaberzadeh, "Does anodal transcranial direct current stimulation enhance excitability of the motor cortex and motor function in healthy individuals and subjects with stroke: A systematic review and meta-analysis," Clinical Neurophysiology, vol. 123, no. 4, pp. 644–657, April 2012.

[26] D. Mellado, C. Blanco, L. E. Sánchez, and E. Fernández-Medina, "A systematic review of security requirements engineering," Computer Standards & Interfaces, vol. 32, no. 4, pp. 153–165, June 2010.

[27] F. Liu, M. Qiu, and S. Zhai, "Tolerability and effectiveness of (S)-amlodipine compared with racemic amlodipine in hypertension: A systematic review and meta-analysis," Current Therapeutic Research, vol. 71, no. 1, pp. 1–29, February 2010.

[28] B. Kitchenham, "What's up with software metrics? – A preliminary mapping study," Journal of Systems and Software, vol. 83, no. 1, pp. 37–51, January 2010.

[29] P. Klainin-Yobas, M. A. A. Cho, and D. Creedy, "Efficacy of mindfulness-based interventions on depressive symptoms among people with mental disorders: A meta-analysis," International Journal of Nursing Studies, vol. 49, no. 1, pp. 109–121, January 2012.

[30] S. Chen, A. Flower, A. Ritchie, J. Liu, A. Molassiotis, H. Yu, and G. Lewith, "Oral Chinese herbal medicine (CHM) as an adjuvant treatment during chemotherapy for non-small cell lung cancer: A systematic review," Lung Cancer, vol. 68, no. 2, pp. 137–145, May 2010.

[31] J. Wojtusiak, R. S. Michalski, T. Simanivanh, and A. V. Baranova, "Towards application of rule learning to the meta-analysis of clinical data: An example of the metabolic syndrome, International Journal of Medical Informatics," vol. 78, no. 12, pp. e104–e111, December 2009.

[32] H. Zhang, M. A. Babar, and P. Tell, "Identifying relevant studies in software engineering," Information and Software Technology, vol. 53, no. 6, pp. 625–637, June 2011.

[33] A. Talaei-Khoei, P. Ray, N. Parameshwaran, and L. Lewis, "A framework for awareness maintenance," Journal of Network and Computer Applications, vol. 35, no. 1, pp. 199–210, January 2012.

[34] A. M. Watt, M. Patkin, M. J. Sinnott, R. J. Black, and G. J. Maddern, "Scalpel safety in the operative setting: A systematic review," Surgery, vol. 147, no. 1, pp. 98–106, January 2010.

[35] M. Jahangirian, T. Eldabi, L. Garg, G. T. Jun, A. Naseer, B. Patel, L. Stergioulas, and T. Young, "A rapid review method for extremely large corpora of literature: Applications to the domains of modelling, simulation, and management," International Journal of Information Management, vol. 31, no. 3, pp. 234–243, June 2011.

[36] E. I. Benchimol, D. G. Manuel, T. To, A. M. Griffiths, L. Rabeneck, and A. Guttmann, "Development and use of reporting guidelines for assessing the quality of validation studies of health administrative data," Journal of Clinical Epidemiology, vol. 64, no. 8, pp. 821–829, August 2011.

[37] S. E. Belanger, "Literature Review and Analysis of Biological Complexity in Model Stream Ecosystems: Influence of Size and Experimental Design," Ecotoxicology and Environmental Safety, vol. 36, no. 1, pp. 1–16, February 1997.

Enforcing Contracts for Aspect-oriented programs with Annotations, Pointcuts and Advice

Henrique Rebêlo Ricardo Lima Alexandre Mota César Oliveira Márcio Ribeiro

Federal University of Pernambuco, PE, Brazil
{hemr, rmf , acm, calo, mmr3}@cin.ufpe.br

Abstract

Over the last years, several proposals have advocated that a notion of interface between the base code and aspect code is necessary for reasoning about aspect-oriented programming (AOP), and for overcoming pointcut fragility. However, existing work that are AOP based, have not shown how one can specify these interfaces to facilitate modular reasoning and specify control effects, such when advice does not proceed. The main contribution of this work is a new form of interface for AOP that we call crosscut programming interface with design rules, or XPIDR. XPIDRs extend the notion of crosscut programming interfaces (XPIs) with expressive design rules that allow modular understanding and enforcement of control flow effects. We also show that since our approach with XPIDRs do not require any new AOP construct, they can be adopted in a straightforward manner by the AOP community.

1 Introduction

Aspect-oriented programming (AOP) [5] is a well-known technique that explicitly supports the modularization of crosscutting concerns. With the emergence of AOP, in the literature, we have cookbooks that guide the "aspectization" of specific crosscutting concerns, such as contract enforcement [12, 13], and so forth. Thus, through modularization of such crosscutting concerns, one might increase code modularity and reusability.

However, there is some consensus in the aspect-oriented community that AOP constructs aimed to support crosscutting modularity can, in fact, compromise class modularity. A typical approach employs a notion of interface between classes and aspects. These interfaces, also refereed as *design rules*, have been discussed and well researched [4, 9, 1]. The research question behind this work is how effective plain AOP techniques, like those present in languages such as AspectJ, can be used to enforce contracts for classes and aspects.

1.1 The Problems

The recognition that the Design by Contract (DbC) [8] technique can be implemented and better modularized using AOP is not new. There are several related works that implement such dynamic contract checking using aspects [14, 11, 12]. In addition, DbC as aspects have been also explored to AOP itself [4, 14]. However, the behavioral contracts, such as pre- and postconditions, used to specify advice methods in AOP are used as black box. Such black boxes hide the internal control flow states from the base code and other aspects.

Figure 1 illustrates the classical figure editor [6] that uses contracts expressed by crosscutting interfaces (XPIs) [4]. XPIs provide a design rule between the base code (classes) and aspects. For example, the XPI Changed specifies the pointcut jp (lines 10−11), which is exposed (implemented) by class Point (lines 2−8) and advised by aspect Update (lines 9−15). The behavioral contracts of XPI are used together with the behavioral interface specification language JML [7]. Thus, the behavioral contracts are written with requires clauses (preconditions) and ensures clauses (postconditions). Hence the XPI Changed states that any advice, which uses the pointcut jp (lines 10−11), must guarantee that the exposed target object fe is non-null either before and after advice's execution. Moreover, we use a ensures clause in the method setX of class Point (base code) stating that after method's execution the **this** .x must be equal to the value passed to the parameter x.

So, to illustrate the problems of the black box approach, we draw on the same problems discussed in the literature [9, 1]. Consider the classical figure editor example depicted in Figure 1 [6], which uses XPIs [4].

```
1 class Fig {}
2 class Point extends Fig{
3  int x; int y;
4  //@ ensures this.x == x;
5  void setX(int x){
6   this.x = x;
7  }
8 }
```

```
 9 aspect Changed{
10  pointcut jp(Fig fe):
11   call(void Fig+.set*(..))
12    && target(fe);
13  //@ requires fe!= null;
14  //@ ensures fe!= null;
15 }
```

```
16 aspect Update{
17  void around (Fig fe):
18   Changed.jp(fe){
19    proceed(fe);
20    Display.update();
21  }
22 }
```

Figure 1. A behavioral contract for aspect interfaces using Crosscutting Interfaces (XPI) [4].

```
1 dr Changed {
2  class Fig {}
3  class Point extends Fig {
4    void setX(int x){
5      set(int Point.x);
6    }
7  }
```

```
 8 aspect Update {
 9  pointcut jp:
10    call(void Fig+.set*(..))
11   && target (fe);
12  void around(Fig fe): jp(fe){
13   proceed(fe);
14   xcall(void Display.update());
15  }
16 }
```

Figure 2. Design rules for aspect interfaces in LSD [9].

The first problem [1] with the use of behavioral contracts to AOP is that they are insufficient to guarantee internal control flow effects. For instance, if the call to **proceed** method (line 19) is for any reason missed, the contract in lines 13−14 is not enough to alert the programmer about this mistake. By missing the **proceed** method, the original call to the join point setX is skipped. Hence, the expression **this**.x = x is not evaluated. We have the same problem even if we provide behavioral contracts for the base code stating that after method setX's execution, we must have the field **this**.x assigned to x (as depicted in Figure 1 with the use of ensures clause in line 4). Since the enforcement of these contracts are attached to the execution of the method setX, they are also skipped when **proceed** is missing.

The second problem [1] by using behavioral contracts is that the reasoning about the control effects of an advice to other advice seems difficult. Consider we have another concern such as a Logging aspect, which logs any change to the field **this**.x by setX method calls. Assuming we have the call to **proceed** method (line 19 in Figure 1) missed (as discussed), we can have different results when composing the system with both Update and Logging concerns. Possible results comprehends the skipped execution of the Logging concern or even its normal execution. The former happens when the Update concern is executed first, whereas the latter happens when the Logging is considered firstly instead. This scenario can be even worst with the applicability of other concerns with their respective advising code.

The third problem with the conventional use of the behavioral contracts [9] is that they are not sufficient for specifying components behavior such as required method calls and field access or change. For example, by using such behavioral contracts, we cannot guarantee that the **around** advice of aspect Update has a call to Display.update() method. This mandatory call is needed to enforce the realization of the concern Update. In the same manner, we cannot ensure that the method setX of class Point really assign the value of parameter x to field **this**.x. Again, this is mandatory to ensure the realization of the Update concern.

1.2 Previous Work on These Problems

As discussed, design by contract (DbC) methodologies for AOP have been considered [4, 14] in the literature. However, none of these works can handle the mentioned problems because they rely on black box behavioral contracts. Black box approach deals with only the relationship between its inputs and outputs. In other words, all available information deals with what is before and after a method's execution [3]. No information is known about what happens-in-between. Therefore, the contracts for AOP expressed, for example, by XPIs [4] are not expressive enough to mitigate the problems we described. XPIs represents contracts for AOP using aspects constructs.

A previous work on these problems showed how to mitigate them by using the Language for Specifying Design Rules (LSD) [9]. This language is another concept of an interface used to decouple classes and aspects. It is used for improving modularity and maximizing independent development opportunities. LSD provides *behavioral rules* to specify internal control flow effects. Table 1 presents the six design rules adopted by LSD. As an example of a detailed design such as mandatory method calls or field state change, the line 5 of the Figure 2 presents a **set** behavioral rule which states that the method setX of class Point must have a field state change denoted by **this**.x = x. In the as-

Table 1. Behavioral Rules in LSD [9].

Rule	Description
call(method)	It must have a *method* call within the defined scope.
xcall(method)	It must have a *method* call exclusively within the defined scope.
get(field)	It must have a *field* state access within the defined scope.
xget(field)	It must have a *field* state access exclusively within the defined scope.
set(field)	It must have a *field* state change within the defined scope.
xset(field)	It must have a *field* state change exclusively within the defined scope.

pect code, we can observe the xcall behavioral rule in line 14. Such a rule states that the **around** advice of aspect Update must have a call to Display.update() method. Also, since the xcall rule starts with an 'x', the method Display.update() cannot be called in other place inside the scope of the design rule interface Changed.

Since a specifier using LSD can decide what to hide and reveal, in relation to specific internal control flow effects, we would say that LSD is also a grey box approach [3]. However, LSD does not support behavioral contracts such as preconditions. This can be mitigated by incorporating JML features to LSD design rule language. It is important to note that LSD is not a standard AspectJ program. It is a good proposal to be implemented in the current AspectJ language. Therefore, currently, the authors of LSD are working towards a release for AO developers [9].

Another work on these problems uses the notion of *translucid contracts* [1]. A translucid contract describes in a detailed way the behavior of aspects when applied to the AO interfaces. With translucid contracts, the specifier decide to hide some details, while revealing others. Translucid contracts are based on grey box specification approach [3]. Translucid contracts are similar to XPIs (Figure 1), but instead of AspectJ, translucid contract is implemented in Ptolemy with its quantified typed events [10].

Finally, the AO community has a plenty of techniques to reason about their programs [14, 4, 9, 1]. But the existing behavioral contracts for AO interfaces, which are AspectJ-based [4, 14], are insufficiently to advert developers about inconsistencies in AO program designs and implementations. Therefore, the AO community is lacking for an approach to define expressive contracts for AO programs using aspect constructs.

1.3 Contributions to the State-of-the-art

Then main contribution of this work is the use of standard AO constructs like those of AspectJ to enforce control flow effects. We extend the XPI approach [4] to add more expressive contract checking properties for AO programs. Likewise XPI [4], we employ aspects to enforce AO contracts combined with grey box specification based approach [3] taken from LSD [9] and translucid contracts [1]. To provide the grey box specification approach, we draw on the work of LSD. Hence, we demonstrate how to apply behavioral rules, such as xcall, using plain aspect constructs.

In this paper we focus on three mechanisms to enforce contracts for AOP: (i) metadata annotations, (ii) pointcuts and (iii) advice [6]. Our goal is understand how the combination of these approaches can provide a more expressive way to check contracts for aspects than those already researched [14, 4]. We discuss some benefits of the use of annotations as a part of our solution to promote contracts for AO programs.

2 XPIs with Design Rules

The key idea behind our design methodology is to introduce a design phase for each crosscutting concern. Hence, a designer establishes a crosscutting design rule interface to decouple the base and the aspect design. Such a crosscutting design rule is based on a previous work by Griswold *et al.* [4], the well known crosscut programming interfaces (XPIs). The main difference is that we extend XPIs with the notion of behavioral rules [9]. Hence with such behavioral rules, we enhance XPIs with a grey box specification capability [3]. We call our enhanced XPI as *crosscut programming interface with design rule*, or XPIDR.

We implement XPIDRs as syntactic constructs in AspectJ. A XPIDR is composed of the following elements: (i) a name; (ii) a set of one or more abstract join points; (iii) a scope in which the abstract join points are defined; (iv) during the system's concern specification, the specifier can expose or hide specific join points, and (v) all the exposed join points should also present their specifications using behavioral rules and contracts.

All of these elements are declared by using standard AspectJ constructs. Such elements enable one to provide an expressive interface to be respected by the base and crosscutting concern implementation of the system.

2.1 The Figure Editor with XPIDRs

Figure 3 illustrates the use of XPIDR, written in AspectJ, for the figure editor example (lines 1–16). By convention, aspects that specify XPIDRs begin with an "XDR" to distinguish them from non-interface aspects. Hence, the XPIDR XDRUpdate declared in Figure 3 is the same aspect illustrated previously in Figures 1 and 2 named Changed.

XPIDR comprehends simple pointcut declarations in which aspects can provide the advising code without needing to refer directly to the underlying source code (lines 2–3). In addition, we can also specify properties of the underlying pointcuts in a grey box style. Thus, the specifier

```
 1 public aspect XDRUpdate {
 2  public pointcut jp(Fig fe): target(fe)
 3   && (call(void Fig+.set*(..))) && @annotation(UpdateConcern);
 4  public pointcut bcscope(): within(Fig+);
 5  public pointcut acscope(): within(*Update*);
 6  public pointcut xpidrscope(): bcscope() || acscope();
 7  /* Grey box specification of the Update concern base code */
 8  declare @method: void Point.setX(int):   @UpdateConcern(Method.POINT_SETX);
 9  declare @method: void Point.setX(int):   @Set("int Point.x");
10  /* Grey box specification of the Update crosscutting concern code */
11  declare @method: void Display.update():  @UpdateConcern(Method.DISPLAY_UPDATE);
12  declare @method: void Update.aroundAdvBody(Fig):  @UpdateConcern(Method.UPDATE_AROUND);
13  declare @method: void Update.aroundAdvBody(Fig):  @Requires("fe != null");
14  declare @method: void Update.aroundAdvBody(Fig):  @XCall("public static void Display.update()");
15  declare @method: void Update.aroundAdvBody(Fig):  @Ensures("fe != null");
16 }
```

Figure 3. An example of the XPIDR for the update concern of the figure editor.

can choose whether or not reveal specific details of code to be advised. For example, lines 8−9 explicitly present details of the base code. So, developers referring this XPIDR know that the based code specifically contains a Point.setX method. Besides that, we can observe that such a method contains two attached annotation properties.

The former (line 8) is an annotation which can be declared anywhere. It refers to the update concern that the system realizes. Furthermore, it provides an enumeration property (Method.POINT_SETX) which specifically identifies which particular element of the Update concern it belongs to (the method Point.setX in this case). We use a enumeration property because we can mark several methods and advice of a particular concern.

The latter denotes the **set** behavioral rule [9]. Hence, we can exactly specify what fields must be set within the method Point.setX. In this case, the design rule on line 9 states that the method Point.setX must have an a assignment (field state change) to the field Point.x. We also represent behavioral rules as metadata annotations. Thus, the **set** design rule is denoted by **@Set**, which is attached to the Point.setX method. With XPIDR we can also employ five more design (behavioral) rules. Table 1 presents the six design rules we draw from LSD [9] and that we made available by XPIDR. Note that since AspectJ 5, we can attach (mark) methods using the AspectJ supplying annotation feature such as **declare @method** (lines 8−15).

The syntactic part of the XPIDR also exposes other two methods: (i) the method Display.update in line 11, and (ii) the method **void** Update.aroundAdvBody(Fig) in lines 12−15. Both are related to the crosscutting concern code. The second one (Update.aroundAdvBody) denotes a separate method to represent an advice since AspectJ does not support supplying annotations for advice yet. The method Update.aroundAdvBody is composed by the behavioral rule @XCall to denote that the advice must have a call to method Display.update. This mandatory call is also exclusive in the scope of the XPIDR for the Update concern (since the notation starts with an ‘x’).

Considering the behavioral contracts, pre- and postcondition are specifed in lines 13 and 15 (see Figure 3), respectively. These conditions state that the target object (denoted by fe) must be non-null before and after method Update.aroudAdvBody’s execution. We refer to fe which is the exposed target Fig object by the pointcut jp. We also represent behavioral contracts as metadata annotations. We adopted the work by Boysen [2] in our XPIDR, which already provide Java 5 annotations for JML [7]. These JML behavioral contracts as metadata annotations can be also attached to the base code.

Note that we use simple Java methods to refer to AspectJ advice since advice declarations in AspectJ are unnamed. We cannot explicitly refer them in our XPIDR to attach annotations with design rules. With such a limitation we rely on simple methods that are supposed to encapsulate the advice behaviors. For example, to implement the update concern, we should have an aspect named Update and according to our XPIDR it must declare the method Update.aroundAdvBody. Such a method is called from within an **around** advice declared in the same aspect Update.

Figure 4 presents a Update aspect (lines 1−14) using our XPIDR provided in Figure 3. The aspect now depends only on the abstract public pointcut signatures of XDRUpdate, not on implementation details of the Fig and Point classes. As mentioned, we cannot supply annotations for AspectJ advice. Hence, we also specify the advice declared in the Update aspect. In line 6 we mark the **around** advice with a custom metadata annotation (@AroundAdvice UpdateConcern). In addition, the **around** advice (lines 10−13) is specifed with two behavioral rules. The former

```
 1 public privileged aspect Update {
 2  void aroundAdvBody(Fig fe) {
 3       Display.update();
 4  }
 5
 6  @AroundAdviceUpdateConcern
 7  @XCall("void aspects.update.Update.
 8        aroundAdvBody(classes.Fig)")
 9  @Call("proceed(fe)")
10  void around(Fig fe): XDRUpdate.jp(fe){
11   proceed(fe);
12   aroundAdvBody(fe);
13  }
14 }
```

Figure 4. The Update aspect implementation.

stating that such an advice must have a exclusive call (line 7) to the method Update.aroundAdvBody. The latter stating that it must call the **proceed** method (line 9).

3 Contract Enforcement Implementation

In this section we describe how to implement the contract enforcement of the XPIDRs previously discussed. Due to the lack of space, secondary implementation details are discussed in our technical report available at `http://www.cin.ufpe.br/~hemr/seke12`. The main points of our strategy are covered in this section.

3.1 Checking Structural rules

Structural rules [9] are design rules that describe constraints about classes and aspects members. Its realization is similar to Java interfaces, but additional constraints (beyond required public methods) such as required fields or expected AspectJ inter-type declarations. By using our XPIDRs, we can force a developer to implement the required types, methods and fields we declare. For example, in Figure 3 in line 8, we expose the method Point.setX. Until the class Point is declared we got a warning message saying that "no match for this type name". Once the required type is declared, we got a compile-time error saying that the method "void Point.setX(**int**) does not exist". In this case the developer is forced to implement the exposed member by the XPIDR. This is an interesting feature described by LSD design rule language [9] that we can support using the AspectJ **declare** @<member> syntax (also known as supplying annotation syntax). In summary, as with LSD [9], XPIDRs enforce structural rules at compile-time.

3.2 Checking Behavioral Rules and Contracts

As discussed, a XPIDR interface adopts mechanisms such as behavioral rules and contracts to provide a detailed design specification in a grey box style [3] for program modules like classes and aspects. Before checking such design rules and contracts of the Update concern, we gather information (using a trace-based mechanism) about all the control flow join points related to the Update concern.

```
 1 /* Collection internal control flow join points
 2  * of the @UpdateConcern */
 3 before(UpdateConcern upc): upcWC(upc)
 4      && XDRUpdate.staticxpidrscope(){
 5  Util.addTrace(thisJoinPoint);
 6 }
 7 /* Collection internal control flow join points
 8  * of the @AroundAdviceUpdateConcern */
 9 before(): aroundAdvWC() {
10  Util.addTrace(thisJoinPoint);
11 }
```

Figure 5. Aspect code for tracing control flow join points.

Figure 5 presents the two **before** advice (lines 1−11) used to trace all control flow join points of the update concern. The first one (lines 3−6) traces all the join points that occurs within any join point marked with the Update concern annotation. The second **before** advice (lines 9−11) traces all the join points within the **around** advice, which is marked with a specific annotation, of the aspect Update. As observed, we store each monitored join point (lines 5 and 10) by using the method Util.addTrace. Such a method is available as a part of the XPIDR API.

4 Discussion

We showed that AOP can be used to provide expressive contract checking of aspect-oriented code. The main motivation of our approach is that it does not requires any new AOP-like construct to provide contract checking of AOP code. Instead of new constructs, we reuse and combine existing features like metadata annotations, pointcuts and advice.

With the grey box specification approach, the specifier can decide the degree in which the details are revealed. Hence, our XPIDR illustrated in Figure 3 exposes that we must have a class Point and a method setX. Also, the **@Set** design rule states that the method Point.setX must have a field state change to field Point.x. Suppose now that the specifier decided to keep details more hidden and still obeying the **@Set** design rule:

```
declare @method: * Fig+.set*(..): @Set("* Fig+.*");
```

This syntax abstracts the previous one by making details more hidden. However, the @Set design rule still guarantee properties of these **set** methods. Now, we just known that any **set** method declared in any type Fig must have a field state change to any field declared in type Fig.

The use of metadata annotations brings two additional benefits for the XPIDR interface specification and verification. The first one guarantees that the specific elements exposed by the XPIDR (using the AspectJ supplying annotation syntax) must be declared by the base and aspect code. The second one is that unlike XPIs, we can now check specific properties of a particular AspectJ advice. We match a specific advice based on its annotation type.

In relation to the use of design rules, we can minimize the impact of the well known problem of pointcut fragility. For example, if a developer changes the base code, by removing a join point shadow (a method call), can lead the aspect code to behave in a different way from what is expected. Hence, our XPIDR checks the method call occurrence and if it is not found, the developer is warned about the missing join point.

4.1 Feature Request

Some problems we had during XPIDR specification could be avoided if we could add a supply clause for advice. Therefore, we submitted a feature request to the AspectJ team, which they hope to consider for a future version of AspectJ. We suggested the support of a new construct that marks an advice with an annotation. For example, it could be used as in the following:

```
declare @advice: update.around(..): @UpdateConcern;
```

This declaration would avoid the extra annotations added directly in the advice declaration, such as the one used to say that an around advice must proceed.

5 Summary

Metadata annotations, pointcuts and advice are useful techniques commonly used for separating concerns in source code. We have combined these three AO techniques towards a design by contract methodology for AOP. Each of these AO mechanisms makes a different kind of task: annotations mark join points related to the base and aspect code; pointcuts bindings sets of join points marked with annotations, and advice provide the checking code implementation to sets of join points denoted by pointcuts.

We have combined the notion of crosscut programming interfaces (XPIs) [4] with more expressive design rules adopted by LSD [9]. Such design rules are specified and checked using the three AO mechanisms we focus in this paper. We call our enhanced XPIs as *crosscut programming interfaces with design rules*, or XPIDRs.

Our main contribution is that we have devised a practical alternative way to a enable an expressive design by contract methodology for AOP using existing AOP constructs like those of AspectJ. In addition, our approach, called XPIDRs, is a grey box specification based approach, which provides means to detailed specify control flow effects for both base and aspect code. In a grey box specification, the designer decides whether or not to expose control flow effects. As a result, only interesting control flow effects are exposed.

Finally, and most importantly, using XPIDRs a developer does not require new AO constructs, which in turns makes the adoption by the AO community more easier. We argued that the design rules of XPIDRs showed to be more expressive than the existing AO based techniques [4, 14].

References

[1] M. Bagherzadeh et al. Translucid contracts: expressive specification and modular verification for aspect-oriented interfaces. In *Proceedings of AOSD'11*, 2011.

[2] K. P. Boysen. A specification language design for jml using Java 5 annotations. Technical report, Iowa State University, 2008.

[3] M. Büchi and W. Weck. The greybox approach: When blackbox specification hide too much. Technical report, Aug. 06 1999.

[4] W. G. Griswold et al. Modular software design with crosscutting interfaces. *IEEE Softw.*, 23:51–60, January 2006.

[5] G. Kiczales et al. Aspect-oriented programming. In *ECOOP 97*, 1997.

[6] G. Kiczales et al. Getting Started with AspectJ. *Commun. ACM*, 44(10):59–65, 2001.

[7] G. T. Leavens, A. L. Baker, and C. Ruby. Preliminary design of JML: A behavioral interface specification language for Java. *ACM SIGSOFT Software Engineering Notes*, 31(3):1–38, Mar. 2006.

[8] B. Meyer. Applying "design by contract". *Computer*, 25(10):40–51, 1992.

[9] A. C. Neto et al. A design rule language for aspect-oriented programming. In *SBLP '09*, 2009.

[10] H. Rajan and G. T. Leavens. Ptolemy: A language with quantified, typed events. In *ECOOP 2008*.

[11] H. Rebêlo et al. Implementing java modeling language contracts with aspectj. In *Proc. of the 2008 ACM SAC*, 2008.

[12] H. Rebêlo et al. The contract enforcement aspect pattern. In *Proceedings of the 8th SugarLoafPlop*, pages 99–114, 2010.

[13] H. Rebêlo et al. Assessing the impact of aspects on design by contract effort: A quantitative study. In *SEKE*, pages 450–455, 2011.

[14] T. Skotiniotis and D. H. Lorenz. Cona: aspects for contracts and contracts for aspects. *ACM SIGPLAN Notices*, 39(10):196–197, Oct. 2004.

A. Online Appendix

We invite researchers to replicate, analyze, and compare our XPIDR implementation of the figure editor against its counterpart in XPI. The implementations are available at: `http://www.cin.ufpe.br/~hemr/seke12`.

Towards More Generic Aspect-Oriented Programming: Rethinking the AOP Joinpoint Concept

Jonathan Cook
New Mexico State University
Las Cruces, NM 88003 USA

Amjad Nusayr
University of Houston – Victoria
Victoria, TX 77901 USA

E-mail: joncook@nmsu.edu

Abstract

In aspect oriented programming, the concepts of joinpoint and pointcut have always been central and have, as their names imply, been centered around the idea of a point *in a program's execution. Furthermore, in practical terms this has often been synonymous with points in the static representation of the program where invocation of advice can be inserted. We present here ideas for rethinking both of these, most significantly presenting the idea of redefining a joinpoint to be an* interval *of program execution. This redefinition cleans up some concepts in AOP and opens the door to new ideas and mechanisms for future AOP research to consider.*

1. Introduction

In aspect-oriented programming (AOP), the *joinpoint model* and the *advice mechanisms* are important concepts in order to understand the capabilities for a particular AOP framework. These two essentially capture the weaving capability of the AOP framework, determining where in a computation advice can be applied (joinpoints), how advice will be applied to the underlying program (advice mechanisms), and how abstract the specification of this weaving can be (in the pointcut expression language).

Existing notions of joinpoints and their respective joinpoint models have suffered from a restrictive foundational definition that has hampered the expansion of AOP ideas into the full range of capabilities that it could encompass. This restrictive foundational definition is:

> *A joinpoint is a well-defined point in the execution of a program where advice can be applied.*

This basic definition can be found in essentially equivalent wording in many references e.g., [11], although other researchers have also noted its insufficiency [3, 5, 12].

The problem with this definition is with the focus on the word *point* (which reifies in the term *pointcut*). We claim that this word has stymied the development of ideas that can expand where and how AOP ideas can be applied. In fact, joinpoints should not be thought of as points in the program execution but should be a much more abstract concept, which then impacts how they should be specified.

Another issue is that the joinpoint model has not traditionally been well-separated from the *advice execution model*. The joinpoint model is typically reified into the *pointcut expression language*, but the way a pointcut expression is specified usually intertwines it with specifying how the advice will be executed, with the traditional mechanisms being *before, after*, or *around* the joinpoint.

Just as AOP embraces the notion of separation of concerns, in turn the concept of the joinpoint model should be separated from the concept of the advice execution model. In this paper we present a rethinking of the notions of the joinpoint model and advice execution model, and show how this new thinking can expand and enhance the future application of AOP. Our own interests are focused on using AOP to support the broad needs of runtime monitoring instrumentation, and the ideas presented in this paper will help fulfill those needs.

2. A New Approach to Joinpoints

We believe that the foundational ideas and definitions of joinpoints early on in the development of AOP have hindered the potential application of AOP ideas into new realms. In viewing the body of AOP work we see that for the most part joinpoints have largely corresponded to *locations in the source code* where a weaving compiler could insert the advice code so that program and advice execution was very efficient. Even joinpoint designators such as object field access, which appear data related and not code related, are translated into advice weaving at code locations where the field is accessed. Dynamic weaving, where runtime conditions could influence the execution of advice, still often uses underlying mechanisms that involve identifying a set of code locations which might execute the advice and then either performing static weaving augmented with dynamic checks (residues), or applying dynamic weaving on these locations as needed.

While Java supports efficient object field access joinpoints because it is easy to identify the bytecode instructions that access them, in less memory-safe languages such as C++ an object field access joinpoint designator could not be reduced to a small set of locations in the source code, but would rather likely need mapped to many pointer accesses, even with the application of sophisticated points-to analyses. Because of this, AOP frameworks targeting such languages have not attempted to provide these non-code-centric joinpoint designators.

In moving away from a code-centric view in our application of AOP to runtime monitoring, we identified other *dimensions* of joinpoint designators that cannot be mapped to the code-centric weaving model but nevertheless are useful. One is the data dimension that the above example demonstrates; two other dimensions are time and probability. We defined temporal joinpoint designators that define a joinpoint at an absolute wall-clock time or relative interval-based times. For example, a joinpoint could be at 1:00am each night to run advice that would check consistency in server data structures, or might be every hour of runtime to run advice that would dump out usage statistics. These joinpoints are equally "points in the execution of a program" but cannot be related to specific locations in the code. The probability dimension enables advice execution *sampling*, not selecting points in program executions directly but rather influencing the advice execution model using criteria that are not related to the program itself, another diversion away from thinking of joinpoints as program-centric entities.

Our work thus brought new ways of thinking about how "a well-defined point in the program execution" might be defined, but as we thought of these and how to execute advice at these points, we started to formulate another notion that expanded the idea of a joinpoint.

This second and more important idea that changes the definition of joinpoint is to abandon the thinking of joinpoints as *points* in the execution. Rather, they should be thought of as abstract *intervals* of program execution whose endpoints are defined as abstract *execution conditions*. An interval might be reducible to a point but does not necessarily need to be. Indeed we think this extension actually captures the existing notions better, and the main reason for this extension is to separate the understanding of the *advice execution model* from the joinpoint model, which currently is intertwined together in the *pointcut language*.

Pointcut expressions are supposed to be abstract specifications of sets of joinpoints, but they actually also define the advice execution mechanism as well—typically using the well-known *before, after*, and *around* mechanisms. If joinpoints are points in a program execution, the keywords representing these mechanisms actually make no sense. A point in Euclidean space is a dimensionless, infinitely small location. Mapping this idea to program execution, a point is a location *between* two execution steps that change the program state. In this sense, the ideas of *before* and *after* collapse into the same thing, both being between the two execution steps surrounding the point, and *around* is meaningless since there is nothing there to envelope.

Only in thinking of a joinpoint as an interval of execution steps do these advice execution mechanisms make sense, and take on their traditional meanings. *Before* is before the interval begins, *after* is after it ends, and *around* is around the entire execution interval. Thus applying advice execution around a method call defines the joinpoint as the interval from the invocation of the method to the return of the method to its caller.

It is true that one can view the combination of the advice execution mechanism and the pointcut expression as signifying a point in the execution (e.g., *before* and *method-call* signify the point before the call), but this we think is confounding two separate "concerns" that should indeed be separate, and it means that the pointcut expression does not by itself determine points in the program execution.

Another violation of the view of a joinpoint as a point is in the treatment of threads by most AOP frameworks. In typical AOP frameworks like AspectJ, advice execution in AspectJ will occur in one thread, and other program threads will continue to execute; this means that the advice *does not* execute at a point in the program but rather during an interval of program execution, where other threads are advancing the program execution and state while the advice is executing. In reality, current joinpoint models define a joinpoint as a point in a thread's execution, not a point in a program's execution, despite the general usage of the latter.

We think our notion of a joinpoint being a program interval solves these issues. It separates the idea of a joinpoint from the advice execution mechanisms, and it can cleanly handle issues of parallelism. We think it ought to be reified into the joinpoint model, and consequently in the pointcut expression language, and will be pursuing that end in our research. Rather than always implicitly defining intervals as current pointcut languages do, the ability to abstractly specify the execution conditions for the start of the interval and separately the execution conditions for the end of the interval, will allow new and novel applications of AOP for software engineering.

Masuhara et al. [14] raised a similar issue but concluded that joinpoints *should* be points, and devised an event-based understanding of joinpoints and weaving. Although seemingly opposite of our emphasis on intervals, in reality intervals must be denoted by some notion of event at their endpoints, and so our approaches have some similarity. Recognizing the interval explicitly, however, will better support some of the issues in concurrency such as relaxing the point in time when advice might actually execute.

3. Extending the Advice Execution Model

With a new understanding of what a joinpoint is (an interval of program execution), we will pursue an expansion of the notions of what it means to weave and execute advice. Existing notions are obviously successful and useful, and map to this new understanding easily: *before* means to execute advice before the interval, *after* means to execute the advice after the interval, and *around* means to execute some part of the advice before the interval, some part after, and controlling the execution of the interval.

However, we can also imagine and define new ideas for advice execution, in particular dealing concretely with notions of concurrency. At least two notions of concurrency can be dealt with, both of which have been so far only been treated sporadically, without devising an encompassing abstraction [2, 9].

The first is with program concurrency within a multithreaded or distributed program. Advice execution has traditionally been applied to a single thread—whichever thread reached the (thread-specific) joinpoint. However, if advice needs to inspect and reason over one particular program state, all threads should be stopped during advice execution. We encountered this issue when creating time domain joinpoint designators (e.g., invoke advice at a particular wall-clock time). When the joinpoint occurs it is not particular to any single thread, so in our initial experimentation we chose to suspend all application threads. But the general issue applies even to existing joinpoint designators that are "triggered" by one thread but may need to inspect shared data before any other threads change the data.

This idea is extensible to particular thread groups, the generality being that advice execution should be defined not in relation to simply one thread but to a set of threads. This leads the joinpoint model to need joinpoint definitions over sets of threads rather than a single thread. For example, defining a joinpoint condition as when all threads reach a synchronization point is a condition over a set of threads rather than just over one. In our research we will be devising new specification methods to deal with these issues.

The second notion in concurrency is that of how advice is executed. The traditional "weaving" idea is that advice is executed in the program thread that reached the joinpoint, but there is no inherent reason this needs to be the case. Advice could be executed in its own thread while the application thread(s) continue to execute This will allow computationally intensive advice to execute without stopping the application, provided that the advice does not need a single application state snapshot. Even here, with separate conditions specifying the endpoint for the joinpoint interval, novel combinations could be made—for example, advice could execute in its own thread and application threads could continue *until* any application thread attempts to make a change in some important data structure the advice is inspecting; here the endpoint of the joinpoint interval would be a modification to the data structure.

Concurrent advice execution will enable weaving ideas such as *during*, meaning advice is executed somewhere within the joinpoint interval; if any threads reached a condition that cause them to leave the interval, they would suspend until the advice is finished. In the minimal case where the interval is defined to be a point, this captures the idea of "stop all threads" while the advice executes. Other ideas could capture soft- or hard-realtime conditions, such as executing the advice "as soon as possible" after the beginning condition of the joinpoint, or "within 10 milliseconds" of the beginning condition.

4. Related Work

There is much recent activity and novel ideas for extending AOP in a variety of manners, and several approaches to understanding the fundamental ideas in AOP that relate to the ideas we present here.

Masuhara et al. [14] raised the similar issue of whether joinpoints should be intervals or points, and they and others have continued work relating to their view of joinpoints as events [3, 13]. Others have modeled and investigated a variety of views on how AOP should work [1, 4]. Kojarski and Lorenz [12] define an elegant model to understand the pieces of AOP and how different understandings of AOP can be formally modeled; we can build on such approaches for our work. Very recently Binder et al. [5] have reiterated the need for AOP to be expanded for supporting the many issues in runtime monitoring and dynamic analysis. Dyer and Rajan [10] investigated new AOP infrastructure ideas, explicitly working on arguing for more extensive join point models (thus allowing more pointcut designators) and embodying those in an intermediate language and virtual machine support for weaving.

A very nice formal framework for *Monitor-Oriented Programming* was detailed in [8]. This work describes the monitoring task in high-level formal notations, and demonstrates how AOP can be used to provide a rigorous framework for building runtime verification analyses. The language used to describe the monitoring task may provide a good foundation for thinking about more general AOP pointcut expressions.

Much of the notion of joinpoints being intervals comes from thinking about concurrency. Even though the notion of thread and advice interaction has been touched on in some previous work [2, 7], this work did not propose a concrete model for advice execution in threads. Most popular AOP languages effectively ignore concurrency; in AspectJ the thread that reaches a joinpoint will execute the advice, and nothing is specified beyond that. Ansaloni and Binder [2] introduced a framework that enables asynchronous advice execution by collecting advice invocations in a buffer and executing the advices in a separate thread. Douence et al. [9] created Concurrent Event-based AOP (CEAOP) which defines concurrent AOP using labeled transition systems, and overcomes some of the limitations of synchronization and thread communications. CEAOP supports concurrency in the underlying program, and concurrent execution of advice with the base program by allowing the advice body executions in parallel.

5. Conclusion

Fundamental to AOP is the notion of a joinpoint, which has traditionally been thought of as a point in the program execution that can practically be mapped to points in the program itself. We argued here that the notion of a joinpoint really should be a defined interval of program execution whose defining conditions may or may not be related to loci in the program itself. This change better captures existing usage of AOP, in particular with the around advice application and the treatment of threads, and it allows thinking about new and novel directions for AOP. We also argue here that this change in joinpoint thinking also helps to better separate advice execution from pointcut definition, and allows the imagination of new types of advice execution that have wide applicability in the software engineering field.

In this paper we are only presenting ideas of how the abstractions in AOP might be generalized and expanded. We are not attempting to claim that it is straightforward to put these ideas into practice. Indeed the existing ideas within AOP are arguably there because it was somewhat obvious how to implement them efficiently. Other ideas have followed similar paths. The idea of tracematches as joinpoint designators was introduced first, then followed by work in optimizing their detection [6].

Working out other details as well, such as how to create a usable advice language that embodies these ideas without overwhelming the user, may be problematic. Certainly some of the success of AOP, as with other technologies, is found in the combination of power and simplicity. Yet the continuing research in AOP should consider breaking away from ideas that limit what can be imagined and implemented, and should explore radical new and novel approaches to AOP.

References

[1] M. Achenbach and K. Ostermann. A Meta-Aspect Protocol for Developing Dynamic Analyses. In *Proc. 1st Int'l Conf. on Runtime Verification*, pages 153–167, Berlin, Heidelberg, 2010. Springer-Verlag.

[2] D. Ansaloni, W. Binder, A. Villazón, and P. Moret. Parallel Dynamic Analysis on Multicores with Aspect-Oriented Programming. In *2010 Conference on Aspect-Oriented Software Development (AOSD)*, pages 1–12, 2010.

[3] T. Aotani and H. Masuhara. SCoPE: an AspectJ Compiler for Supporting User-Defined Analysis-Based Pointcuts. In *Proc. 6th Int'l Conf. on Aspect-Oriented Software Development*, pages 161–172, New York, 2007. ACM.

[4] P. Avgustinov, T. Ekman, and J. Tibble. Modularity First: A Case for Mixing AOP and Attribute Grammars. In *Proc. 7th Int'l Conf. on Aspect-Oriented Software Development*, AOSD '08, pages 25–35, New York, 2008. ACM.

[5] W. Binder, P. Moret, D. Ansaloni, A. Sarimbekov, A. Yokokawa, and E. Tanter. Towards a Domain-Specific Aspect Language for Dynamic Program Analysis: Position Paper. In *Proc. 6th Workshop on Domain-Specific Aspect Languages*, pages 9–11, New York, 2011. ACM.

[6] E. Bodden, L. Hendren, and O. Lhoták. A Staged Static Program Analysis to Improve the Performance of Runtime Monitoring. In *ECOOP*, pages 525–549, 2007.

[7] W. Cazzola, A. Cicchetti, and A. Pierantonio. Towards a Model-Driven Join Point Model. In *Proceedings of the 2006 ACM Symposium on Applied Computing*, SAC '06, pages 1306–1307, New York, 2006. ACM.

[8] F. Chen and G. Roşu. MOP: an Efficient and Generic Runtime Verification Framework. In *OOPSLA '07: Proc. 22nd ACM SIGPLAN Conf. on Object Oriented Programming, Systems, and Applications*, pages 569–588, New York, NY, USA, 2007. ACM.

[9] R. Douence, D. Le Botlan, J. Noyé, and M. Südholt. Concurrent Aspects. In *Proceedings of the 5th International Conference on Generative Programming and Component Engineering*, GPCE '06, pages 79–88, New York, 2006. ACM.

[10] R. Dyer and H. Rajan. Nu: a Dynamic Aspect-Oriented Intermediate Language Model and Virtual Machine for Flexible Runtime Adaptation. In *Proc. 7th Int'l Conf. on Aspect-Oriented Software Devel.*, pages 191–202. ACM, 2008.

[11] G. Kiczales and M. Mezini. Aspect-Oriented Programming and Modular Reasoning. In *Proceedings of the 27th International Conference on Software Engineering*, ICSE '05, pages 49–58, New York, 2005. ACM.

[12] S. Kojarski and D. H. Lorenz. Modeling Aspect Mechanisms: a Top-Down Approach. In *Proc. 28th International Conference on Software Engineering*, pages 212–221, New York, 2006. ACM.

[13] E. Marques, L. Veiga, and P. Ferreira. An Extensible Framework for Middleware Design Based on Concurrent Event-Based AOP. In *Proc. 9th Int'l Workshop on Adaptive and Reflective Middleware*, pages 26–31, New York, 2010. ACM.

[14] H. Masuhara, Y. Endoh, and A. Yonezawa. A Fine-Grained Join Point Model for More Reusable Aspects. In *Proc. Fourth ASIAN Symposium on Programming Languages and Systems, (LNCS 4279)*, pages 131–147, Nov. 2006.

Aspect-Orientation in the Development of Embedded Systems: A Systematic Review

Leonardo Simas Duarte and Elisa Yumi Nakagawa
Department of Computer Systems, University of São Paulo - USP
PO Box 668, 13560-970, São Carlos, SP, Brazil
{*ldu, elisa*}*@icmc.usp.br*

***Abstract*—Currently, a diversity of embedded systems has been produced, from systems for consumer electronics to critical environments, causing a considerable impact to the society. In another perspective, aspect-orientation approach has arisen, intending to contribute to the development of reusable, maintainable, and evolvable software systems. Considering its relevance, this approach has been also applied to the development of embedded systems. However, there is not a complete panorama about how aspect-orientation has been explored in the development of such systems. Thus, this paper aims at exploring, organizing, and summarizing the contributions about the use of aspect-orientation in the development of embedded systems. For this, we conducted a systematic review that is a technique that provides an overview of a research area to assess the quantify of evidences existing on a topic of interest. As main result, we have observed that in the last years considerable knowledge related to the aspect-orientation in the embedded system development has been accumulated; however, more studies must be conducted yet. Furthermore, this work intends to contribute to the identification of interesting and important open research areas.**

***Keywords*-Embedded system; aspect-orientation; systematic review.**

I. Introduction

Embedded systems refer to computing systems designed to dedicated features, sometimes as part of a complete device[14]. Currently, a considerable number of embedded systems has been developed, such as for automobiles, aircrafts, PDAs (Personal Digital Assistants), digital decoders, and mobile devices. Furthermore, these systems must consider the constant growing of processing power, data storage and graphic capacity, platform convergence and value-added features, such as location-based and interactive multimedia services.

In another perspective, Aspect-Oriented Programming (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 [7], [8]. 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 facility to write software can be achieved with AOP [10].

Considering the relevance of AOP, initiatives to the development of embedded systems using AOP have more and more emerged. Different studies can be found, such as comparisons of AOP and OOP (Object-Oriented Programming) to develop such systems [12]. However, there is a lack of a detailed panorama about why and how AOP has been adopted to the development of embedded systems. Thus, the main objective of this paper is to explore, organize, and summarize the contributions about the use of AOP in embedded systems.

This paper is organized as follows. In Section II we present the conducted systematic review. In Section III we discuss about achieved results and summarize our contributions.

II. Conducted Systematic Review

Our systematic review was conducted on a hybrid domain involving Aspect-Orientation and Embedded Systems, aiming at identifying possibly all primary studies (i.e., a case study or an experimental study divulged in a publication) that explore the use of AOP in the embedded system development. This systematic review was carried out by three people (one researcher in software engineering, one specialist in systematic review, and one graduate student). In order to conduct our systematic review, we have followed the process proposed by Kitchenham [9], illustrated in Figure 1. Following, we detail each step:

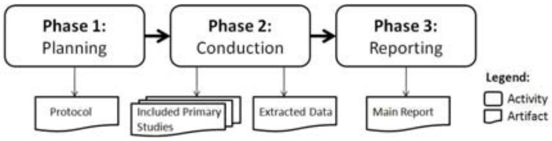


Figure 1. Systematic Review Process (Adapted from [9])

A. Step 1: Systematic Review Planning

In this step, 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.

Research Questions: These questions are structured corresponding to the objective that is intended with the systematic review and drive the review for further steps. Aiming at discovering all primary studies to understand and summarize about AOP in the embedded system development, the following research questions (RQ) were established: (i) ***RQ 1:*** What is the current state of the adoption of AOP in the development of embedded systems? (ii) ***RQ 2:*** How AOP has been used in the embedded system development? (iii) ***RQ 3:*** What are the benefits and limitations by using AOP in embedded systems?

Search Strategy: In order to establish the search strategy, we identified then initially specific keywords for the two research fields: AOP and Embedded System. Thus, for AOP field, we identified the terms "AOP", "AO", "crosscutting concern", and "cross-cutting concern". Following, we found synonyms for these keywords: "aspect oriented programming", "aspect oriented software", "aspect oriented application", "aspect oriented app", "aspect oriented program", "aspect orientation", and "aspect-based". According to the systematic review specialist, a *technological term* "AspectJ" was included. Furthermore, for Embedded Systems field, the following keywords were used: "reactive system", "real-time system", "embedded", "cyber-physical system", "pervasive system", "ubiquitous system", "wearable computer", and "consumer electronic". In addition to the search string, we selected larger publications databases as sources of primary studies: IEEE Xplore[1], ACM Digital Library[2], Springer Link[3], Scirus[4], ScienceDirect[5], and SCOPUS[6].

Inclusion and Exclusion Criteria: Two basic parameters that play an important role at the systematic review planning is to define the Inclusion Criteria (IC) and Exclusion Criteria (EC): (i) ***IC 1:*** The primary study explore the use of AOP in the embedded system development; (ii) ***IC 2:*** The primary study presents how AOP has been used in the embedded system development; and (iii) ***IC 3:*** The primary study presents benefits and limitations provided by the use of AOP in embedded systems; (iv) ***EC 1:*** The primary study does not address AOP in embedded system development; (v) ***EC 2:*** The primary study presents an abstract and/or an introductory section that seem related to AOP in embedded system development; however, the rest of the text is not in fact related to; and (vi) ***EC 3:*** The primary study does not present any abstract or it is not available for further reading.

Data Extraction and Synthesis Method: To extract data, we use control tables to each research question. These tables summarize results aiming at facilitating to obtain conclusions. The data of each primary study will be independently extracted by two reviewers. If disagreement occurs, discussion will be conducted.

B. *Step 2: Systematic Review Conduction*

The search by primary studies was conducted according to previously established planning. This identification was made by looking for all primary studies that match with the search string on the selected databases. As result, a total of 297 studies were identified, all studies were considered primary even in the event of repetition between the databases, in order to identify the relevance of each database, as shown in Table I. Following, through reading of abstracts and application of the inclusion and exclusion criteria, 104 studies were then selected to be fully read. At the end, 25 studies, summarized in Table II, were considered as relevant to our systematic review. Following, a more detailed analysis was conducted on the 25 primary studies included in our systematic review and data were extracted.

Table I
DATABASE RELEVANCE

Database	Obtained	Included	Percentage Index
IEEE	44	10	22.7%
ACM	77	6	7.8%
Scopus	79	5	6.3%
Springerlink	17	1	5.9%
ISI Web	11	1	9.0%
Scirus	69	2	2.9%

C. *Step 3: Systematic Review Reporting*

The systematic review provides us a perspective of how AOP has been explored in embedded system development. Through primary studies included in our systematic review, we have identified six research areas: OOP versus AOP when used in embedded system development (S2, S8, S10, S23 and S24), AOP used at distributed embedded systems (S1, S2, S5, S6, S9, S12, S22, S23, S24 and S25), AOP used in consumer electronics (S8, S10, S24 and [11] by specialist recommendation), AOP used in software development process for embedded systems (S1, S9 and S23), SoC used in the architectures of embedded systems (S1, S5, S11, S18, S24 and S25), and AOP for real-time embedded systems (S2, S4, S5, S6, S9, S10, S11, S12, S15, S17, S20, S22 and S23). We have also examined the maturity of the ideas and concepts, as well as the status of their applications by looking into the research methods used in each study's validation. It is also observed that the primary studies related to AOP in the context of embedded systems are concentrated in the last years. Figure 2 shows the number of primary studies distributed through the years.

The research area which aggregates the most number of studies is the use of AOP on real-time embedded systems; two main studies present different initiatives to apply AOP: [1] and [4]. The work of Beuche et. al [1] show the use of AOP on the operational system layer, exploring customization and reuse. The usage of aspects imprints the

[1] http://www.ieeexplore.ieee.org
[2] http://www.portal.acm.org
[3] http://www.springer.com/lncs
[4] http://www.scirus.com/
[5] http://www.sciencedirect.com
[6] http://www.scopus.com

Table II
SELECTED PRIMARY STUDIES

#	Authors/Title	Year
S1	Aoyama, M. and Yoshino, A., Aspect-Oriented Requirements Engineering Methodology for Automotive Software Product Lines.	2008
S2	Beuche, D., Spinczyk, O., and Schröder-Preikschat, W., Finegrain application specific customization for embedded software.	2002
S3	Beuche, D., Guerrout, A., Papajewski, H., Schröder-Preikschat, W., Spinczyk, O., and Spinczyk, U., The pure family of object-oriented operating systems for deeply embedded systems.	1999
S4	Pericles, L. and Papadopoulos, G., Using aspect-oriented software development in real-time systems software: A review of scheduling, resource allocation and synchronization.	2006
S5	Deng, G., Schmidt, D. C., and Gokhale, A., Addressing crosscutting deployment and configuration concerns of distributed realtime and embedded systems via aspect-oriented & model-driven software development.	2006
S6	Freitas, E., Wehrmeister, M., Pereira, C., Wagner, F., Silva, E., and Carvalho, F., Using aspect-oriented concepts in the requirements analysis of distributed real-time embedded systems.	2007
S7	Haupt, M. and M., M., Virtual Machine Support for Aspects with Advice Instance Tables.	2005
S8	Hundt, C.; Glesner, S., Optimizing aspectual execution mechanisms for embedded.	2009
S9	Jingyong, L., Yong, Z., Lichen, Z., and Yong, C., Aspect-oriented middleware-based real-time and embedded systems software process.	2009
S10	KARTAL, Y. and SCHMIDT, E., An evaluation of aspect oriented programming for embedded real-time systems.	2007
S11	Zhang, L. and Liu, R., Aspect-oriented real-time system modeling method based on UML.	2005
S12	Mouavi, M., Russello, G., Chaudron, M., Reniers, M., Basten, T., Corsaro, A., Shukla, S., Gupta, R., and Schmidt, D., Using aspect-gamma in the design of embedded systems.	2002
S13	Machta, N., Benani, T., and Benahmed, S., Weaving real-time constraints on behavioral and structural application model.	2009
S14	Stankovic, J. A., Nagaraddi, P., Yu, Z., and He, Z., Exploiting prescriptive aspects: a design time capability.	2004
S15	Noda, N. and Kishi, T., Aspect-oriented modeling for embedded software design.	2007
S16	Noro, M., Sawada, A., Hachisu, Y., and Banno, M., E-AoSAS++ and its software development environment.	2007
S17	Roychoudhury, S., Bunse, C., and Höpfner, H., Applying state-of-the-art techniques for embedded software adaptation.	2009
S18	Schimdt, P., Milstein, J., and Alvarado, S., Architectural assessment of embedded systems using aspect-oriented programming principles.	2005
S19	Spinczyk, O. and Lohmann, D., The design and implementation of aspectc++.	2007
S20	Tsang, S., Clarke, S., and Baniassad, E., An evaluation of aspect-oriented programming for Java-based real-time systems development.	2004
S21	Tuohimaa, S. and Leppänen, V., A compact aspect-based security monitor for J2ME applications.	2007
S22	Wehrmeister, M., Freitas, E., Pereira, C., and Wagner, F., An aspect-oriented approach for dealing with non-functional requirements in a model-driven development of distributed embedded real-time systems.	2007
S23	Wehrmeister, M., Freitas, E., Orfanus, D., Pereira, C., and Rammig, F., Evaluating aspect and object-oriented concepts to model distributed embedded real-time systems using rt-uml.	2008
S24	Fanjiang, Y., Kuo, J., Ma, S., and Huang, W., An aspect-oriented approach for mobile embedded software modeling.	2010
S25	Lohmann, D., and Spinczyk, O., Ciao: An aspect-oriented operating-system family for resource-constrained embedded systems.	2009

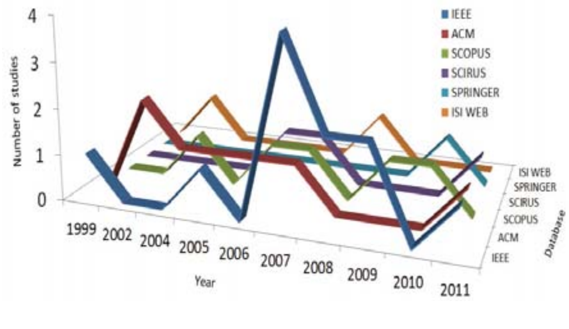


Figure 2. Distribution of the primary studies

crosscutting properties to be concentrated in the aspect, allowing easier derivation of future applications, like the use of AOP for treatment of threads, exceptions, and process scheduling, which are primary features, given the resource constraints of embedded systems. Another study [4] presents the development of the main features of a virtual machine[7] with the use of aspects to agglutinate instance tables for Java language. This work explored execution runtime, library consulting, and component constructions. This research was distinguished from the others, because it used for the first time aspect-orientation approach on a user layer level in embedded systems, bringing future investigations of this scenario.

Primary studies present also comparative results between OOP and AOP when used in the context of embedded systems. At the study of [6], it is presented an experimental evaluation of both paradigms for real-time embedded systems, applying qualitative and quantitative metrics. Positive results using AOP were achieved, where used on the operational system layer. In another work [3], AOP was used at the requirement analysis, eliciting and storing knowledge of the specific components to be hold as crosscutting concerns.

[7] Software implementation of a machine (i.e., a computer) that executes programs like a physical machine.

In another perspective, [2], [5], [6] and [11] explored AOP in the development of embedded systems for consumer electronics, such as mobiles. These works are mainly related to consumer's needs, resources constraints, software product line, and usability. In particular, Kartal and Schmidt [6] conducted an investigated involving a platform of real-time embedded systems for consumer electronics, specifically the use of AOP at the application layer of such systems.

At the end, it becomes clear by the results of our systematic review that, although there are various studies that explore the use of AOP in the development of embedded systems, there is a lack of synergy between both topics (AOP and embedded systems), in compliance with the fragmentation of the studies and their multidisciplinary. This can be observed in [13] and [3]. For the lack of space, a discussion about other areas are not presented herein.

III. Conclusion

In the last years, embedded system development has been an increasing, important concern, bringing considerable challenges to both academy and industry. Advances in technology, for instance, increase in processing power, together with new requirements and needs of users, as well as shorter time-to-market and better quality, have contributed to difficult this development. New development approaches must be then investigated, such as the use of AOP. Thus, main contribution of this paper was to present a detailed panorama about how AOP has been explored in the development of embedded systems. This review identified 25 primary studies that have explored the use of AOP in the embedded system context. Furthermore, they are classified into six main research areas, what can show the variability of research that has been conducted. In spite of different initiatives in that direction, we have observed that there is a need of more studies that consolidate the advantages and limitations of AOP in embedded systems.

Regarding research questions established for our systematic review, it is observed that all of them were answered. This suggests that, the general, knowledge about AOP in the embedded system development 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.

Considering knowledge arisen from this work, it is possible to identify interesting and new research lines, such as (i) to explore the use of AOP in specific types of embedded systems, such as critical or safety-critical systems; and (ii) to conduct comparative studies using aspect-orientation approach in different layers of the architecture of embedded systems. For the future work, we intend to conduct again this systematic review in order to update results of this work.

Acknowledgment: The authors would like to thank the Brazilian funding agencies CNPQ, FAPESP, and FAPEAM.

References

[1] D. Beuche, A. Guerrout, H. Papajewski, W. Schröder-Preikschat, O. Spinczyk, and U. Spinczyk. The pure family of object-oriented operating systems for deeply embedded systems. In *ISORC'1999*, pages 45–53, Saint Malo, France, 1999.

[2] Y. Fanjiang, J. Kuo, S. Ma, and W. Huang. An aspect-oriented approach for mobile embedded software modeling. In *ICCSA'10*, pages 257–272, Fukuoka, Japan, 2010.

[3] E. Freitas, M. Wehrmeister, C. Pereira, F. Wagner, E. Silva, and F. Carvalho. Using aspect-oriented concepts in the requirements analysis of distributed real-time embedded systems. In A. Rettberg, M. Zanella, R. Dömer, A. Gerstlauer, and F. Rammig, editors, *Embedded System Design: Topics, Techniques and Trends*, volume 231 of *IFIP*, pages 221–230. 2007.

[4] M. Haupt and M. Mezini. *Virtual Machine Support for Aspects with Advice Instance Tables*. Phd thesis, Technical University of Darmstadt, Germany, 2005.

[5] C. Hundt and S. Glesner. Optimizing aspectual execution mechanisms for embedded. In *Journal Electronic Notes in Theoretical Computer Science*, number 2, pages 35–45, Berlin, Germany, 2009. Elsevier.

[6] Y. B. Kartal and E. G. Schmidt. An evaluation of aspect oriented programming for embedded real-time systems. In *ISCIS'2007*, pages 1–6, Ankara, Turkey, 2007.

[7] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. G. Griswold. Getting started with AspectJ. *Communications of the ACM*, 44(10):59–65, 2001.

[8] G. Kiczales, J. Irwin, J. Lamping, J. Loingtier, C. Lopes, C. Maeda, and A. Menhdhekar. Aspect-oriented programming. In *ECOOP'1997*, pages 220–242, Jyväskylä, Finland, 1997.

[9] B. Kitchenham. Procedures for performing systematic reviews. Technical Report TR/SE-0401, Keele University, UK, 2004.

[10] R. Laddad. Aspect-oriented programming will improve quality. *IEEE Software*, 20(6):90–91, 2003.

[11] D. Saraiva, L. Pereira, T. Batista, F. C. Delicato, and P. F. Pires. Architecting a model-driven aspect-oriented product line for a digital tv middleware: a refactoring experience. In *ECSA'2010*, pages 166–181, Kopenhagen, Dinamark, 2010.

[12] M. A. Wehrmeister, E. Freitas, D. Orfanus, C. E. Pereira, and F. Rammig. Evaluating aspect and object-oriented concepts to model distributed embedded real-time systems using rt-uml. In *IFAC'2008*, pages 44–54, Seoul, Korea, 2008.

[13] M.A. Wehrmeister, E. Freitas, C.E. Pereira, and F. Wagner. An aspect-oriented approach for dealing with non-functional requirements in a model-driven development of distributed embedded real-time systems. In *ISORC'2007*, pages 428–432, Vienna, Austria, 2007.

[14] W. H. Wolf. *Computers as components: principles of embedded computing system design*. Morgan Kaufmann, 2001.

Evaluating Open Source Reverse Engineering Tools for Teaching Software Engineering

Swapna S. Gokhale, Thérèse Smith, and Robert McCartney

Department of Computer Science & Engineering
University of Connecticut, Storrs, CT 06269-2155
E-mail: {ssg,tms08012,robert}@engr.uconn.edu

Abstract

This paper reports our experiences and lessons in the evaluation of open source, reverse engineering tools to teach Software Engineering (SE). The chosen tools were expected to meet a dual objective, namely: (i) teach students software comprehension, maintenance and evolutions skills; and (ii) train them in modern SE tools. A majority of the tools were disappointing because they did not deliver their promised functionality. Moreover, the tool that was ultimately selected because it appeared to function as stated, posed unexpected problems. We summarize our lessons, which we hope will inform others who wish to undertake a similar exercise.

Keywords

Software Engineering Education, Tools, Reverse Engineering

1 Introduction and Motivation

The software industry has witnessed phenomenal growth over the past two decades. Starting from its humble beginnings in the 1950s [17], today it is worth more than $233 billion with a projected annual growth rate of 17% [23]. This dramatic explosion has created an extensive need for skilled and talented software engineers. To respond to this need, most educational institutions now include Software Engineering (SE) as a mandatory course in their computing curriculum.

The two key realities of today's software development environment include: (a) multiple implementation technologies; and (b) the need for maintenance over extended lifetimes [28]. As a result, students hoping to work as software engineers can expect to maintain and evolve large, complex legacy software. Therefore, one of the primary objectives of the SE course should be to teach students how to comprehend and evolve existing code. Development of several subsidiary supporting competencies is implied. For example, the students should be able to: (i) extract design decisions and rationale from the code in a top-down, organized manner using abstractions and tools; (ii) distinguish between well- and poorly-specified requirements, and especially identify those that will be difficult to implement given the structure and organization of the existing code base; and (iii) produce documentation of good quality explaining the design rationale.

The software industry commonly uses various tools to perform software development activities. Some of these tools improve efficiency and quality of management and housekeeping tasks; for example, defect tracking and cost estimation tools [13]. Some tools, on the other hand, support core design, implementation and testing functions; for example, configuration and version control, test case generation and reverse engineering tools [13]. Increasingly advanced tools continue to be developed, however, software engineers show tremendous resistance to their adoption. This resistance, in part, could be attributed to the sharp learning curve associated with these tools [9]. Alleviating this resistance can encourage the adoption of these tools, and ultimately increase the efficiency and reduce the costs associated with software development and maintenance. A possible way to mitigate this resistance may be to expose and train students in the different types of SE tools during their school projects.

Our SE course thus has a two-fold objective, namely, to teach students maintenance and evolution skills, while training them in modern SE tools. To meet this dual objective, we sought to evaluate contemporary, open source tools that would reverse engineer UML diagrams from the source code. What we expected to be relatively straightforward, given the plethora of open source UML tools, turned out to be rather cumbersome and tedious. Specifically, we observed that many tools disappointed in delivering their promised functionality. Moreover, the tool that was ultimately chosen because it performed smoothly during evaluation, posed unexpected problems during actual laboratory assignments. We disseminate our experiences and lessons through this paper, hoping to inform others who may wish to undertake a similar evaluation.

The paper is organized as follows: Section 2 outlines the selection criteria. Section 3 presents tool selection and evaluation. Section 4 shares our experiences and students' reactions. Section 5 summarizes the lessons learned. Section 6 surveys contemporary tools. Section 7 concludes and offers future directions.

2 Selection Criteria

Various types of tools may be used in SE depending on its emphasis, which in turn depends on when the course appears in the curriculum. SE may be offered according to two broad models, namely, "SE early" and "SE late". In the early model, SE is offered to sophomores and juniors, and most often represents the students' first opportunity to get acquainted with the tools used in the construction of software systems. In the late model, SE is offered to seniors and may even be merged with the capstone project. SE late students may have had opportunities to acquire experience in the processes and tools necessary to build software.

The tools in the SE early approach may cater towards foundational software development skills, including requirements analysis, design, testing, and configuration management. On the other hand, the late SE course may introduce students to specialized, sophisticated tools such as those used for debugging, defect tracking, cost estimation, automatic code generation and project management. Our approach is SE early and we sought to expose our students to configuration management and reverse engineering tools. To shortlist these two types of tools for further experimentation, we defined the following criteria:

1. Integrated Environment: Integrated development environments (IDEs) are recommended in the industry [10] because such environments, where a collection of tools interoperate [24], can support cross-process improvements [10]. Studies also show that practicing software engineers prefer integrated tools [18]. Finally, academics report that teaching with integrated tools can be less distracting from the course objectives [19]. Thus, we choose to teach SE using an Integrated Development Environment (IDE).

Our choice of IDE is Eclipse because students have used it in a previous course, thus we prefer tools that integrate with Eclipse. Even though we may pick tools from multiple sources, our first criterion is that the impression of an integrated workstation for interaction with the code must be maintained through Eclipse and its plug-ins. Moreover, the plug-ins should work with Java, which is the language of the course projects, since it is the only one commonly known to the students.

2. Open Source: To teach students code comprehension, maintenance and evolution, a supply of code which represents legacy software is necessary. Aspects of this representative software include that it is written by someone else, preferably a team, has perhaps only moderately good documentation and design, and does not follow a uniform style. A certain level of complexity is desirable, as is appeal to the students. The course projects are based on open source software because it offers a rich volume of code that exhibits these characteristics. Therefore, a significant criterion is that the tools themselves are open source.

3. Unified Process (UML): The course exercises a software process, similar to IBM Rational Unified Process [34], because of UML's industrial and academic popularity [16, 29, 33]. The students are expected to learn class and sequence diagrams. Thus, the chosen tools should be capable of drawing, and wherever appropriate, extracting these diagrams from the code, which we refer to as "reverse engineering". Static class diagrams can be extracted without executing the code, whereas, dynamic sequence diagrams can be extracted only by executing the code.

3 Tool selection and evaluation

We short listed several configuration management and reverse engineering tools for further exploration. This section summarizes these tools and their evaluation results, subject to the criteria defined in Section 2.

CVS, SVN and Git were the candidate configration management tools. We included CVS [5] because it was available in Eclipse as installed in the lab, and SVN [27] because it was familiar to the course staff. Although Git [14] was neither installed nor familiar, we included it because it was comparatively more modern. Finally, we chose Git and Eclipse plug-in EGit because of their distrbuted design, scalability, and support from a worldwide community [14]. This distributed design would allow students to update from one another, independently of any central repository.

Table 1. Candidate reverse engineering tools

Tool	Reason for elimination
Amateras [2]	Extracts class diagrams
ArgoUML [3]	Standalone
ATL [4]	Not specialized to UML
BoUML [6]	Not free
Diver [8]	Extracts sequence diagrams
UML2 Tools [31]	Incompatible with Eclipse Indigo
Fujaba [11]	Different interpretation
Green UML [15]	Promising, but crashed
Open Model Sphere [20]	Standalone
Papyrus [21]	Extracts class diagrams
StarUML [26]	Standalone
Umbrello UML Modeller [30]	Not windows-based
Violet [32]	Exemplar, no reverse engineering

Preliminarily, we discovered many reverse engineering tools; some of these were created for educational purposes, while others were intended for a different audience. An examination of this set led to a short list of candidate reverse engineering tools, summarized in Table 1. Our further investigation eliminated most of these tools because they failed to meet one or more of the selection criteria, and these reasons are also summarized in the table. Violet represents a large class of tools that does not offer reverse engineering capabilities. BoUML had been free, but was no longer so when we actually experimented with it. ArgoUML and OpenSphere were standalone and not Eclipse plug-ins. StarUML used a different interpretation of reverse engineering from ours; it simply involved generating a message on to a sequence diagram by clicking on a method call. Fujaba's documentation was out of date, and it had limited capabilities when integrated into Eclipse. ATL was not specialized for UML diagrams, rather it was a model transformation tool. UML2Tools was too old, and not compatible with Eclipse Indigo. Umbrello was eliminated because it was not Microsoft Windows-based. Finally, GreenUML appeared promising, however, it crashed with one of the course projects.

Our exploration did not lead to a single Eclipse plug-in that could extract both static class and dynamic sequence diagrams. Amateras was selected to extract class diagrams. Papyrus was an option to substitute for Amateras. The only free, Eclipse plug-in that actually extracted dynamic sequence diagrams was Diver. Diver's documentation was decent and it had been runner up for an Eclipse community award, so it seemed more than adequate. It also worked in conjunction with all the course projects. All these tools integrated with Eclipse, and offered an added advantage of being intended for industrial use.

4. Experiences and Reactions

This section reports experiences and difficulties in using the chosen tools. It also summarizes the students' reactions as they experimented with these tools.

4.1 Experiences and Difficulties

In this section, we describe the challenges involved in set up, installation, and use of the tools. We also discuss the approaches taken to resolve these challenges.

Integrated Development Environment We chose Eclipse as the IDE because of student familiarity, but the thought of mandating it had not occurred. Several student teams chose to use NetBeans instead of Eclipse. Thus, they used Eclipse to complete their assignments as the reverse engineering tools were reached through Eclipse, and ended up in using two different development environments.

Configuration Management Tools The presemester setup of Git and EGit with Eclipse was complicated by the multiple components of the tool set. Data was conveyed easily across each individual interface, but representations of data to be entered on one end of the tool chain and consumed at an end farther away than the next immediate tool were not always easy to deduce. Installation of this tool chain had to be worked out through a combination of Internet search and extensive trial and error.

Git and EGit provided the students with a seamless integration with Eclipse. Prepared keystroke/click by keystroke/click instructions showed them how to use it. Once again, the use of Git and Egit was not incentivized, i.e., no contribution to the final grade encouraged them to use these tools. Thus, the students chose whatever they found to be the easiest way to collaborate, which included emailing each other the project code, and storing it informally.

Reverse Engineering Tools The first reverse engineering assignment was to extract static class diagrams using either Amateras or Papyrus. Using these tools was a simple matter of opening the project from the repository to view the Java classes, then dragging and dropping these classes onto a diagram canvas. Most students chose Amateras because they found it extremely easy. This is not surprising, as dragging from a list onto a canvas produces the class, and the tool automatically provides associations such as inheritance. The diagram produced by the tool was visually appealing. This experience suggests that the assignment should ask more than an intelligently arranged display of classes. For example, the students may be asked to comment upon class containment and specialization hierarchies, or to search for implementations of patterns such as the model-view-controller [12].

The next reverse engineering assignment involved drawing sequence diagrams for specifying requirements. The students were expected to describe how a planned modification of their software would be implemented with communicating classes. Both Papyrus and Amateras provided the ability to draw sequence diagrams from dragging classes and manually entering messages. The set up for Amateras is simple. One places the jar files in a directory within the structure for Eclipse, starts Eclipse, and finds the capability in place. Papyrus was even easier to install, following the normal Eclipse procedure to install new software.

The last assignment involved extracting dynamic sequence diagrams from the code. Diver, the chosen tool, posed several problems, although it was tested on two operating systems (Windows/XP SP3 and Vista) and on all twenty course projects. Moreover, the test of the installation on laboratory computers, which were Windows 7, had proceeded far enough to watch the collection of events. This testing, however, was not sufficient and a failure mode surfaced where the events were collected but not being saved. Availability of the source code was helpful in debugging, and we discovered where the files were being saved from two working installations. We first conjectured that a needed write-privilege was absent, however, that was not the case.

We then contacted the developers at University of Victoria, who were very helpful. Simultaneously, we investigated the problem in house, in coordination with the course staff and the staff responsible for maintaining laboratory computers; the latter finally resolved the issue. Most students could now generate and record sequence diagrams. On a subset of the projects, a second difficulty manifested where the ability to restrict the tracing events to modules did not seem to work, though this might be due to a lack of understanding about what is required for this function. For this subset of projects, students collected events that were generic to the Java libraries. They did not see sequences of calls between top-level objects in their projects.

4.2 Students' Reactions

The students were exposed to one configuration management and three reverse engineering tools. They successfully reverse engineered their projects using these tools, though with different degrees of support. They became able, during the course, to express themselves at this level of abstraction.

The students were delighted to learn the value of sequence diagrams in understanding their code, and reported that they wished they had learned about dynamic analysis earlier. The initial resistance, showed by some students, dissipated quickly, after they had the opportunity to study the sequence diagrams produced by the tool. Some students found the notation (object instance, timeline, lifeline) in sequence diagrams sufficiently unintuitive to recognize it right away. Thus, each student team was given a brief tutorial to the sequence diagram drawing tool. The tutorial also demonstrated how constructors or how object construction appeared on the diagram. Overall, the students enjoyed sequence diagrams more than class diagrams, probably because they had not seen this diagram before, and they became involved with annotating the messages with the note tool. By comparison, the class diagrams gave them almost no creative activity.

Retrospectively, we felt that the students would have better appreciated the value of sequence diagrams if they had extracted these before drawing one for their planned enhancement. This extraction may have produced a detailed understanding of the code, and led to more meaningful and interesting enhancements. Although the assignments were handed in this order, the time consuming difficulties encountered with Diver forced us to flip their series of completion.

5 Lessons Learned

In this section, we summarize the lessons learned from the exercise of selecting, evaluating and experimenting with open source tools to teach SE.

- **Installation and set up:** It took excessive time to install the set of tools that ultimately worked together to provide a laboratory environment. Tool chains, where each tool complies with the interfaces of its neighbors, may lack documentation about data transformations, and hence, may require cumbersome trial and error.
- **Functions and capabilities:** The definitions of terms may not be consistent and uniform. Thus, although a given tool promises a particular function, its specific implementation may be different.

Some tools may perform the needed functions, but only in versions incompatible with the laboratory set up. Thus, sufficient time must be allocated to experiment and ensure each tool's capabilities.

- **Debugging:** Subtle bugs can exist, such that the tool works fine in one environment, but not the needed environment. Open source tools may be beneficial, should the need to debug arise.
- **Incentivization:** Students' preferences for development environments and collaborative styles may be different than those planned for the course. It is recommended to provide an incentive (grade points) for the use of tools that are strongly preferred, perhaps from the need to enforce consistency and uniformity, or for the sake of efficiency. Without incentivization, students can wander off in unanticipated directions, giving them less of the planned valuable experience and resulting in a marginally satisfactory outcome.
- **Exploratory assignments:** The assignments should expect students to go beyond what can be easily and routinely accomplished by the tool. Such creative assignments will teach them how to explore and learn a tool, and gain insights into the value of the abstraction implemented by the tool.

6 Contemporary Tools

This section surveys the approaches taken to select tools to teach SE. Some institutions select tools that are designed for the industry. An Internet examination of the syllabi of SE courses suggests that many open source, industrially relevant tools exist, and have been found useful in instruction. These tools, each addressing an important and interesting facet of SE, are so numerous that one could imagine using a different one each week of the semester, for example [22, 25]. Innovative use of these tools occurs as well, for example de Marcos et al. [7] report a combined informatics and philosophy course using StarUML.

A completely different approach is to abjure tools suitable for the industry because they take too long to learn, or misdirect students' focus, and instead develop a lightweight tool set that supports the software development process, and in doing so, help novices develop an understanding and awareness of the principles and practices as they work [1]. Yelmo et al. [35] adopt this approach and use a tailored, lightweight version of the IBM Rational Unified Process. They also develop a web-based tool to adapt RUP to a specific project.

Our approach can be considered to be a hybrid of the above two strategies. Because we lacked the resources to develop a custom, lightweight tool set, we sought to evaluate existing tools. Our search was narrowed to open source tools, not only because it was the theme of the course, but also because the availability of the source code offered the potential to customize the tools to meet our objectives. Finally, although industrial relevance was not a significant selection or evaluation criteria, the chosen tools were intended to be used in the industry. This industrial relevance was a plus because it can conceivably help students when they seek employment as software engineers.

7. Conclusions and Future Directions

We reported our experiences and lessons in evaluating open source tools to teach SE. Despite the myriad of tools, this evaluation was cumbersome because many tools did not deliver their promised functionality. Moreover, those tools that do appear to work, can be unpredictable when called upon. Thus, such evaluation can require significant trial and error and preparedness to tackle unexpected, subtle issues.

We want more for reverse engineering of sequence diagrams, in particular the ability to restrict the sequence diagrams to specific classes. For this purpose, we might investigate the code of Diver, as well as continue searching for more tools. Ultimately, we envision a set of tools connected through Eclipse, allowing students to learn, measure, and develop abstractions, so that the software not only remains useful but can better evolve with new requirements.

Acknowledgments

This work was supported in part by the National Science Foundation under grants DUE-1044061 and CNS-0643971. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the National Science Foundation.

References

[1] K. Alfert, J. Pleumann, and J. Schroder. Software engineering education needs adequate modeling tools. In *Proc. of 17th Conf. on Software Engineering Education and Training*, pages 72–77, Mar. 2004.

[2] `http://amateras.sourceforge.jp/cgi-bin/fswiki_en/wiki.cgi?page=AmaterasUML`.

[3] http://argouml.tigris.org/.

[4] http://www.eclipse.org/atl/.

[5] B. Berliner and N. B. Ruparelia. Early days of CVS. *SIGSOFT Softw. Eng. Notes*, 35:5–6, Oct. 2010.

[6] http://www.bouml.fr/.

[7] L. de Marcos, F. Flores, and J.-J. Martínez. Modeling with Plato: The Unified Modeling Language in a cultural context. In *Proc. of the 15th Annual Conf. on Innovation and Technology in Computer Science Education*, pages 249–253, 2010.

[8] http://marketplace.eclipse.org/content/diver-dynamic-interactive-views-reverse-engineering.

[9] J. Feller, B. Fitzgerald, S. A. Hissam, and K. R. Lakhani, editors. *Perspectives on Free and Open Software*. Cambridge, MA: The MIT Press, 2005.

[10] http://www.forrester.com/rb/Research/building_it_strategy_for_it_tooling/q/id/59686/t/2.

[11] http://www2.cs.uni-paderborn.de/cs/ag-schaefer/Lehre/PG/FUJABA/.

[12] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns Elements of Reusable Object-Oriented Software*. Addison Wesley, 1995.

[13] C. Ghezzi, M. Jazayeri, and D. Mandrioli. *Fundamentals of Software Engineering*. Prentice Hall, 1991.

[14] http://git-scm.com/.

[15] http://green.sourceforge.net/.

[16] R. Kerbs. Student teamwork: A capstone course in game programming. In *Proc. of 37th Annual Frontiers In Education Conference*, pages F4D–15 –F4D–19, Oct. 2007.

[17] E. C. Kubie. Recollections of the first software company. *IEEE Annals of the History of Computing*, 16(2):65–71, 1994.

[18] T. C. Lethbridge and J. Singer. Understanding software maintenance tools: Some empirical research. In *Proc. of Workshop on Empirical Studiess of Software Maintenance*, pages 157–162, 1997.

[19] A. Meneely and L. Williams. On preparing students for distributed software development with a synchronous, collaborative development platform. *SIGCSE Bull.*, 41:529–533, Mar. 2009.

[20] http://judebert.com/progress/archives/358-Rating-UML-Editors-Open-ModelSphere.html.

[21] http://www.papyrusuml.org/scripts/home/publigen/content/templates/show.asp?L=EN&P=55&vTicker=alleza&ITEMID=3. http://www.eclipse.org/modeling/mdt/papyrus/.

[22] http://assassin.cs.rpi.edu/~hollingd/sdd/.

[23] Software Engineering Facts and Figures. http://www.bsa.org/country/Public%20Policy/~/media/Files/Policy/Security/General/sw_factsfigures.ashx, May 2010.

[24] J. Singer, T. Lethbridge, N. Vinson, and N. Anquetil. An examination of software engineering work practices. In *Proc. of CASCON First Decade High Impact Papers*, pages 174–188, 2010.

[25] http://www.stanford.edu/class/cs295/.

[26] http://staruml.sourceforge.net/en/.

[27] http://subversion.apache.org/.

[28] http://www.uml.org/Visual_Modeling.pdf.

[29] http://homepages.inf.ed.ac.uk/perdita/Book/.

[30] http://uml.sourceforge.net/.

[31] http://wiki.eclipse.org/MDT.

[32] http://sourceforge.net/projects/violet/.

[33] www.uml.org/Visual_Modeling.pdf.

[34] X. Wu and C. Ge. The research on necessity and plan for using extreme programming in rational unified process. In *Proc. of Intl. Conf. on Computational Intelligence and Software Engineering*, pages 1–3, Dec. 2010.

[35] J.C. Yelmo and J. Fernandez-Corugedo. An experience of educational innovation for the collaborative learning in software engineering. In *Proc. of Global Engineering Education Conference*, pages 567–574, Apr. 2011.

Coordination Model to Support Visualization of Aspect-Oriented Programs

Álvaro F. d'Arce, Rogério E. Garcia, Ronaldo C. M. Correia, Danilo M. Eler
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
alvaro@darce.com, {rogerio,ronaldo,danilo}@fct.unesp.br

Abstract

Program Comprehension tasks represent a strategic role in Software Engineering activities, demanding time and effort, representing a considerable cost in maintenance. Aspect-Oriented Programming has specific elements to compose program behavior (e.g., aspects, pointcuts, join points and advices). The cognitive process can be improved by employing visual techniques to support source code analysis of its structure and behavior. A Software Visualization tool must be able to provide suitable visual representations for the program. Particularly, visual exploration of Aspect-Oriented programs requires some features to map how aspects crosscut some program structures. In this paper we propose a tool and an architecture to improve Aspect-Oriented program understanding supported by three coordinated visualization techniques.

Keywords: Program Understanding, Aspect-Oriented Program, Software Visualization, Software Engineering.

1 Introduction

Program Comprehension involves individual programmers understanding about what a program does and how it does it, in order to make functional changes and extensions, for example, without introducing defects [6]. Karahasanovic and his partners argue that Program Comprehension remains incomplete, requiring a deeper understanding of employed strategies, and before any modification (e.g., maintenance tasks), understanding the underlying mechanisms might improve the comprehension [13]. Acquiring knowledge about large programs may delay Program Comprehension tasks, which motivate the use of Software Visualization as an alternative approach to improve the cognitive process.

However, Aspect-Oriented programs comprehension can become more complex to be done, because separated units of code interfere in the behavior of other units. Aspect-Oriented Programming (*AOP*) supports crosscut concerns modularization, by structures that add behavior to selected elements of the programming language semantics [14]. Thus, *AOP* isolates implementation that otherwise would be spread and tangled throughout the base code. This new feature increases the program structural complexity, making harder to achieve the understanding of its architecture and structure, bringing up challenges to Aspect-Oriented Program Comprehension.

Based on that, Software Visualization can be an alternative approach to help software engineers to cope with structural complexity – due to fragmentation and the need to compose fragments – since the visual representations are able to work with an Aspect-Oriented program effectively, having specific visual mappings for Aspect-Oriented programs and mechanisms to gather and to organize data representing *AOP* features.

Visualization techniques, some extensions and tools have been proposed to support software visualization: *TreeMaps* [12, 2, 18], *Polymetric Views* [15, 3] to visualize hierarchical structures; *Hyperbolic Trees* [17] to visualize large hierarchical structures; *Bars and Stripes* for inter-relational structures visualization [1]; *UML 3D* to visualize packages, classes and methods [10]; and *Dependence Graphs* for inter-dependency level visualization [19]. These visualization techniques can be used to build visual representations to support comprehension tasks, but when applied to an Aspect-Oriented program as artifact, they are able to represent only the *aspects* before the *weaving* process – its fragmented code – and not the resulting code after the *weaver* – its tangled and spread code.

In this paper we propose a tool and an architecture supported by coordinated visual mappings which tackle Aspect-Oriented programs. The proposed approach is capable of analyzing a source code after the *weaving* process, enabling a visual exploration of the software structure (i.e., methods, *aspects*, *advices* and *pointcuts*) and how it all is related. Furthermore, our tool is able to visually show all structures involved in structural tests' results.

The remainder of this paper is organized as follow. Section 2 presents some considerations about Aspect-Oriented programs and also some related works. Section 3 presents some considerations about the developed tool to apply the proposed visual mapping and its architecture. Section 4 presents some applications of the proposed approach. Finally, the final remarks and future works are presented in Conclusions section.

2 Background

Kiczales et al. [14] argue that Object-Oriented Programming (*OOP*) is a technology that can aid Software Engineering, because the underlying object model provides a better fit to real domain problems. But they pointed out programming problems. For instance, *OOP* techniques are not sufficient to clearly capture important design decisions about different concerns implemented in a software system – some requirements (usually non-functional) cannot be clearly mapped to isolated units of implementation. Mechanisms to persist objects in relational data bases and security issues are examples of those concerns, usually named crosscutting concerns, because they tend to cut across multiple units of implementation [8].

Aspect-Oriented Programming (*AOP*) tackles the crosscutting concerns problem supporting the definition of implementation units (*aspects*) that cut across the system units (base code), composing the expected behavior [14]. An *AOP* language must define elements in order to combine functional and non-functional code (*aspects* and base code). Such elements establish a model to define points in the base code where additional behavior may be defined (*pointcuts*); a mechanism to identify these points (*join points*); encapsulation units for these points specifications and behavior enhancements (*aspects*); and a process to combine *aspects* with the base code (*weaver*) [8].

Functional requirements (like business rules) are coded inside a component language (e.g., Java) and non-functional requirements (e.g., logging, persistence, connection and transaction routines) are coded inside *aspects*. The *weaver* process joins the component language code with the *aspects* code, resulting in a runnable program. The language employed as case study in this paper is the *AspectJ*, which is an extension of Java language to support *AOP*.

An Aspect-Oriented program was used for this paper to generate views. It was designed and coded using *AspectJ* [16] and represents an on-line music store simulation, in which a billing service observes song-play events to bill the customer. An *aspect* implementation of the *Observer* design pattern [11] was used to add billing and statistics. Additionally, *aspects* track plays and lyric-shows events were coded to charge the customer appropriately and to update a list of the most frequently played songs.

One may observe the statical relations among *aspects* and classes, but it is not possible to conclude how *Aspects* crosscut a base code. Their relationships are important elements to Program Comprehension and should be mapped into visual structures to improve the cognitive process, which motivated extending Software Visualization techniques to deal with such elements.

After weaving and compilation processes of an AspectJ program, specific structures of an Aspect-Oriented program are tangled and spread in the bytecode. However, compilers use marks about *aspects* and their *advices* that can be retrieved by analyzing their signatures. The *AspectJ* compiler stores its components' signatures with specific markups, making it possible to obtain and to organize data about a compiled *AspectJ* program to a visual exploration.

2.1 Related Works

In Aspect Programs some interesting program elements can be highlighted by a Software Visualization tool in certain visual scenarios, such as, *aspects*, *advices* and *pointcuts*. It allows viewing how an *aspect* crosscuts one or more program structures, how much a program structure is modified by one or more *aspects*, the resulting tangled and spread code after the *weaving* process.

A *TreeMap* extension was proposed to show advised relations of Aspect-Oriented programs [18], but it consists in an user-based mining task by searching regular expressions representing *aspects* and classes elements. Another tool, AspectJ Development Toolkit (an *Eclipse* toolkit for *AOP*), has the *AJDT Visualizer* [5] – a *Bars and Stripes* visual representation of classes and *aspects*, highlighting its affected lines by aspects – but no visualization about how an *aspect* produces specified behavior in a class or other *aspect*. SourceMiner [4] can be used to enhance concern modularization analysis, but the concerns (focusing in *AOP*) need to be manually mapped. Another visualization tool, named AspectMap [9], shows implicit invocations in the source code. It allows visualizing *join points* shadows where *aspects* are specified to execute, showing the *advice* type and specified precedence information for a given *join point*.

In the next section we describe the coordination model proposed, the software visualization tool that implements such a model and its architecture, and how the visual mapping is performed.

3 The Software Visualization Tool

In this paper we propose a software visualization tool, called $SoftViz_{OAH}$, which employs a coordinated multiple views (*CMV*) approach to visualize Aspect-Oriented programs. The *CMV* schema is depicted in Figure 1. In this *CMV* approach we employee three visual representations and one content list. The *Structural Presentation* aims to show how the source code is hierarchically organized in packages, classes, methods, *aspects* and *advices*. The *Inter-Units Presentation* aims to show *aspects* crosscutting between methods and *advices*. The *Intra-Method Presentation* aims to show the intra-method behavior after the *weaving* process. The *Methods/Advices List* aims to show the methods and *advices* of classes or *aspects* selected in visual exploration. These visualizations are coordinated – the coordination is represented by the arrows – to allow exploring distinct levels of view.

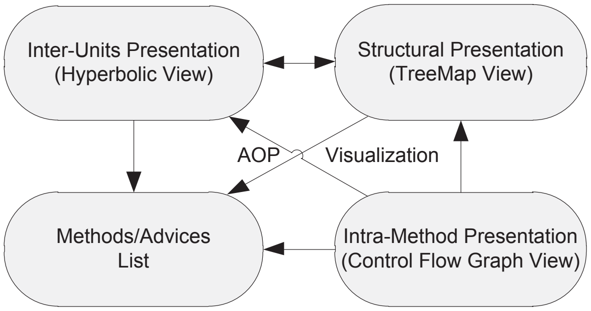


Figure 1. Visual mapping presentations schema

For the *Structural Presentation*, we have chosen the *TreeMap* technique (Figure 3(b)) to represent the program's hierarchical structure including *AOP* elements (*aspects* and advices). For the *Inter-Units Presentation*, we have chosen an *hyperbolic view* (Figure 3(a)) to represent classes' dependencies coupled with *aspects* crosscuts. For the *Intra-Method Presentation*, we have chosen a *Control Flow Graph* (*CFG*), Figure 4(b), to visualize the advised source code representing a piece of code behavior, showing *advices* over the code after the *weaving* process.

About the color mapping, each program element (i.e., the whole program, classes, methods, *aspects* and *advices*) has its own predefined color on visual mapping. All visual presentations use the same color mapping. When a test case is applied, a gradient from green to red (*Gradient Test Case*) is used to color the methods and *advices* represented or involved in the visual representation, according to their structural tests' results. The more the method or *advice* has failed in the test, the redder it is colored. The more the method or *advice* has succeeded, the greener it is colored. If the method or *advice* is not covered by the test case, it is colored using their predefined color. Such color mapping allows highlighting some *AOP* elements and viewing how succeeded (or failed) a test case covered the source code and how much of the whole source code is covered by a test case.

3.1 $SoftViz_{OAH}$ architecture

$SoftViz_{OAH}$ is a standalone desktop Software Visualization tool organized in three layers, *Dataset*, *Control* and *Visualization*, as illustrated in Figure 2. It uses program *bytecodes* as input and applies the proposed coordinated visual mapping to visualize *Java* and *AspectJ* programs structure and behavior as well as provides test case results visualization – test cases created with *JUnit*.

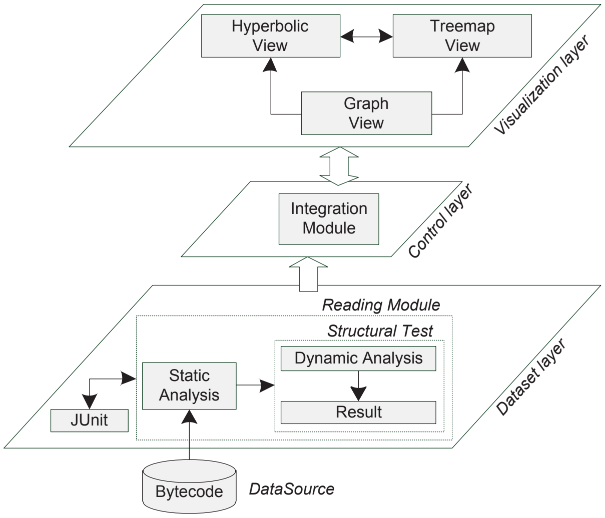


Figure 2. $SoftViz_{OAH}$ **architecture**

The *Dataset* layer performes static and dynamic analyses, creating necessary data sets to generate the visual representations. The *Reading Module* performes the *Static Analysis*, that reads a program's *bytecode* and analyzes its structure (packages, classes, methods, *aspects*, *advices* and test cases), execution sequence and its *aspects* crosscuts. During the *bytecode* reading, each unit data flow stream is analyzed and put into a Control Flow Graph data structure. An instrumentation technique is performed, inserting new instructions in each unit code containing calls from methods and *advices*, providing feedback to allow monitoring test cases by *Dynamic Analysis*). A list with all classes, *aspects* and their information obtained by the instrumentation is built. From this list, a hierarchical structure is built to be used to generate the *TreeMap* visual presentation and also a node table and an edge table, grouping informations like superclass references, variable types, methods' parameters and returns. By the interpretation of these tables and the organization of the links among the nodes, a *Dependence Graph* is constructed to be used to generate the *Hyperbolic* visual representation. When the *Structural Test* is in execution, the *Dynamic Analysis* makes an execution path by monitoring test cases and registers each visited portion of code of each code unit. The coverage criteria are then verified from this path to determine the test's *Result*, in which each criterion has its own standard to verify whether the test has been completely or partially met. The instrumentation sends to the tool, during a test execution, information about each visited portion of code. This information is stored to be used in each visual representation. By these two analysis steps (static and dynamic analyses), the tool gathers data to generate visual representations [7].

At *Control* layer, the data obtained from *Reading Module* (to generate visual presentations – all presentations are generated at the same time) are organized by the *Integration Module*, which provides mechanisms to coordinate the visualizations. These mechanisms capture user interactions in visual presentations to reflect the related events into another view. From each captured event, caused by an user interaction in a specific visualization, the *Integration Module* obtains data about the selected program unit (class, method, *aspect* or *advice*). An event is then sent to the Visualization layer, informing the new visual representation state (i.e., highlighted items), performing, this way, the coordination, using the data structures stored in the *Dataset* layer (i.e., the list containing information provided by the instrumentation, the hierarchical structure, the node and edge tables and the graph data structure).

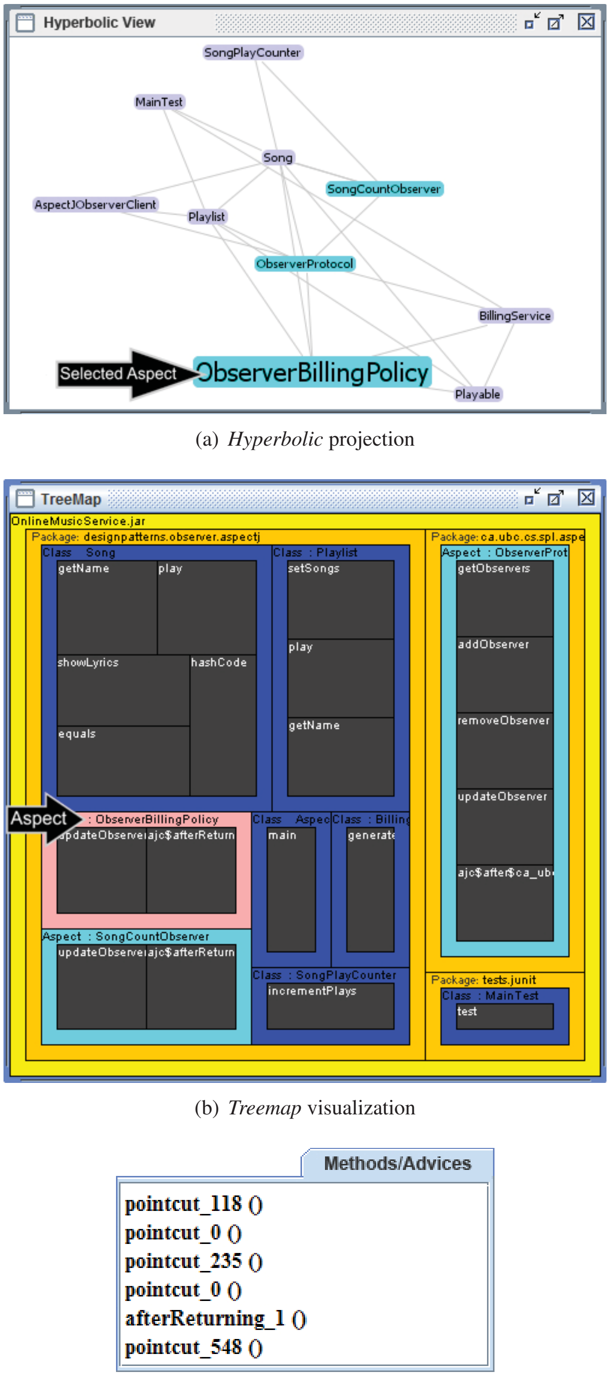
The *Visualization* layer is responsible for providing *CFG*, *TreeMap* and *Hyperbolic* visual presentations, highlighting *AOP* features (i.e., *aspects* and its crosscuts) depending on the visualization in question. This layer receives data from the *Control* layer (i.e., events captured from user interactions and data obtained from the *Dataset* layer) to generate and update each visual representation.

From the working together of these three layers, the developed Software Visualization tool uses the proposed coordinated visual mapping to visualize *Java* and *AspectJ* compiled programs. The next section presents an application and descriptions of the coordinated multiple views from $SoftViz_{OAH}$.

4 Applications

In this section we show an application of $SoftViz_{OAH}$: we performed a visual exploration of the Aspect-Oriented program of an on-line music store simulation, coded in *AspectJ* by Lesiecki [16].

Figure 4(b) shows a *Control Flow Graph* visual representation used to present methods, *advices* or test case executions. In this visual representation, basic code blocks are represented by rectangles outlined by a continuous line; code blocks advised by *aspects* (*advices*) are represented by rectangles outlined by a dashed line; the respective *pointcuts* are represented by the *advice* sequence in the graph; and method returns are represented by rectangles outlined by a double line. Inside each rectangle a number indicates the code block execution sequence. Normal code sequences are represented by continuous lines, and automatic generated code sequence (as exceptions) are represented by dashed lines. When a test case is applied, each rectangle is



(a) *Hyperbolic* projection

(b) *Treemap* visualization

pointcut_118 ()
pointcut_0 ()
pointcut_235 ()
pointcut_0 ()
afterReturning_1 ()
pointcut_548 ()

(c) List of Methods/Advices

Figure 3. Coordination from Hyperbolic to TreeMap view and Methods/Advices List

colored according to the *Gradient Test Case*, as shown in Figure 4. Otherwise, a predefined color is used. The global view of the *Control Flow Graph View* represents the tangled and spread code. In the example depicted in Figure 4(b), the code is tangled (*advices* among the code) and the *aspect* is thinly spread (*advices* are close to each other).

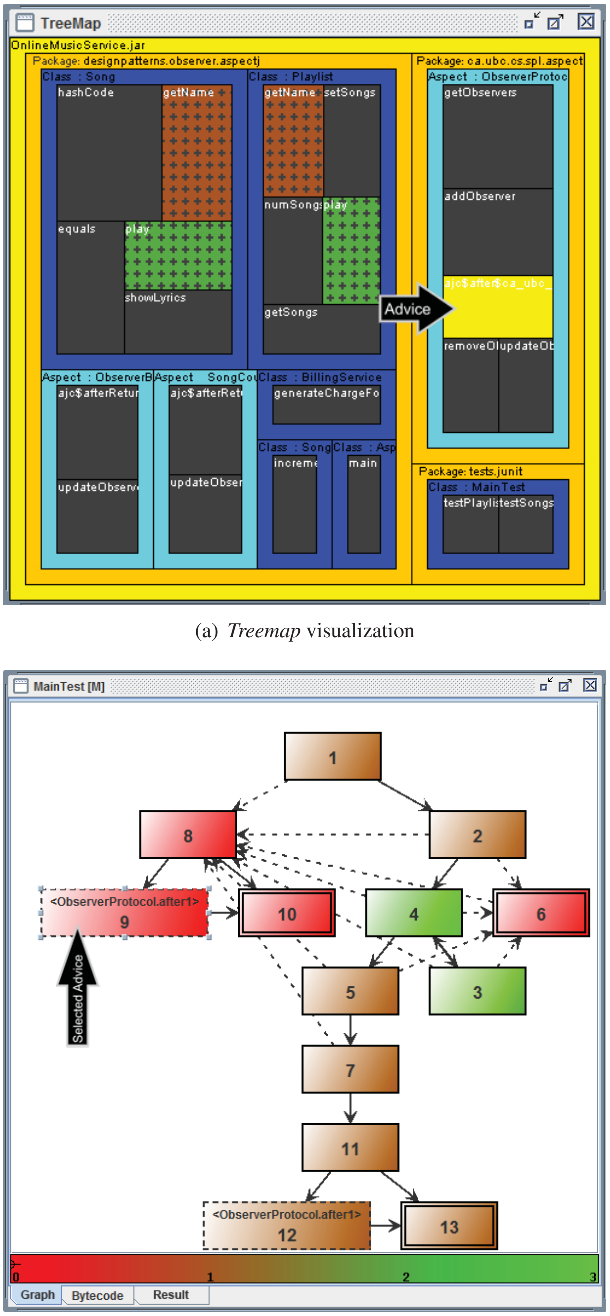
We employed the *TreeMap* visual representation to present hierarchical structures in nested rectangles, as shown in Figure 4(a). Rectangles represent program structures (e.g., the whole program, packages, classes, methods, *aspects* and *advices*). The rectangle size represents the number of calls of a represented method or *advice* inside a specified context – the whole program, classes, methods, *aspects*, *advices* and test cases. When a test case is applied, the *Gradient Test Case* is used to color all program methods and *advices* in the test case scope, as illustrated in classes *Song* and *Playlist* shown in Figure 4. The *TreeMap* representation provides a global view of the whole program hierarchical structure with highlighted *aspects*.

In the *Hyperbolic* visual presentation, depicted in Figure 3(a), nodes represent program classes and *aspects*; and edges represent dependency between each class or *aspect*, i.e. method calls and *aspect* crosscuts. Classes and *aspects* are colored employing different colors. When a test case is applied, the *Gradient Test Case* is used to color the edges, considering the methods and *advices* belonging to the two classes or *aspects* connected by the edge in the test case scope. Otherwise, edges are colored using a predefined color. The global view of the *Hyperbolic View* represents the classes dependences and (highlighted) *aspects* crosscuts.

All three visual representations and the content list are coordinated: an user interaction event in which one reflects in others to represent the same state in all visualizations. The coordination schema is shown Figure 1. Figure 3 depicts the coordination from *Hyperbolic* to *TreeMap* view and *Methods/Advices List*. In the *Hyperbolic* view is selected the *aspect ObserverBillingPolicy*, which is highlighted in the *TreeMap* view. Additionally, the *advices* of the selected *aspect* are shown in *Methods/Advices List* 3(c). By this coordination, one may observe classes and *aspects* dependences, its hierarchical structures and the respective methods and *advices* involved in each class and *aspect* dependence (i.e. methods call and *advices* crosscuts).

In Figure 4 the *Control Flow Graph* shows an intra-method behavior after the *weaving* process. The block color is based on the test case performed. In this view is selected the *advice ObserverProtocl.after1*, which is highlighted in *TreeMap View*. By this coordination, one may observe the weaved code flow structure – code behavior – and the related methods and *advices* involved in each code sequence.

The software visualization tool $(SoftViz_{OAH})$ proposed in this paper is able to asses the visual mapping and improve the understanding of Aspect Oriented Programming. By interacting with those coordinated visual presentations, in hierarchical structure, classes' dependence, *aspects* crosscuts, fragmented, tangled and spread code, one may observe how *aspects* crosscut classes.



(a) *Treemap* visualization

(b) *Control Flow Graph* visualization

Figure 4. Coordination from TreeMap view to Control Flow Graph visual presentation

5 Final Remarks and Further Works

Most software artifacts are abstract, having no physical representation. Focusing on source code, obtaining insights can be difficult, especially because of the amount of lines in software systems and its structure. Our motivation came from analyzing the associated *AOP* source code, since the new *AOP* features and consequences – fragmented, tangled and spread code – must be externalized by a visual mapping. There are some tools that provide visual representations. Some of them support Aspect-Oriented programs, some have the necessity of manually mapping concerns, and some provide no *aspect* identification. But most of them do not provide a more broad aid to entire program understanding.

So, a coordinated visual mapping was proposed to present Aspect-Oriented programs' features using three visual presentations and a content list which aim to externalize structural organization, the relations among classes and *aspects*, and the advised code. The proposed coordination schema makes it possible to highlight selected elements on different detail levels, allowing the user to gather information about *aspect* and its spreading to the source code – how *aspects* crosscut the program structures, composing program behavior.

We also developed a tool ($SoftViz_{OAH}$) to assess the visual coordination model proposed. And both Object Oriented and *AOP* source code has been used on assessing the coordinated visual mapping. Also, $SoftViz_{OAH}$ is able to show test cases' results along with the visual representations, that allows analyzing how the test cases go through source code and their coverage. Hitherto, our evaluation has shown that such feature is helpful in locating defects (where the defect is).

Also according to preliminary evaluation, the tool's functionalities are useful to support Aspect-Oriented program understanding. However, to assess the effectiveness and efficiency of the visual proposed mapping in a more realistic manner, a controlled experiment has been planned, and will be conducted using an experimentation process, aimed to locate defects previously inserted into *AOP* source codes.

References

[1] J. Baldwin, D. Myers, M. Storey, and Y. Coady. Assembly code visualization and analysis: An old dog can learn new tricks! *PLATEAU '09 Workshop at Onward!*, 2009.

[2] B. B. Bederson, B. Shneiderman, and M. Wattenberg. Ordered and quantum treemaps: Making effective use of 2d space to display hierarchies. *ACM Transactions on Graphics (TOG)*, 4(21):833–854, October 2002.

[3] G. Carneiro, R. Magnavita, and M. Mendonça. Combining software visualization paradigms to support software comprehension activities. In *4th ACM symposium on Software visualization (SoftVis'08)*, pages 201–202, NY, USA, 2008.

[4] G. Carneiro, C. Sant'Anna, A. Garcia, C. Chavez, and M. Mendonça. On the use of software visualization to support concern modularization analysis. *ACoM 2009, Colocated with OOPSLA*, 2009.

[5] A. Clement, S. Colyer, and M. Kersten. Aspect-oriented programing with ajdt. *AAOS 2003: Analysis of Aspect-Oriented Software workshop at ECOOP 2003*, 2003.

[6] C. L. Corritore and S. Wiedenbeck. An exploratory study of program comprehension strategies of procedural and object-oriented programmers. *International Journal of Human-Computer Studies*, (54), 2001.

[7] A. F. D'Arce, R. E. Garcia, and R. C. M. Correia. Coordinated visualization of aspect oriented programs. *I Workshop Brasileiro de Visualização de Software (WBVS 2011), II Congresso Brasileiro de Software: Teoria e Prática (CBSoft 2011).*, 2011.

[8] T. Elrad, R. E. Filman, and A. Bader. Aspect-oriented programming – introduction. *Communications of the ACM*, 10(44):29–32, 2001.

[9] J. Fabry, A. Kellens, S. Denier, and S. Ducasse. Aspectmaps: A scalable visualization of join point shadows. *International Conference on Program Comprehension (ICPC)*, 2011.

[10] H. Gall and M. Lanza. Software analysis visualization. *28th International Conference on Software Engineering (ICSE Shangai 2006)*, 2006.

[11] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns*. Addison-Wesley, 2000.

[12] B. Johnson and B. Shneiderman. Tree-maps: a space-filling approach to the visualization of hierarchical information structures. In *Proceedings of the 2nd conference on Visualization '91*, VIS '91, pages 284–291, Los Alamitos, CA, USA, 1991. IEEE Computer Society Press.

[13] A. Karahasanovic', A. K. Levine, and R. Thomas. Comprehension strategies and diffculties in maintaining object-oriented systems: An explorative study. *Journal of System and Software*, 7(80):1541–1559, 2007.

[14] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. G. Griswold. An overview of aspectj. *ECOOP 2001 – Object Oriented Programming*, 2072(2001):327–354, 2001.

[15] M. Lanza. Codecrawler - polymetric views in action. In *19th International Conference on Automated Software Engineerings*, pages 394–395, september 2004.

[16] N. Lesiecki. Aop@work: Enhance design patterns with aspectj. *IBM developerWorks*, 2005. Accessed on Nov/2011.

[17] T. Munzner. Exploring large graphs in 3d hyperbolic space. *IEEE Computer Graphics and Applications*, 18(4):18–23, July/August 1998.

[18] J. Pfeifer and J. Gurd. Visualization-based tool support for the development of aspect-oriented programs. *Proceedings of 5th International Conference on Aspect-Oriented Software Development*, 2006.

[19] T. Würthinger, C. Wimmer, and H. Mössenböck. Visualization of program dependence graphs. *Compiler Construction, 17th International Conference, Springer, vol. 4959, Lecture Notes in Computer Science*, pages 193–196, 2008.

Improving Program Comprehension in Operating System Kernels with Execution Trace Information

Elder Vicente[1], Geycy Dyany[1], Rivalino Matias Jr., Marcelo de Almeida Maia
Faculty of Computing
Federal University of Uberlândia
Uberlândia, MG - Brasil
marcmaia@facom.ufu.br

Abstract—**Operating systems are one of the most complex kinds of software systems ever built. Their complexity results from many factors, in special, the huge size and low-level issues. As consequence, there are many programming challenges to be considered at either the *in-the-large* level or *in-the-small* level. Program comprehension is a crucial problem for developers who need to contribute or reuse the code base in some way. One of the most important challenges in this scenario is to find the functions in the source code that are responsible for a specific feature of the system. Previous work has shown the benefits of using execution trace information to improve the comprehension process of features that can have their execution adequately reproduced. However, the use of execution traces in the comprehension of operating system kernel features is poorly studied. In this work, we show that execution traces can effectively improve program comprehension of kernel features when adequate filters are provided to the instrumentation tools.**

Keywords – program understanding, execution traces, operating systems

I. INTRODUCTION

Linux is a robust operating system that can be used either in desktops or in corporate servers of large companies, supporting several plataforms (ARM, x86, MIPS, SPARC, etc). Statistics presented in [1], [2] show that the kernel source code is under active development, and consequently the evolution of the base code comes together with this development: a new release is created each 60-80 days. Developers worldwide participate in the development of the kernel. For instance, the release 2.6.35 accounted the collaboration of approximately 1145 developers, including the participation of large companies, such as, RedHat, Novell, IBM and others. Moreover, the number of lines of code also grows with the evolution of the kernel. For instance, release 2.6.10 had almost 6 million lines of code and release 2.6.27 had more than 8.5 million lines of code. In this scenario, the task of feature location [3], i.e., the task of finding source code pieces related to user point-of-view software features, is an extremely challenging task because features may not be intuitively located in the code, and even if that was the case, there are so many number of modules and functions to be navigated that the task is still challenging. This situation is even more dramatic for newbie developers working on the kernel.

In order to alleviate this challenge, the source is distributed together with a folder that contains the kernel documentation. There are still some other sources of information that explains the kernel source code [6], **Erro! Fonte de referência não encontrada.**. However, the available documentation is still not sufficient to reference and explain all source code files or all functions that are used to implement the feature of interest. In this way, the documentation only offers a general view of a feature, and not necessarily all features that a developer may be interested are described in the documentation. Moreover, some books had already published on the kernel [7], [9], but still there is no guarantee that the text included in the book will remain updated, considering the kernel evolution throughout the time.

Fortunately, many solutions to the program comprehension problems had already been proposed. The solutions are based on a variety of techniques: static techniques which usually construct graphs from the source code using compiling techniques and perform some kind of query or browsing on that graph; dynamic techniques which usually extract information from the execution of the desired feature to drive the location of the piece of interest; information retrieval techniques which consider the terms written by the programmers to associate with feature terms and enable queries by similarity; and hybrid techniques, which seems to be the most successful ones, that combine the above techniques to enhance the precision and recall of the returned information.

However, program comprehension using dynamic information has not been studied with operating system kernels as the target software [3], [4]. One possible reason is that, if the analysis of dynamic information from the execution still imposes important challenges for systems implemented in high-level languages, such as Java, where the events captured during the execution are typically function entry and exit, consider the situation where important events to be considered in operating system execution traces could be associated with low-level interfaces, such as, interrupt handling. Moreover, the number of different kinds of events to be considered by the instrumentation tool could also be an issue.

Despite this negative scenario for using information of execution traces in the problem of program comprehension for operating systems kernel, we raise the hypothesis that

[1] This work was partially supported by the Brazilian agency CAPES.

developing appropriated filters into the instrumentation tool may provide useful and feasible information for developers.

The goal of this work is to develop a method for producing filters for low-level events that could accurately extract dynamic information of instrumented programs, without incurring in the problem of flooding the developer with unnecessary trace information. The more filters are precise, the more developer's work is simplified. The suitable filters should enable the use of execution trace information for program comprehension problems in operating systems, especially those related to locating pieces of code related to some specific functionality.

In the next section, the tools used in this study will be reviewed. In Section III, we present the study setting. In Section IV the results of the study are presented and in Section V these results are discussed. Finally, Section VI presents our final remarks.

II. Background Tools

A. Ftrace

The *ftrace* is a tool used to trace kernel functions [12], [13]. It is a simple tracer of *kprobes*, but faster and more precise in terms of time analysis [11]. *Ftrace* does not require any specific application in the user-space and all configurations steps are carried out using the *debugfs* file system. *Ftrace* can be used in latency analysis or in debugging performance problems. The *ftrace* infrastructure allows other types of tracing, such as, *irqsoff* plugin, which traces source code areas that disable interrupts. This plugin allows to measure the interval times in which interrupts are disabled in the kernel. *Ftrace* was used in this work to collect traces of the subject program proposed in the experiment.

B. SystemTap

SystemTap provides a scripting environment to analyze the tasks of a Linux system at runtime [10]. The dynamic information can be collected at real-time, which is a flexible way to trace the kernel execution. SystemTap provides mechanisms to extract, filter and summarize in order to simplify the analysis and diagnosis of the desired properties. One positive point is that SystemTap does not require to recompile, to reinstall and to reinitialize the kernel in order to obtain the target data [11].

SystemTap uses an internal scripting language similar to the AWK programming language. An internal analyzer checks the script for syntactic errors and converts it to a C program, which is loaded as a kernel module, as shown in Figure 1. SystemTap also allows creating modules for other versions of the kernel, other than the version of its respective running kernel. It is possible to copy the module to the target system and execute it with the program *staprun*.

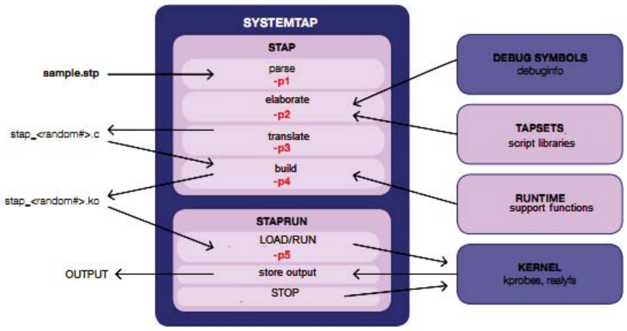


Figure 1. SystemTap operation.

The primary use of SystemTap in this work is to trace Linux kernel events occurring during a pre-specified period of time. SystemTap will be used to find the mean time used by the clock tick handler and also to find how many clock ticks had occurred during the execution of a subject program in the proposed experiment to validate the quality of the experiment result.

C. Hardware interface

In this work, the target architecture will be based on SMP (*Symmetric multiprocessing*) used in several models of Intel x86 processors. In this architecture, the APIC system (*Advanced Programmable Interrupt Controller*) is responsible for generate and manage several kinds of hardware interrupts [15]. Figure 2 shows a typical APIC system, basically consisting of an I/O APIC module responsible to receive interrupt requests from I/O devices (keyboard, network, disks, etc.) and forward them to the Local APIC module (LAPIC), which is integrated into the processor. Each LAPIC contains a set of APIC registers and associated hardware that control the feeding of interrupts to the processor core.

The main source of interrupts are: 1) I/O devices locally connected, i.e., interrupts generated by I/O devices connected directedly to the connectors of local interrupts of the processor (LINT0 and LINT1); 2) I/O devices connected externally, i.e., interrupts generated by I/O devices connected to the module I/O APIC; 3) inter-processor interrupt (IPIs), i.e., a processor can use the IPI mechanism to interrupt other processor or a group of processors; 4) interrupts generated by the timer APIC, i.e., the local APIC timer that can be programmed to send periodical interrupts i.e., *clock ticks*; 5) interrupts generated by the temperature sensor; 6) internal error APIC interrupt signaled when an internal error of the APIC is detected.

In this work, the studied problem will require disabling and enabling timer interrupts (4), to avoid that tasks do not be periodically interrupted.

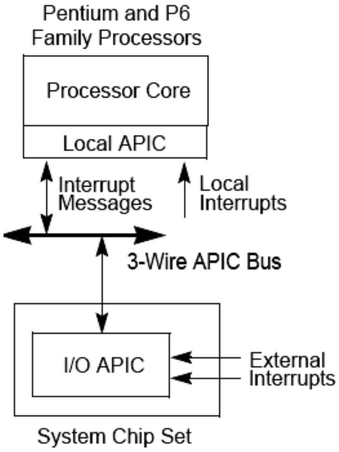


Figure 2. LAPIC and I/O APIC schema.

III. THE STUDY SETTING

In this section, we will present the experiment designed to evaluate the feasibility of using execution traces for program comprehension of operating system kernels.

The experiment design was defined using several components:

1. A problem that required some program comprehension activity in the target operating system kernel;
2. The solution/contribution to the problem cited in the previous item;
3. The method used to support and evaluate the construction of the solution cited in the previous item.

The problem used in the design of the experiment of this study was the *OS Jitter* that is one of the main factors that can introduce delays when processing applications in high performance computing environments – HPC. *OS Jitter* can be considered any interference that an application process or thread experiences because of the execution of internal tasks the operating system. In this study, the Linux kernel was chosen because of the availability of its source code and because it is being used in 91.4% of top 500 supercomputer systems [14]. Among the several types of interferences of the operating system during user application execution, especially for HPC applications, we can highlight: the execution of administrative process (*daemons*) and periodical kernel routines, such as, the clock tick.

In this study, the chosen contribution is to understand where *OS jitter* caused by clock ticks is implemented in the kernel. The proposed solution to reduce OS jitter is based on *tickless* processors, where the CPU is assigned to user application and it will not be interrupted by periodic kernel tasks (*clock ticks*). Currently, the Linux kernel already supports a similar approach –*tickless kernel* – to reduce energy consumption. So, the maintenance task is to reuse the current implementation of tickless kernel of Linux to introduce the feature of executing CPU-bound process without clock interrupts. In this study, the user application that will be executed as a CPU-bound process is a matrix multiplication program.

The method used to support and evaluate the quality of the comprehension process of where clock ticks are handled in the Linux kernel consists of: the general strategy to filter execution traces to find the desired feature in the source code and the validation of the filtering process.

A. Trace Filtering

The filtering strategy needs to isolate the execution of user application (matrix multiplication) to only one CPU, in such way that the events captured in traces are only related to the respective user process. The user program must be instrumented in the following way:

1. Configure the system to run the process only in CPU 1 (*sched_setaffinity* system call).
2. Turn off the trace capture of *ftrace*; configure *ftrace* to capture the trace of CPU 1 only; cleans the log file of *ftrace*.
3. Configure the system to run all other tasks, threads, daemons in CPU 0.
4. Move all timers – scheduled tasks (*workqueues, callouts, etc.*) from CPU 1 to CPU 0.
5. Move all interrupts (network, keyboard, mouse, etc) to CPU 0, except timer interrupt.
6. Activate *ftrace* trace collector.
7. Capture the initial time (*clock_gettime*)
8. Allocate dynamically a matrix 1024x1024.
9. Perform the core application (*matrix multiplication*)
10. Capture the final time (*clock_gettime*).
11. Deactivate *ftrace* trace collector.
12. Get the time used in the user application.

This strategy should guarantee that the trace collected by *ftrace* contains events (function calls) related to the user application and to the timer interrupt handler.

It is expected that function(s) related to timer interrupt handler should be recognized in a reasonable way using the generated trace.

B. Validation of the result of the filter output

In order to validate the quality of the generated trace, the instrumented user program described previously in the Trace Filtering subsection was modified to include a system call between steps 5 and 6 to disable clock tick handling.

Figure 3 shows the system call that should be used with parameter 1 to disable clock ticks. Figures 4 and 5 contain the code to effectively perform the disabling process.

```
int sys_confapic(struct pt_regs *regs) { // ID=337
  int pparameter = regs->bx;
  if (pparameter == 1)
    lapic_suspend(NULL);
  if (pparameter == 2)
    lapic_resume(NULL);
  }
```

Figure 3. Function *sys_confapic*

```
static int lapic_suspend(struct sys_device *dev)  {
  unsigned long flags;
  int maxlvt;
  if (!apic_pm_state.active)
    return 0;
  maxlvt = lapic_get_maxlvt();
  apic_pm_state.apic_id = apic_read(APIC_ID);
  apic_pm_state.apic_taskpri = apic_read(APIC_TASKPRI);
  apic_pm_state.apic_ldr = apic_read(APIC_LDR);
  apic_pm_state.apic_dfr = apic_read(APIC_DFR);
  apic_pm_state.apic_spiv = apic_read(APIC_SPIV);
  apic_pm_state.apic_lvtt = apic_read(APIC_LVTT);
  if (maxlvt >= 4)
    apic_pm_state.apic_lvtpc = apic_read(APIC_LVTPC);
  apic_pm_state.apic_lvt0 = apic_read(APIC_LVT0);
  apic_pm_state.apic_lvt1 = apic_read(APIC_LVT1);
  apic_pm_state.apic_lvterr = apic_read(APIC_LVTERR);
  apic_pm_state.apic_tmict = apic_read(APIC_TMICT);
  apic_pm_state.apic_tdcr = apic_read(APIC_TDCR);
  #ifdef CONFIG_X86_THERMAL_VECTOR
    if (maxlvt >= 5)
      apic_pm_state.apic_thmr = apic_read(APIC_LVTTHMR);
  #endif
  local_irq_save(flags);
  disable_local_APIC();
  if (intr_remapping_enabled)
    disable_intr_remapping();
  local_irq_restore(flags);
  return 0;  }
```

Figure 4.   Lapic suspend

```
void disable_local_APIC(void) {
  unsigned int value;
  if (!x2apic_mode && !apic_phys)
    return;
  clear_local_APIC();
  value = apic_read(APIC_SPIV);
  value &= ~APIC_SPIV_APIC_ENABLED;
  apic_write(APIC_SPIV, value);
  #ifdef CONFIG_X86_32
    if (enabled_via_apicbase) {
      unsigned int l, h;
      rdmsr(MSR_IA32_APICBASE, l, h);
      l &= ~MSR_IA32_APICBASE_ENABLE;
     wrmsr(MSR_IA32_APICBASE, l, h); }
  #endif }
```

Figure 5.   Disable Local APIC

It is also necessary to include a system call between steps 10 and 11 to enable clock interrupts again on the chosen CPU; otherwise it would become inaccessible to other processes after the execution of the experiment. Figure 6 shows the necessary code to re-enable clock ticks.

```
static int lapic_resume(struct sys_device *dev) {
  unsigned int l, h;
  unsigned long flags;
  int maxlvt;
  int ret = 0;
  struct IO_APIC_route_entry **ioapic_entries = NULL;
  if (!apic_pm_state.active)
    return 0;
  local_irq_save(flags);
  if (intr_remapping_enabled) {
    ioapic_entries = alloc_ioapic_entries();
    if (!ioapic_entries) {
      WARN(1, "Alloc ioapiclapic resume failed.");
      ret = -ENOMEM;
     goto restore; }
    ret = save_IO_APIC_setup(ioapic_entries);
    if (ret) {
      WARN(1, "Saving IO-APIC state failed: %d\n", ret);
      free_ioapic_entries(ioapic_entries);
      goto restore; }
    mask_IO_APIC_setup(ioapic_entries);
    mask_8259A();
  }
  if (x2apic_mode)
    enable_x2apic();
  else {
    rdmsr(MSR_IA32_APICBASE, l, h);
    l &= ~MSR_IA32_APICBASE_BASE;
    l |= MSR_IA32_APICBASE_ENABLE | mp_lapic_addr;
    wrmsr(MSR_IA32_APICBASE, l, h);  }
  maxlvt = lapic_get_maxlvt();
  apic_write(APIC_ID, apic_pm_state.apic_id);
  apic_write(APIC_DFR, apic_pm_state.apic_dfr);
  apic_write(APIC_LDR, apic_pm_state.apic_ldr);
...
  apic_write(APIC_TDCR, apic_pm_state.apic_tdcr);
  apic_write(APIC_TMICT, apic_pm_state.apic_tmict);
  apic_write(APIC_ESR, 0);
  apic_read(APIC_ESR);
  apic_write(APIC_LVTERR, apic_pm_state.apic_lvterr);
  apic_write(APIC_ESR, 0);
  apic_read(APIC_ESR);
  if (intr_remapping_enabled) {
    reenable_intr_remapping(x2apic_mode);
    unmask_8259A();
    restore_IO_APIC_setup(ioapic_entries);
    free_ioapic_entries(ioapic_entries); }

restore:
  local_irq_restore(flags);
  return ret;
}
```

Figure 6.   Fragment of *lapic_resume*

The resulted trace after running the instrumented system with clock tick disabling should be compared with the trace where clock ticks were not disabled, in order to see if the filtering process that selected the considered function(s) to implement clock tick handling was correct. In order to confirm the correction, that function(s) should not be present in the new trace.

Another adopted validation step was to verifying the number of calls to the function(s) that are supposed to be the one(s) that handle clock ticks using the SystemTap script shown in Figure 7. A complementary SystemTap script shown in Figure 8 calculates the duration of the execution of these respective functions that can indicate some measure of the OS jitter.

```
global var1
probe begin {
   var1[0]=0;  var1[1]=0;   var1[2]=0;  var1[3]=0;   }

probe kernel.function("<name of the supposed function>").call {
    j=cpu();
    var1[j]=var1[j]+1;
}

probe end {
   printf("ID: %d  Count: %d\n",0, var1[0]);
   printf("ID: %d  Count: %d\n",1, var1[1]);
   printf("ID: %d  Count: %d\n",2, var1[2]);
   printf("ID: %d  Count: %d\n",3, var1[3]);
}
```

Figure 7.   Script to verify the number of clock tick handling

```
global i
global j
global var1

probe begin {  i=0; }

probe kernel.function("<name of the supposed function>").call {
  if (cpu()== $1) { var1[$1]=gettimeofday_ns() }
}

probe kernel.function("<name of the supposed function>").return
{
  if (cpu()== $1) {
    j=gettimeofday_ns()
    printf("%d\n",j - var1[$1])
    i++
    if (i==310) exit()
  } }
```

Figure 8. Script to analyze the duration of clock tick handling

IV. Results

This section present the results obtained after running the method proposed in the previous section.

A. Trace Filtering Result

The result of *trace filtering* is shown in Figure 9. The trace consists of function calls with their respective nesting, which can be considered as a call tree. Approximately, 90% of the trace consists of calls to *smp_apic_timer_interrupt* and their corresponding nested calls. A straightforward analysis of this particular function has shown that this is the function that should be responsible for handling clock ticks.

```
….
smp_apic_timer_interrupt() {
   native_apic_mem_write();
   irq_enter() {
     rcu_irq_enter();
     idle_cpu();
   }
   hrtimer_interrupt() {
     ktime_get();
     _spin_lock();
     __run_hrtimer() {
       __remove_hrtimer();
       tick_sched_timer() {
         ktime_get();
         tick_do_update_jiffies64() {
           _spin_lock();
           do_timer() {
             update_wall_time() {
               update_xtime_cache();
             }
             calc_global_load();
           }
         }
….
```

Figure 9. Trace fragment with clock tick handling

B. Validation Results

The execution of the instrumented user program disabling clock tick handling before the core program (*matrix multiplication*) produced the trace shown in Figure 9. This trace is much smaller than the one shown in Figure 8. It also could be observed that there was no call to *smp_apic_timer_interrupt.*

```
…
sys_clock_gettime() {
  posix_ktime_get_ts() {
    ktime_get_ts() {
      set_normalized_timespec();
    }
  }
  copy_to_user() {
    __copy_to_user_ll();
  }
}

sys_clock_gettime() {
  posix_ktime_get_ts() {
    ktime_get_ts() {
      set_normalized_timespec();
    }
  }
  copy_to_user() {
    __copy_to_user_ll();
  }
}
…
```

Figure 10. Trace fragment without clock ticks

Using the scripts shown in Figure 7 and 8 in the instrumented program that handle clock ticks, we encountered that the time to handle each clock tick is approximately 5652 nanoseconds (mean time from 310 replications) and the number of clock ticks that occurs during a 5 minutes interval is approximately 32234 (mean number from 35 replications).

Using the scripts shown in Figure 7 in the instrumented program that **do not** handle clock ticks, we encountered that still 4 clock ticks were handled. Indeed, 4 is a very small number compared to 32234 that would not invalidate the interpretation that the *smp_apic_timer_interrup* is indeed the function responsible for clock tick handling. These 4 clock tick handling occurrences may have occurred during the time necessary to initiate the script and the user program execution which are not exactly simultaneous.

V. Discussion

Other approaches to analyze execution traces have been extensively used in the program comprehension [4, 16]. However, there is no study whose target system is an operating system kernel. The seminal work in the application of execution trace to program comprehension is the Software Reconnaissance approach that also compares code executed in traces with and without the features [5]. This is an interesting approach but depending on the size of trace, the excess of information may hinder the approach with useless information. In [17], the authors proposed an enhancement in the trace differentiation providing more contextualized differentiation with trace alignment algorithm and also had to propose a trace summarization algorithm in order to dramatically reduce the size of trace, and consequently produce a feasible approach. In this work, we had the same challenge of reducing effectively the size of trace in order to grasp adequately the trace file to extract the desired information of clock tick handling. We proved that our approach was effective because the desired function was readily found. It is important to note that if a newbie kernel developer had to look for this function in a traditional way, browsing the source code from the *main* function until he could find the method, this would be a very hard task that could take several days or even weeks of work.

In [18], another approach to summarize traces was designed using fan-in and fan-out metrics, which is completely different from the specialized filtering approach used in this work. In their approach, the functions are ranked considering their relative importance based on graph metrics which do not consider any semantic information of the respective function. The approach of program comprehension presented in this work guides the developer in a much more precise manner, because it is the developer itself that have the tacit knowledge of what he really needs and thus, he writes the proper filters considering this knowledge. Consequently, this strategy improved dramatically the quality of the information available in the trace. However, it is important to note that our approach requires a more specialized preparation of the trace filtering. Although, the developer can reuse the generic parts of our filtering framework, he stills needs to grasp a filtering solution that will provide a precise "fishing" of the desired functions.

Some authors [19, 20] suggest the integrated use of static and dynamic views of the software system in program comprehension activities. The dynamic views are obtained by profiling the most used features in the system. Nonetheless, the primary goal is to obtain the system architecture to reduce the effort of comprehension of the system. In our approach, we have not focused on the most used features for comprehending the system in an overall manner. We have focused on a well chosen feature in order to provide direct information customized for the process of program comprehension. This requires less effort to make further needed changes in the source code. Nonetheless, we could still improve our process of writing adequate filters and even of browsing the resulted traces with static information. For instance, the use of information retrieval techniques could also be incorporated [21, 22, 23], because we could see that the function names used in kernel are very representative and similar to the developer terms in high-level communication.

VI. Final Remarks

This paper has presented an innovative study in the program comprehension area because it used execution trace information to isolate desired functions in the Linux operating system kernel. Previous studies in this area did not handled OS kernels.

Our findings have shown that despite the huge amount of events that an instrumentation tool can generate, especially in the case of OS kernel which has to cope with much more kinds of different low-level events than a high-level application, the use of proper filtering mechanisms in the instrumentation tool can reduce dramatically the complexity of the execution trace information. This scenario reduces the developer effort during maintenance tasks when he needs to find specific functions in the source code. Future work includes reproducing the subjacent methodology designed in this work in a large scale experiment to produce stronger evidences on our findings. Also, introducing hybrid static techniques in a cohesive methodology is an important step.

References

[1] M. Florian. *Linux Kernel Statistics*. [Online]. Available at: http://www.schoenitzer.de/lks/lks_en.html. Accessed in 2011, December.

[2] *Kernel development statistics for 2.6.35*. [Online]. Available at: http://lwn.net/Articles/395961/. Accessed in 2011, December.

[3] B. Dit, M. Revelle, M. Gethers, D. Poshyvanyk. "Feature location in source code: a taxonomy and survey". Journal of Software Maintenance and Evolution: Research and Practice. In press. Published online. 2011.

[4] B. Cornelissen, A. Zaidman, A. van Deursen, L. Moonen, R. Koschke, "A Systematic Survey of Program Comprehension Through Dynamic Analysis," *in IEEE Transactions on Software Engineering,*Vol.35, 2009, pp. 684–702.

[5] N. Wilde, M. Scully "Software Reconnaissance: mapping program features to code". Software Maintenance: Research and Practice, vol. 7, n. 1, pp. 49-62, 1995.

[6] *The Linux Documentation Project*. [Online]. Disponível: http://tldp.org/. Accessed in 2011, December.

[7] I. T. Bowman, R. Holt, N. Brewster: Linux as a Case Study: Its Extracted Software Architecture. In Proc. of ICSE, pp. 555-563, 1999.

[8] D. P. Bovet and M. Cesati, *"Understanding The Linux Kernel,"* 3th ed. O'Reilly, 2005.

[9] R. Love, "*Linux Kernel Development Second Edition*", Pearson Education, 2th ed. 2005.

[10] SystemTap. [Online]. Available at: http://sourceware.org/systemtap/. Accessed in 2011, December.

[11] R. Matias, I. Becker, B. Leitão, P.R. Maciel "Measuring Software Aging Effects Through OS Kernel Instrumentation," in IEEE Second International Workshop on Software Aging and Rejuvenation, San Jose, CA, 2010, pp. 1-6.

[12] S. Rostedt, "Finding Origins of Latencies Using Ftrace," *in Proc. of the11th Real Time Linux Workshop*, 2009.

[13] Documentation Ftrace. [Online]. Available at: http://www.mjmwired.net/kernel/Documentation/trace/ftrace.txt. Accessado in 2011, November.

[14] Operating System Family Share. November, 2011. [Online]. Available at: http://www.top500.org/charts/list/38/osfam

[15] Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A: System Programming Guide, March 2010.

[16] H. Yan, D. Garlan, B. Schmerl, J. Aldrich, and R. Kazman. DiscoTect: A System for Discovering Architectures from Running Systems. In Proc. of ICSE, pp. 470-479, 2004.

[17] L. Silva and K. Paixão and S. Amo and M. Maia, "On the Use of Execution Trace Alignment for Driving Perfective Changes", In the 15th European Conference on Software Maintenance and Reengineering, pp. 221-230, 2011.

[18] A. Hamou-Lhadj and T. Lethbridge, "Summarizing the Content of Large Traces to Facilitate the Understanding of the Behaviour of a Software System", In Proceedings of the 14th IEEE International Conference on Program Comprehension, IEEE Computer Society, pp. 181-190, 2006.

[19] M. Mit and M. Ernst. Static and Dynamic Analysis: Synergy and Duality. In Proc. of ICSE, pp 24-27, 2003.

[20] K. Sartipi, and N. Dezhkam. An Almalgamated Dynamic and Static Architecture Reconstruction Framework to Control Component Interactions. In Proc. of WCRE, pp 259-268, 2007.

[21] S. K. Lukins, N. A. Kraft, and L. H. Etzkorn, "Bug localization using latent Dirichlet allocation," Information and Software Technology, vol. 52, no. 9, pp. 972 – 990, 2010.

[22] J. Maletic and A. Marcus, "Support for software maintenance using latent semantic analysis," in Proc. 4th Annual IASTED International Conference on Software Engineering and Applications (SEA'00), Las Vegas, 2000, pp. 250–255.

[23] A. Marcus and J. Maletic, "Recovering documentation-to-source-code traceability links using latent semantic indexing," in Proc. of the 25th Intl. Conf. on Software Engineering, 2003, pp. 125 – 135.

An Approach for Software Component Reusing based on Ontological Mapping

Shi-Kuo Chang[1], Francesco Colace[2], Massimo De Santo[2], Emilio Zegarra[1], YongJun Qie[1]

[1]Department of Computer Science, University of Pittsburgh, Pittsburgh, PA 15260 USA
[2]Department of Electronic and Information Engineering, University of Salerno, 84084 Fisciano, Italy
{chang, ezegarra}@cs.pitt.edu, yongjun@pitt.edu,{fcolace, desanto}@unisa.it

Abstract

The ontological formalism is an effective way for enabling interoperability across heterogeneous systems, services and users. In this scenario, a very challenging problem is the learning process of a shared ontology: humans usually represent a same domain by the use of different ontologies, semantically similar, that can differ in their structure, concepts, relations and attributes. An effective solution to this problem is the introduction of methodologies for recognizing semantic similarities among the various views of the domain expressed by the ontologies. This paper shows how an effective ontological mapping approach can improve and support complex processes as software component reusing. First, an approach to the ontology mapping will be described and tested by the use of standards datasets. Its application to the software component reuse will be introduced in the second section of this paper. In particular, a case study and the obtained results will be discussed.

1. Introduction

Many aspects of our daily lives are managed through the adoption of complex and heterogeneous systems. Therefore, their users can experiment some difficulties sharing and developing services and resources. In this scenario, an effective solution is the adoption of the ontology formalism [1]. Ontology represents the main components of a knowledge domain according to the views that the various actors of this domain have about it. A common problem is the use of different ways for representing the same knowledge domain by each actor. Therefore, they create different ontologies, which differ in structure, concepts, attributes and relationships from the other ones, for representing the same domain [2]. In this way, ontology could lose its main feature: allowing the semantic interoperability among actors working on the same knowledge domain [12]. In literature, a solution to this problem is the introduction of methodologies for finding syntactical similarities among the various ontologies in order to obtain a shared and unique ontology [14]. A promising approach is known as ontology mapping and aims to recognize and find common components in the various ontologies. There are three main approaches to the ontology mapping: the lexical, the semantic and the structural approach. Each approach shows good performance under certain operative conditions, while exhibiting weaknesses in others. The main idea of this paper is the introduction of an ontology mapping methodology, which combines the previous approaches. So the obtained results are an original approach to the ontology mapping that is the starting point for a methodology for component reuse. In Section 2, the proposed ontology mapping approach will be described. In Section 3 the experimental results obtained by the use of the proposed approach on standard datasets and a comparison with other approaches will be discussed. In Section 4, we explore the application of ontology mapping to the software component reuse, which is an important issue in software engineering and some experimental results will be presented.

2. The Proposed Approach

What is ontology? This question seems trivial, but the answer is still not clear. An ontology formal definition could be the following: O={C, H_C, H_R, A, I}, where

- C is the set of concepts in a domain
- H_C is the set of the hierarchical relations among concepts
- H_R is the set of generic relations among concepts
- A is the set of axioms
- I is the set of concepts' instances

Ontology mapping can be described as follows: given two ontologies O_A={C_A, H_{AC}, H_{AR}, A_A, I_A} and O_B={C_B, H_{BC}, H_{BR}, A_B, I_B} the mapping operation is a function MAP: OxO -> O that gives an ontology O_C={C_C, H_{CC}, H_{CR}, A_C, I_C} where each set (S means one of the component of ontology) is obtained in the following way: $S_C = \{s_{ci} = s_{aj} = s_{bk})$ where sim(s_{aj}, s_{bk}) > Threshold_C $\forall s_{aj} \in S_A$ and $\forall s_{bk} \in S_B$ and *sim* is a function that measures the likelihood among s_{aj} and s_{bk}. In this paper, a methodology for ontology mapping based on the combination of various approaches that work at lexical, semantic and structural level to find equivalence among the various components of the ontologies is proposed. This approach can be so described:

```
// create mapping groups
for ($c \in O_1,\dots,O_N$) do
```

```
  if (type(c)="class") then
        add c to $groups_{cls}$
  else if (type(c)="property") then
        add c to $groups_{props}$
      end if
  end if
end for
classify ($groups_{cls}$)
classify ($groups_{props}$)
```

The generic classification phase classify($groups_{in}$) is the following:

```
//classify($group_{in}$)
if(size($group_{in}$)>1) then
  remove concept $c_r$ from $group_{in}$
for ($\forall c \in group_{in}$) do
  if ($c_r \in O_i\_and\_c \in O_j with\_i \neq j$)then
    if(type($c_r$) = "class" and type(c)="class")then
    index = calculateIndexSimilarityClasses($c_r$, c)
    else
      if(type($c_r$) = "property" and type(c)="property") then
      index = calculateIndexSimilarityProperties($c_r$, c)
      end if
    if(index>threshold) then
      add mapping found between $c_r$ and c
    end if
  end if
end for
  classify($group_{in}$)
end if
```

According to the proposed approach, the mapping process is a classification problem: it classifies the similarities among the classes, the properties and the relationships and then creates a new ontology that is a common layer representing a shared view of the various ontologies. As previously said, various approaches are in literature ([3]-[11]) in order to find the semantically similar components in the ontologies. The adopted functions are the following:

Editing Distance (ED): This function is so defined:

$$sim_{ed}(x,y) = 1 - \max(0, \frac{\min(|x|,|y|) - ed(x,y)}{\min(|x|,|y|)}) \in [0,1]$$

It aims to calculate the likelihood among words that labelling concepts in the ontology. In particular it compares the syntactical structure of the words and counts the characters that are in the same position in the words x and y by the use of the "ed" function. The value 1 means that the two words are similar.

Trigram Function (TF): This function aims to measure the number of similar trigrams that are in the words that label the concepts in the ontologies.

$$TF(x,y) = \frac{1}{1 + |tri(x)| + |tri(y)| - 2 * |tri(x) \cap tri(y)|} \in [0,1]$$

The function tri(x) gives the set of trigrams that are in the in the word x. The value 1 means that the two words are similar.

Semantic similarity index (SS): This index is so defined

$$SS(w_1, w_2) = \frac{1}{sim_{jc}(w_1, w_2)} \in [0,1]$$

where

$sim_{jc}(w_1, w_2) = 2 * \log P[LSuper(c_1, c_2)] - [\log P(c_1) + \log P(c_2)]$

This index aims to compare from a semantic point of view two words measuring their distance in the taxonomy defined in Wordnet [16]. The value 1 means that the two words are similar.

Granularity (GR): This index measures the mutual position of the words representing the concepts of the ontology in the WordNet taxonomy. This index is so defined:

$$GR(c_1, c_2) = \frac{\min[dens(c_1) * path(c_1, p), dens(c_2) * path(c_2, p)]}{\max[dens(c_1) * path(c_1, p), dens(c_2) * path(c_2, p)]} \in [0,1]$$

where dens(c) is the function representing the density of the concept c. This function is defined as E(c)/E where E is the ratio between the number's arc of the concept and the numbers of its parents while E(c) is the number of the sibling of the concept c. The function path(c_1, p) is the shortest path from c_1 to p that is the first parent common to c_2.

Attribute Index: This index aims to measure the numbers of similar attributes between two nodes. In particular, it is so defined:

$$sim_{att} = \frac{|X \cap Y|}{|X \cap Y| + \alpha(x,y) * \left|\frac{X}{Y}\right| + (1 - \alpha(x,y)) * \left|\frac{Y}{X}\right|} \in [0,1]$$

with $\alpha \in [0,1]$ is a parameter and X and Y are the number of attributes related to the two compared nodes. If this function gives value 1 it means that the words X and Y are similar.

Synonym Index (SI): This index aims to verify if in Wordnet there are synonyms of the word related to the concept in an ontology that label a concept in another ontology. This index can assume value 0 (no synonym) or 1 (synonym).

Derived Index (DE): This index aims to find in WordNet an adjective, representing a node of ontology, derived from the label of a concept that is in the other ontology. This index can assume value 0 (not derived) or 1 (derived).

Property Similarity Index (ISP): This index has the aim to verify the equality between the nodes evaluating their properties. In particular, the following indexes are introduced

- Equality Indexes among superclasses (ISC): this index verifies if the superclasses of the comparing classes are similar. This index can assume value 0 (no equality) or 1(equality).
- Equality indexes among equivalent classes (IEC): this index compares all the classes that are equivalent to the comparing classes. This index assumes value 1 if all the classes are equivalent and 0 otherwise.
- Equivalent Indexes of inherited classes (IIC): this index assumes value 1 if all the inherited classes of the comparing nodes are similar. Also, in this case this index assumes value 1 if all the inherited classes are equivalent and 0 otherwise.
- Equivalent Indexes of disjoint classes (IDC): this index evaluates the similarity among the disjoint classes of the comparing nodes. This index assumes values 1 if all the disjoint nodes are similar and 0 otherwise.

The full similarity index (ISP) is obtained by the following formula:

$$ISP = ISC*IEC*IIC*IDC$$

This index can assume value 0 or 1.

Similarity Index for entities (ISI): This index evaluates if the entities derived from the comparing nodes are equal. The comparison evaluated both the number of entities and their typology. This index can assume value 0 or 1.

Acronym (AC): This index aims to verify if in the two comparing nodes one word is the acronym of the other. If it is true this index is 1, otherwise it is 0.

Fingerprint Index (IM): This index verifies if the word that describes a comparing node is in the other word that describes the other nodes. If the word is contained in the other one this index is 1, otherwise it is 0.

Abbreviation (AB): This index measures if a word that describes a node is an abbreviation of the other that describes the other comparing node. If the word is an abbreviation of the other one this index is 1, otherwise it is 0.

Label (LA): This index measures if the two labels of the comparing nodes are equal. Also, in this case the index assumes value 1 if the nodes have the same label, and 0 otherwise.

There are some approaches in literature for the regroup of these indexes [13]. In this paper we propose the following gruping:

Syntactic indexes (IndSin): These indexes aim to detect the syntactical similarities among the various components of the ontology. The following indexes are syntactic indexes:

- Editing Distance (ED)
- Trigram Function (TF)
- Acronym (AC)
- Fingerprint (IM)
- Abbreviation (AB)
- Label (LA)
- Attributes (ATT)

Semantic indexes (IndSem): These indexes aim to compare the ontologies from a semantic point of view using the WordNet taxonomy. The set of semantic indexes includes:

- Semantic Similarity (SS)
- Granularity (GR)
- Synonym Index (SI)
- Derived (DE)
- Label (LA)

Structural indexes (IndStr): These indexes compare the ontologies from a structural point of view. The indexes of this set are:

- Attributes (ATT)
- Similarity Index for properties (ISP)
- Similarity Index for Entities (ISI)

The three sets of indexes are combined in order to map the ontologies:

$$Mapping(X,Y) = \theta*IndSin(X,Y) + \sigma*IndSem(X,Y) + \omega*IndStr(X,Y)$$

where: $\theta + \sigma + \omega = 1$. In particular:

$$\theta = IndSin(X,Y) / [IndSin(X,Y)+IndSem(X,Y)+IndStr(X,Y)]$$

$$\sigma = IndSem(X,Y)/[IndSin(X,Y)+IndSem(X,Y)+IndStr(X,Y)]$$

$$\omega = IndStr(X,Y)/[IndSin(X,Y)+IndSem(X,Y)+IndStr(X,Y)]$$

and

$IndSin(X,Y) = 0.5*(\alpha*ED + \beta*TF)+ 0.5*[max(AC, IM, AB)]$ where $\alpha = ED / (ED+TF)$ and $\beta = TF / (ED+TF)$

$IndSem(X,Y) = 0.5*(\gamma*SS + \delta*GR)+ 0.5*[max(SI, DE, LA)]$ where

$\gamma = SS / (SS+GR)$ and $\delta = GR / (SS+GR)$

$IndStr(X,Y) = 0.5*(ATT) + 0.5*[max(ISI, ISP)]$

After this first step, the mapping among the various nodes that are in the ontologies is obtained. The second step is the mapping among the relations. So the following index is introduced:

$IndRel(x,y) = min[Mapping(A,C), Mapping(B,D), RO(x,y)]$

where x and y are the comparing relations while A and B are their domains and C and D are their co-domains. In particular,

$RO(R_1, R_2) = sqrt\{CM[d(R_1), d(R_2)]* CM[r(R_1), r(R_2)]\}$ and

$$CM(C_1, C_2) = \frac{|UC(C_1, H_1) \cap UC(C_2, H_2)|}{|UC(C_1, H_1) \cup UC(C_2, H_2)|}$$

In this case H_1 is the taxonomy related to the concept C_1 while H_2 is the taxonomy related to the concept C_2. The function UC (Upward Cotopy), where the H function considers three levels in the up and down direction, is so defined:

$$UC(C_i, H) = \{C_j \in C \mid H(C_i, C_j)\}$$

The last step is the mapping among the attributes. This task is accomplished by the introduction of this index:

$$IndAtt(x, y) = \max(IndSin(x, y), \min(Mapping(A, C), equal(type_{range_x}, type_{range_y})))$$

After this phase, the mapping process is ended. I

3. Experiment Setup and Results

The experimental evaluation of the proposed approach has been obtained by the use of standard datasets. In particular, the experimental approach adopted was the same one suggested by the SEALS project [21]. The SEALS Project has developed a reference infrastructure, the SEALS Platform, for supporting the formal evaluation of semantic methodologies as the ontological mapping. This allows both large-scale evaluations to be run as well as ad-hoc evaluations by individuals or organizations. The SEALS evaluation setup aims at evaluating the competence of matching systems with respect to different evaluation criteria and focuses on demonstrating the feasibility and benefits of automating matching approach. The evaluation setup contains three different experimental scenarios:

- Scenario 1:
  a. Dataset: Benchmark - the goal of this dataset is the identification of the areas in which each matching algorithm is strong or weak. The test is based on a particular ontology dedicated to a very narrow domain of bibliography and a number of alternative ontologies on the same domain for which alignments are provided.
  b. Criteria: conformance with expected results
- Scenario 2:
  a. Dataset: Anatomy - the anatomy real world case is about matching the Adult Mouse Anatomy (2744 classes) and the NCI Thesaurus (3304 classes) describing the human anatomy.
  b. Criteria: conformance with expected results
- Scenario 3:
  a. Test data: Conference - collection of conference organization ontologies. This effort was expected to materialize in alignments as well as in interesting individual correspondences ('nuggets'), aggregated statistical observations and/or implicit design patterns.
  b. Criteria: conformance with expected results and alignment coherence

So in order to evaluate the performance of the proposed approach the following indexes, suggested by the SEALS project, have been adopted:

$$precision = \frac{\#correct\_mappings}{\#correct\_mappings + \#wrong\_mappings}$$

$$recall = \frac{\#correct\_mappings}{\#correct\_mappings + \#missed\_mappings}$$

$$F_{Measure} = \frac{[(b^2 + 1) * precision * recall]}{(b^2 * precision + recall)}$$

The $F_{Measure}$ has been evaluated with the parameter b set at the value 1. The results obtained by the use of the proposed approach have been compared with the same ones obtained by other methodologies developed in the literature and reported in the SEALS project web page. The experimental results are shown in Figure 1.

Benchmark Dataset

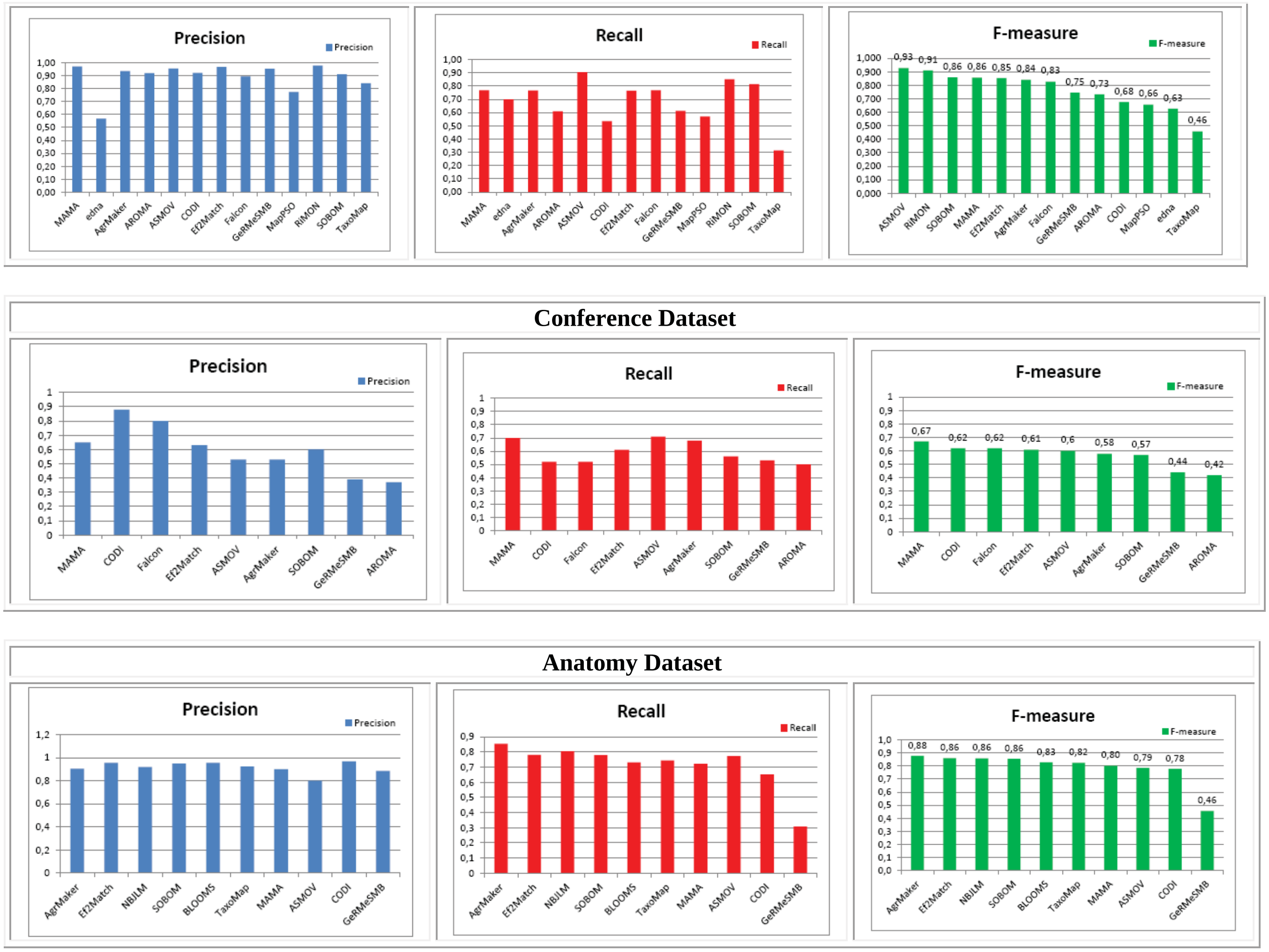


Figure 1. Experimental results of the proposed approach.

In figure 2 the average $F_{Measure}$ value has been reported (Figure 2).

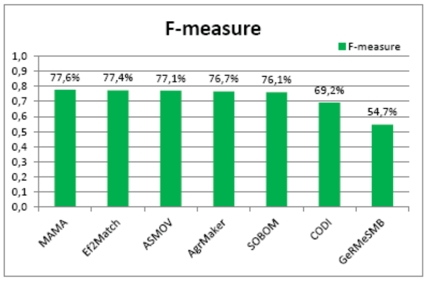


Figure 2. The average F-measure.

The proposed approach shows good results for each dataset and the best value of average $F_{Measure}$. It means that the proposed approach is very general and can manage the ontology mapping problem in various cases with good results. In the next section the application of the proposed ontology mapping approach to the component reuse problem.

4. Component Reuse based upon Ontology Mapping

Ontology mapping, which is an important part of ontology integration, can promote sharing and communication among different ontologies. The incorporation of ontology into software engineering can improve the reuse of software assets effectively [18]. In recent years, it has become less likely to develop complete new software systems from scratch. It becomes very important to develop software by adapting or combining existing reusable components [17]. We observe that requirement specification can provide a data source for ontology model and the vital link for the combination of software engineering and ontology [19]. With this insight, a software component reuse approach based on ontology mapping is formulated in Figure 3. The main idea is to process customer requirements and reusable components using ontology mapping techniques, and then construct the mapping between ontology nodes and reusable components. This approach can promote the reuse of software components. We can construct the target ontology model by analyzing requirement specification and then calculate the similarity between target ontology and source ontology using ontology mapping techniques. Therefore, we can identify the matched source ontology nodes and the corresponding sets of reusable components and then construct the mapping between the target ontology nodes and the reusable components. We assume that the mapping between source ontology nodes and reusable components has been realized, so every source ontology node has matched several reusable components. After the ontology mapping, the target ontology nodes also have matched the reusable components through the "bridge" of the source ontology nodes.

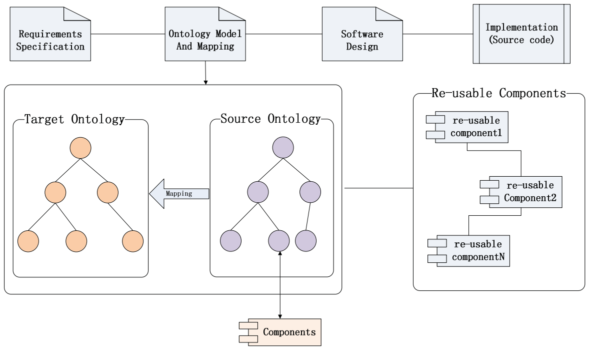


Figure 3. Software component reuse approach based on ontology mapping.

CROM Algorithm: Through the above analysis, we find that the mapping between ontology nodes and reusable components is the key to realize the CROM (Component Reuse through Ontology Mapping) algorithm. We can construct the mapping by using requirement engineering. With requirement engineering, we will decompose requirement specification into several fragments and every fragment of requirement contains several functional points. Each functional point contains the input and output, which are designed to match the requirement. The requirement and the corresponding input/output are the basis for the design and implementation of software components. Each node of ontology model is related to each fragment of requirement specification, and each fragment is related to several functional points so that each node of ontology model is also related to several functional points. This unique approach to construct the mapping between ontology node and reusable components by the functional point is the heart of the CROM algorithm. In general, the mapping operation is illustrated in Figure 4.

Ontology nodes and reusable components have N:N relationship. Every ontology node may correspond to several reusable components, and every reusable component may correspond to several ontology nodes. Even though function description of customer requirement is the same, the attributes are often different, so it is difficult for each software component to completely meet different requirements of different ontology node. We need to calculate the matching degree between software component and ontology node. We consider the matching degree between a concept of the ontology node C and a reusable software component S to be a number P(C, S) between 0 and 1, with 0 representing unmatched and 1 representing completely matched. The formula is: P(C, S) = f (S) / f (C), with f (S) representing the matched functional points number of the current reusable software component, and f (C) representing the total functional points number of the current ontology node.

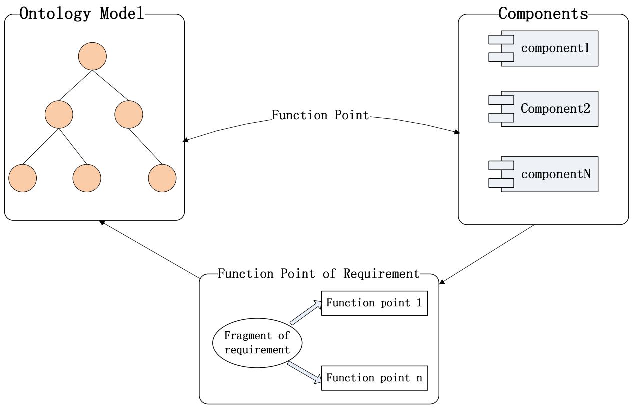


Figure 4. The mapping between ontology node and reusable components.

Experimental Design: The experiment is divided into two parts. The first part is to construct the application domain ontology model and the mapping between the application domain ontology nodes and the reusable software components. The data source is the application domain requirement specification and several sets of reusable software components. In order to make the experiment data more reasonable, we choose three groups ontology model, which are from very similar to the general ontology to very dissimilar. Firstly, we need to construct the *application domain ontology* model. We adopt the ontological concept to divide the application domain requirement specification into different application fragments. For example, the *general equipment application* may contain four fragments: equipment, purchase, storage and organization. Every fragment of the requirement contains the completely functional points and input/output and then we will construct the ontology on the application domain by processing different fragments describing the application domain and by mining them into common vocabularies as domain-specific concepts. Therefore, every application domain ontology node has several corresponding functional points and input/output. For example, equipment maintenance node may contain new, updating and delete three functional points. Secondly, we construct the mapping between application domain ontology nodes and reusable software components based on the functional points. We will match the functional points and input/output between every node and the reusable components by traversing the application domain ontology and then calculating P(C, S). If the number P(C, S) is greater than 0, the reusable component satisfies matching conditions. Because the reusable components are limited, we cannot guarantee all of the application domain ontology nodes have corresponding components that can be matched. Those nodes will be matched later during the software development and then we can put those components into the reusable components libraries.

The second part is to realize the reuse of the software components based on the ontology mapping technology and the result of the first part. Firstly, we need to manually calculate the rate of component reuse for three groups' ontology model. As previously said, the number P(C, S) represent the matching degree between a concept of the ontology node and a reusable software component. If the number P(C, S) is 1 with the same component and the different ontology model, we can consider that this component is reusable. Apparently, the rate of component reuse is much higher for the similar ontologies. The results are shown in Figure 5.

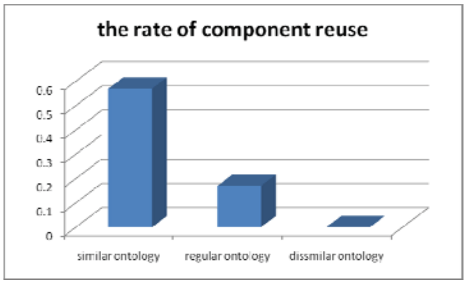


Figure 5. Contrast component reuse rate.

Secondly, we need to improve the rate of component reuse by using ontology mapping technology. We choose the similar ontology model for example, application domain ontology and sub-application domain ontology. We need to realize the mapping between sub-application domain ontology and application domain ontology. We can calculate the mapping index according to the proposed approach. The result shows that 45 nodes are equal, 9 nodes are similar and 3 nodes are not similar. According to the software components reuse approach based on the ontology mapping, it is highly likely that the reusable components that match the 45 application domain ontology nodes also match the 45 sub-application domain ontology nodes one by one.

Finally, we will check the mapping result between the sub-application domain ontology nodes and the reusable software components by the functional points. We can calculate the number P(C, S) between sub-application domain ontology nodes and the reusable components according to the second step of the first part, and then calculate r1, defined as the ratio of reused components over total number of components, to measure the rate of component reuse for the sub-application domain. For those components that are not directly reusable because of different attributes, but with high P(C, S) value, we can adjust the P(C, S) value according to results of the previous step, which was calculated by using the proposed approach to map the attribute concepts. Then we can calculate r2, the revised rate of component reuse for the sub-application domain. We have developed a software tool to calculate r2. This tool can adjust the number P(C, S) by analyzing the ontology mapping result and component attributes, and then calculate r2. This tool also can compare r1 and r2, and then draw the chart. The results are shown in Figure 6.

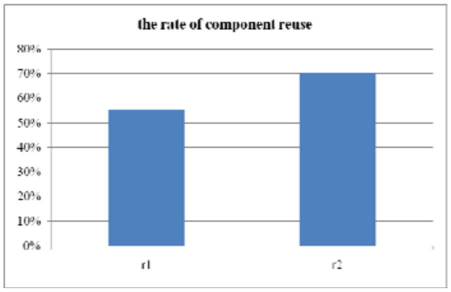


Figure 6. Improvement in component reuse rate.

Through this experiment, we conclude that we can increase the percentage of the components reuse by matching attribute names through ontology mapping. In other words, we can systematically change the name of variables in a program so that it can be reused in another sub-application domain. We also have developed two different components to realize component reuse algorithm and then select better component by using the Slow Intelligence System (SIS) framework [20]. The experiment is divided into three steps. The first step is to implement the super-component that can analyze the ontology mapping result. The super-component includes two components, namely BySAX and ByDOM. The BySAX component is to parse ontology mapping XML document by SAX (Simple API for XML), which is an event-based sequential access parser API developed for XML documents. The ByDOM component is to parse ontology mapping XML document by JDOM API, which is a Java-based document object model for XML. In the future, other ontology mapping algorithms can be added to this super-component. The second step is to modify several SIS core java files and config files, so as to support component reuse algorithms. The final step is to run the super-component based on the SIS framework and then obtain the analysis result. The result shows that ByDOM is the better algorithm, whose runtime is 18.54 seconds on a PC. The process is shown in Figure 7.

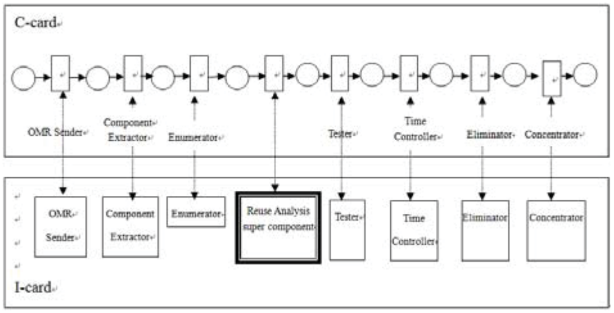


Figure 7. Component selection by using SIS approach.

The OMR (Ontology Matching Result) is sent to component extractor by OMR Sender, so that corresponding components can be extracted from components library, which P(C,S) is less than 1.Then the Enumerator invokes the super-component that creates various reuse analysis algorithms. The Tester collects and presents the test results. The Time Controller restarts the reuse analysis cycle with a different super component. The Eliminator eliminates the inferior algorithms, and finally the Concentrator selects the best algorithms with the optimal parameters.

5. Conclusion

This paper presented a novel approach for ontology mapping. It uses many existing and/or improved indexes and a novel way to combine them. Experimental results demonstrate the effectiveness of the approach approach. Future research topics include the continuous improvement of the various indexes and the definition of a merging approach. As a practical application of this methodology, the software component reuse approach based on ontology mapping emphasizes the application of the ontology mapping technology and the reuse of software components. In software engineering, ontology and ontology mapping technology can be very useful to promote the sharing and reuse of the domain knowledge. Through the mapping between the ontology nodes and reusable software components, we can achieve the reuse of the software assets from the ontology model to the software components. Further research topics include the refinement of mapping techniques between the ontology nodes and software components.

References

[1] Thomas C. Jepsen, "Just What Is An Ontology, Anyway?," It Professional, Vol. 11, No. 5, P p. 22-27, Sep./Oct. 2009, Doi:10.1109/Mitp.2009.105

[2] M. Shamsfard, “The State Of The Art In Ontology Learning : A Framework For Comparison,” Computer Engineering, 2007, Pp. 1-28

[3] L. Zhou, “Ontology Learning: State Of The Art And Open Issues,” Information Technology And Management, Vol. 8 , Mar. 2007, Pp. 241-252.

[4] H. Bao, H. Liu, J. Yu, And H. Xu, “An Ontology-Based Semantic Integration For Digital,” Architecture, 2005, Pp. 626 – 631

[5] G. Li, Z. Luo, And J. Shao, “Multi-Mapping Based Ontology Merging System Design,” Framework, 2010, Pp. 5-11.

[6] M. Mao, Y. Peng, And M. Spring, “Ontology Mapping: As A Binary Classification Problem,” 2008 Fourth International Conference On Semantics, Knowledge And Grid, Dec. 2008, Pp. 20-25.

[7] C. Zheng, Y.-Ping Shen, And L.I.N. Mei, “Ontology Mapping Based On Structures And Instances,” Machine Learning, 2010, Pp. 11-14.

[8] L. Guan-Yu And L. Shu-Peng, “Formal Concept Analysis Based Ontology Merging Method 2 3,” Science And Technology, 2010, Pp. 279-282.

[9] S. Li, H. Fan, Y. Wang, And L. Hong, “A Model And Approach For Heterogeneous Ontology Automatic Merging,” 2009 International Joint Conference On Computational Sciences And Optimization, Apr. 2009, Pp. 214-217.

[10] H. Xia, Z. Li, H. Wang, And Y. Gu, “A Lightweight Method Of Web Service Ontology Merging Based On Concept Lattice,” The 2nd IEEE Asia-Pacific Service Computing Conference (Apscc 2007), Dec. 2007, Pp. 304-311.

[11] Studer R, Benjamins Vr, Fensel D, “Knowledge Engineering: Principles And Methods”, IEEE Transactions On Data And Knowledge Engineering, 25(1-2): 161- 199, 1998

[12] Wu Ya-Juan Lang Ji-Sheng Shang Fu-Hua, “A Similarity-Based Approach For Ontology Mapping”, Proceedings Of 2009 4th International Conference On Computer Science & Education

[13] P. Shvaiko, J. Euzenat, “Ontology Matching”, Springer, 2007

[14] P. Shvaiko, J. Euzenat, “Ontology matching: state of the art and future challenges”, IEEE Transactions on Knowledge and Data Engineering, 2012

[15] Nie Zhao-Hui, Wang Yin–Ling, “Research On Attribute Mapping Between Similarity Ontology”, Computer Simulation. 2006

[16] Wordnet: http://wordnet.princeton.edu/

[17] Motoshi Saeki, "Ontology-Based Software Development Techniques", ERCIM News No.58, 2004

[18] Waralak V. Siricharoen, “Ontologies and object models in object oriented software engineering”, International Journal of Computer Science,33(1):1–6, 2007

[19] Haruhiko Kaiya, Motoshi Saeki, “Ontology Based Requirements Analysis: Lightweight Semantic Processing Approach”, Pp.1-8, 2005

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

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

Online Anomaly Detection for Components in OSGi-based Software

Tao Wang
Institute of Software, Chinese Academy of Sciences
Beijing 100190, P.R. China
Graduate University, Chinese Academy of Sciences
Beijing 100049, P.R. China
wangtao08@otcaix.iscas.ac.cn

Wenbo Zhang, Jun Wei, Jianhua Zhang, Hua Zhong
Institute of Software, Chinese Academy of Sciences
Beijing 100190, P.R. China
{zhangwenbo, wj, zhangjianhua07,
zhongh}@otcaix.iscas.ac.cn

***Abstract*—OSGi has become one of the most promising frameworks for managing component-based software. The OSGi-based components delivered by different vendors are usually black-box program units which lack source code and design documents. Thus it is difficult to evaluate their quality by static code analysis. However, defective components may lead to the failure of the whole system eventually. In this paper, we propose an online method for detecting anomalous components in OSGi-based software. A thread-tracing method is presented to monitor the CPU utilization of components. The method considers both the dynamic service invocation and static method invocation. Furthermore, based on the monitored data, we detect anomalous components by control charts, which can detect the anomalous trend of CPU utilization without prior knowledge. The experimental results show that our method is of high accuracy for monitoring CPU utilization in component perspective without significant overhead, and can detect anomalous components effectively.**

Keywords- Anomaly Detection; Component Monitoring; OSGi; CPU Utilization; Control Chart

I. INTRODUCTION

The component-based software engineering greatly improves the efficiency and quality of software development; organizations always adopt it for developing large-scale complex software [1]. In recent year, OSGi (Open Service Gateway initiative) has become one of the most promising frameworks for building and managing component-based software. The OSGi framework, which provides a component model and a service registry, is an execution environment for dynamically loadable components [2]. OSGi technology is attracting growing interest, and a large number of large-scale projects have released new versions with OSGi, such as JEE application server Websphere, IDE eclipse and the BMW automobile control system. The services based on OSGi are always implemented as components, and the COTS (Commercial Off-The Shelf) market around OSGi is emerging [3], where the number of third party components is increasing. However, a defective component may affect all the related components and lead to the failure of the whole system eventually. Thus it is a critical issue for COTS to ensure the quality of components [4].

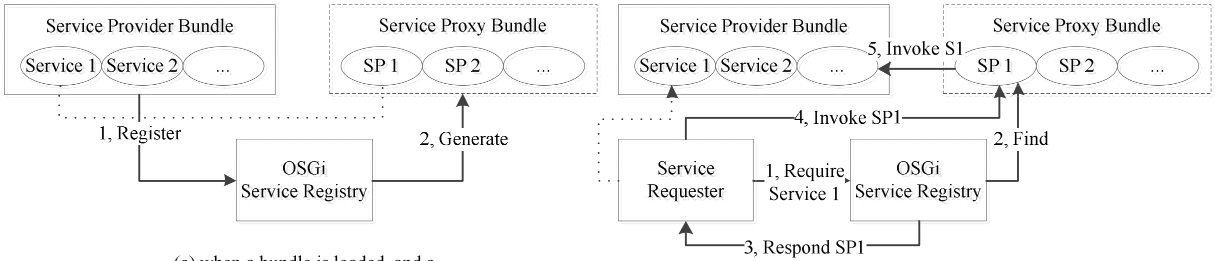
However, since the COTS components are usually black-box program units which lack source code and design documents, it is difficult to understand the characteristics of components, and evaluate their quality by static code analysis. Furthermore, some runtime factors, e.g. access sequences, concurrency number and resource usage, may cause contextual anomalies [21], which are difficult to be eliminated through testing [5]. Therefore, detecting anomalous components online is essential for improving the reliability of OSGi-based software.

This paper proposes an online method of detecting anomalous components in OSGi-based software. A thread-tracing method is proposed to monitor CPU utilization of components. It is an online method, which neither modifies software nor introduces significant overhead. Furthermore, the control charts for CPU utilization are introduced to detect anomalous components. They can detect the anomalous trends of CPU utilization without prior knowledge. The experimental results demonstrate that our method can monitor resource utilization in a high accuracy without significant overhead, and detect the anomalous components effectively.

The rest of this paper is organized as follows. Section II presents a thread-tracing method for monitoring components. Section III introduces control charts to detect anomalous components. Section IV provides experimental results to validate the method in accuracy, overhead and effectiveness. Section V discusses the limitations and our future work. Section VI reviews the related works, followed by conclusion in section VII.

II. MONITORING CPU UTILIZATION OF COMPONENTS

An OSGi service platform is composed of service providers, service requesters and a service registry. A service provider registers services to publish, and a service requester discovers services from the service registry to invoke. The service described as a Java interface is always packaged as a standard JAR file – "bundle", in which service implementation, related resource files and manifest files are included. Bundles interact with each other as service invocation. Since bundles are basic management units in OSGi, we take them as monitored targets. We analyze component-based software from CPU utilization which is an important property for evaluating a component [6].



(a) when a bundle is loaded, and a service is registered

(b) when a service is required

Figure 1 Proxy Generation

A thread is the basic unit to which the operating system allocates processor time. Thus the CPU utilization of a bundle is the sum of the CPU time consumed by different threads, which execute within the bundle. We have two monitoring perspectives that are bundle and thread. From the bundle perspective, threads are grouped into different bundles, and each thread belongs to a specific bundle. Thus we add the CPU time of every thread in the bundle. From the thread perspective, a thread execution is divided into some stages, each of which belongs to a specific bundle. Thus we add the CPU time of every stage in the bundle. Because of the frequent interactions between bundles by service invocation, the relations between bundles and threads vary dynamically. If we follow the first perspective, the thread schedule model should be modified, and significant overhead will be introduced as presented in [8]. Therefore, we adopt the second choice through tracing thread transfer between bundles.

It is easy to calculate CPU time utilized by a thread during a period using the JVMTI [22] provided by the JVM (Java Virtual Machine). Thus, how to divide the CPU time of a thread into different bundles becomes an essential question.

A. *Monitoring Dynamic Service Invocation*

Algorithm 1 describes the method of monitoring bundle CPU utilization. Every thread is tagged with a bundle ID, when a thread is created. OSGi invokes start () method in the Activator class to start the bundle, when a bundle is initialized. We set the bundle ID of the thread as the started bundle through labeling the thread before and after the start method in the OSGi platform (line 1-3). When a new thread is created, we set the bundle ID of the thread as that of its parent thread (line 4-8). A thread in bundles is traced to decide whether the thread transfer happens. When a service is invoked, if the service provider and the service consumer are in different bundles, thread transfer happens (line 9-12).

Service proxy is generated and used to provide services. We use an event-driven mechanism to trace service invocation through listening to the events in the service registry. To avoid affecting the execution code in the original bundle and deal with the arriving services during execution, we create a proxy object for every required service (line 13-16). Figure 4 describes the event-driven monitoring method. A proxy class is generated when a service is registered. The proxy class is instantiated when the service is invoked. We also modify the service registry to redirect service requests to the service proxy. Thus the proxy object instead of the original service provides service for a service consumer transparently. In every proxy class, the monitoring point is ins erted before and after the service invocation to label the changed bundle ID of a thread.

The CPU time of every bundle is calculated. If the bundle ID of a thread varies after entering the invoked service, the CPU time is calculated and added to the original bundle, and the time stamp is updated (line 17-21). After exiting from the service, the CPU time is calculated and added to the invoked bundle (line 22-28). Finally, the monitor calculates the CPU utilization for every bundle in period (line 29-31).

Algorithm 1：Bundle CPU monitoring

```
Input: OSGi-based software
Output: $u_1,u_2,…,u_n$, where $n$ is the number of bundles
1.  LISTEN (bundle $b$ is initialized)
2.      SET the bundle ID of thread $t$ as $b$ ;
3.      SET time stamp for $t$ ;
4.  LISTEN (thread $ts$ is initialized)
5.      GET current thread $tp$; //$tp$ is the parent of $ts$
6.      GET bundle ID of $tp$ as $bp$;
7.      CREATE new thread $ts$;
8.      SET bundleID of ts as $bp$;
9.  LISTEN (service $s_i$ invoke service $s_j$)
10.     GET the bundle ID of $s_i$ as $b_i$ ;
11.     GET the bundle ID of $s_j$ as $b_j$ ;
12.     IF ($b_i$ != $b_j$)
13.         GET proxy of $s_j$ as $p_j$ ;
14.         IF ($p_j$ = NULL)
15.           CREATE $p_j$ for $s_j$;
16.         ENDIF
17.         GET $bundle\_info_i$ of $b_i$ ;
18.         GET current thread $t$;
19.         $bundle\_info_i$.CPUTime += $t$.Calculate_CPU();
20.         SET time stamp for $t$ ;
21.         SET bundle ID of thread t as $b_j$;
22.         EXECUTE $p$ ;
23.         GET $bundle\_info_j$ of $b_j$ ;
24.         $bundle\_Info_j$.CPUTime += $t$.Calculate_CPU() ;
25.         SET time stamp for $t$ ;
26.         SET bundle ID of t as $b_i$;
27.         EXIT $p$ ;
28.       ENDIF
29. FOR (i=1;i<= n; i++)
30.     $u_i$ = $bundle\_info_i$.CPUTime ;
31. ENDFOR
```

B. *Monitoring Static Method Invocation*

Although the OSGi specification recommends developers to implement the interactions between components with service invocation, some developers are used to invoke the functions from other components with the traditional static method invocation. Thus we propose an AOP based method to trace the thread transfer between components. OSGi framework analyzes the metadata file recording the exported packages automatically, when a bundle is loaded dynamically. We extend the original OSGi framework, so that it reports the exported methods to our monitoring tool when the analysis is finished. Our tool decides whether the classes being loaded are exported by the bundle according to the report. Then, we use AOP to insert the monitoring points into the beginning and the ending of the public method in the class which exports methods.

We note that some exported packages are not invoked by other components. The thread transfer does not happen, when the invoking method and the invoked method in the same bundle. However, the redundant monitoring points introduce unnecessary overhead. Therefore, we use a static code analysis method to reduce the number of monitoring points before weaving class. Method invocations usually take the form of "targetObj.methodName(parameters)"; the key to analyzing the calling relationship is to know the possible types of the objects which the *targetObj* may point to. We use the class hierarchy analysis method [25] to gain this knowledge. All the subtypes of *targetObj*'s type are among the possible types, and we can get all the possible target methods denoted by the *methodName*. With the knowledge of the target methods in every invocation statement, we can easily acquire the calling relationship between methods, classes, and packages. If two packages in two different bundles have calling relationship in OSGi, the corresponding imported and exported packages should be specified in the metadata files of the two bundles. Thus we do not weave the methods invoked in the same bundle to reduce monitoring overhead.

The object of our AOP based method is to trace the thread transfer between components, when a component invokes the others with the static method invocation. The other steps of monitoring bundle CPU utilization are similar with the method introduced in subsection A.

III. ANOMALY DETECTION WITH CONTROL CHARTS

Based on the results of monitoring CPU utilization of components, we can further detect anomalous components. The metrics of CPU utilization help developers to evaluate components, and find underlying problems, for example CPU exhaustion caused by an endless loop. A control chart is a statistical tool used to distinguish between variation resulting from common causes and variation resulting from special causes [9] in a process. Thus we use control charts to detect the symptoms of CPU utilization.

Control charts monitor component resource utilization, and raise an alarm if the metrics are not in stability. The stability is defined as a state in which the resource utilization has displayed a certain degree of consistency in the past, and is expected to do so in the future. For example, in an application server, the CPU utilization of a web container should be kept within a reasonable range under stable workload. When a problem happens, e.g. a spin lock fault, the CPU utilization of the web container shows anomalous trends. Then control charts will detect the gradually increasing trend in CPU utilization of the component, even if it is still within a reasonable range.

We make use of the XmR (Individual X and Moving Range) control charts, in which the individual (X) chart displays individual measurement, and the moving range (MR) chart shows variability between one data point and the next. Two XmR charts are employed to represent CPU utilization for every component to detect anomalies, respectively. Collect CPU utilization x_i of every component in period. Thus we can get the control charts as follows.

$$\bar{x} = (x_1 + x_2 + ... + x_n) \Big/ n \ ,$$

$$mR_i = \left| x_{i+1} - x_i \right| ,$$

$$\overline{mR} = (mR_1 + mR_2 + ... + mR_n) \Big/ n \ ,$$

$$UCL_x = \bar{x} + \alpha \overline{mR} ,$$

$$LCL_x = \bar{x} - \alpha \overline{mR} ,$$

$$UCL_{mR} = \beta \overline{mR} ,$$

$$LCL_{mR} = None .$$

According to the statistics theory, we use constant α and β which are specified as 2.66 and 3.268, respectively [9]. Thus we can get XmR control charts, and any point out of the normal scale, which is between the LCL and UCL, will be detected as an anomaly.

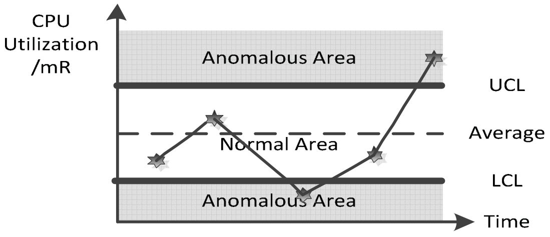


Figure 2 Example of Control Chart

Figure 2 gives an example of anomaly detection with the control charts. The x-axis represents sampling period, and the y-axis represents CPU utilization and mR. The scale between the LCL and UCL is regarded as the normal area, while the other scales are regarded as the anomalous areas. The system is detected in an anomaly status when the monitored points occur in the anomalous area. For example, the first, second and fourth points are in the normal area, while the third and fifth points are in the anomalous area.

IV. Evaluation

A. Monitoring Accuracy

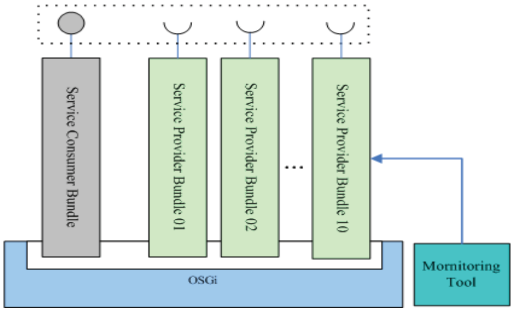

Service Consumer Bundle
Service Provider Bundle 01
Service Provider Bundle 02
...
Service Provider Bundle 10
OSGi
Monitoring Tool

Figure 3 Accuracy Evaluation Environment

Since there is no standard benchmark for evaluating the accuracy of our CPU monitoring method, we implemented a simulation to get a better understanding of the accuracy. The simulation consists of eleven bundles including a service consumer bundle and ten service provider bundles, as is shown in Figure 3. Each of these service provider bundles implements a service whose function is to loop for a fixed period, and the service consumer bundle invokes these services and counts the time spent on each service. These bundles are deployed on the OSGi platform, in which their services are registered. The service consumer bundle invokes the service provider bundles for several loops in random order. The service is assigned a quantitative execution time from 5 milliseconds till 50 milliseconds, respectively, as is described in the x-axis of Figure 4. The service consumer bundle invokes services for 10, 30, 50, 70, 90 and 110 loops in six experiments. The expected CPU time is the product of service time and the number of loops in each experiment, and then we compare the expected CPU time with the monitored measurement.

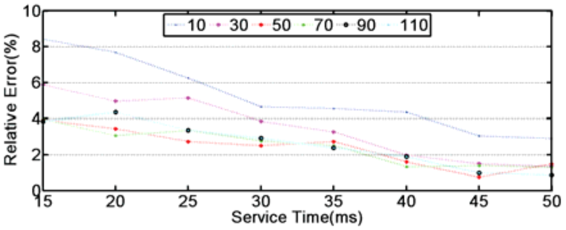


Figure 4. CPU Monitoring Accuracy

The accuracy of monitoring CPU time in terms of relative error is shown in Figure 4, in which the curves indicate the results of different experiments. It is seen that the relative error decreases with the service time increases. This is explained that the longer the effective service time, the smaller the proportion of overhead brought from tracing thread is. Furthermore, we observe that the relative error of the curve indicating 15 loops is higher than that of the curves indicating 70, 90 and 110 loops, and the curves indicating 70, 90 and 110 loops are consistent. When the service loops for more times, the error rate falls to about 1%. So our method has relatively high accuracy when the system runs for a longer time, because the stochastic error is canceled out.

B. Performance and Resource Overhead

In this part, we apply our method in a real application server - OnceAS [7], which has been transformed to the OSGi framework [10]. The overhead introduced by our method is evaluated in this subsection. The overhead is considered from two perspectives that are performance metrics including average response time and throughput, and the resource utilization including CPU time and heap memory.

In our experiments, we use a testbed of a standard three-tier e-commerce application, and simulate the operations of an online bookstore, according to TPC-W specification [11]. The client's access to the web site occurs as a session consisting of a sequence of consecutive individual requests. Users log in to the website, browse the products, add several books into the shopping cart, check out the order and log out of the website.

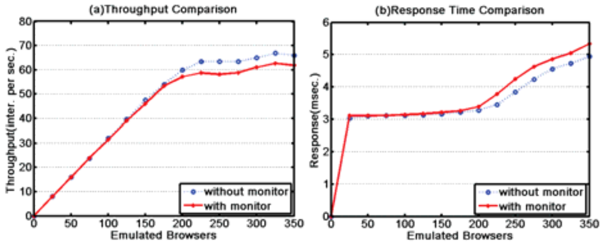


Figure 5. Performance Overhead

We simulate 25 to 350 concurrent browsers with different threads. The performance metrics evaluated for this scenario are the throughput that is the number of completed transactions per second and average response time that is the time taken to complete a transaction. As is shown in Figure 5, from the comparison of performance metrics, we can see that the performance of OnceAS with and without monitoring is considered equivalent. When the number of concurrent users is less than 175, the system is not saturated, so its throughput increases and response time keeps about 3.2 seconds as the number of users grows. After that point, the system becomes saturated, so its throughput does not increase anymore and the response time increases.

The performance overhead is less than 3.2 percent when the number of browsers does not exceed 175, and less than 10.3 percent after that point, so the performance overhead brought from monitoring is not significant.

This is explained by the low resource overhead. The CPU utilization of the system with monitoring is about 8 percent more than that of the system without monitoring. The overhead is caused by tagging threads and objects, and tracing threads with the JVMTI. At the same time, the memory utilization of the system with monitoring is about 9M bytes more than that of the system without monitoring. The overhead is caused by additional service proxy objects. From the above results, we can see our method without significant overhead is applicable in the real deployment environment.

C. Effectiveness of Detecting Anomalous Components

To validate our method for detecting anomalous components, we inject one typical fault in the HTTP service bundle which is responsible for parsing HTTP requests in OnceAS. Since injecting faults is a difficult issue which is out of our scope, we choose one typical real fault as analyzed in [12], and inject them with the method used in [18][19].

In the experiment, we also use the testbed mentioned in subsection B. We simulate 300 concurrent users from 1st to 8th minute, and 400 concurrent users from 9th to 15th minute. Each experiment lasts 15 minutes, the injected faults are triggered in the HTTP service in the 12.5 minute through timing automatically, and we monitor system status every minute. As is shown in Figure 6, the x-axis represents sampling time, the y-axis in (a)X-chart represents resource utilization per-interaction, and the y-axis in (b)mR-chart represents the moving range. The results show that individual measurements and moving ranges are in the normal scales before the fault is injected. Nevertheless, some anomalies are detected after the fault is injected.

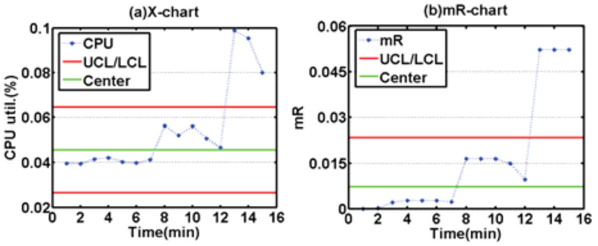


Figure 6. Control Chart for CPU Intensive Loop

The CPU intensive loop fault results from circular wait or endless loop in program such as spin lock fault. We inject it by inserting the additional computation operation which is a loop for 5ms. In each interaction with the injected service, these operations are triggered to consume additional CPU time. As is shown in F igure 6 w hich describes the XmR control charts of CPU utilization, after injecting this fault, the individual measurements and moving ranges are both higher than UCL from the 13th minute. Thus we detect that some anomalies occur in the HTTP service, and they are related to CPU processor.

V. Discussion and Future Work

The accuracy is an important factor for any monitoring tool. We use JVMTI, which is a naïve code based method, to calculate the CPU utilization of every thread. The native agent probes CPU for the calculation of cycles by sampling. The accuracy of our method is subject to the CPU resolution time of the operation system. It is impossible for us to im prove the absolute precision defined by the CPU resolution time. For example, that of the Windows XP is 15. 625 milliseconds, so our method cannot exceed that if our monitoring tool is deployed on the Windows XP. In the future work, we plan to use a statistical method such as Kalman filter to correct the monitored data. Furthermore, we can also use some platform-specific tools to improve the precision.

The overhead is an obstacle for the application of a monitoring tool. Tagging object introduces a major significant overhead in our method. If threads are created and destroyed, or components interact with each other frequently, lots of thread objects ought to be tagged. There will be lots of calls to the monitoring agent function, and these calls of the naïve codes are much slower than the Java method calls, so a great overhead is introduced. In the future work, we plan to use a dynamic map to r ecord the relationship between Bundle IDs and Thread IDs for reducing the traps in the naïve code to a minimum.

Although our method can detect anomalous components effectively, we cannot locate the root cause of a fault in a line of code, and operators ought to follow many other anomalous metrics to narrow down the possible causes. In the future work, we plan to extend our method to collect other metrics for fine-grain fault location. Furthermore, since OSGi provides a hot plug-and-play mechanism for components, we will implement a framework to rejuvenate the component through re-installing it or replace it with another one automatically, when an anomalous component is detected

VI. Related Work

A. Java Application Resource Monitoring

Prevailing methods to measure CPU consumption in Java applications mainly rely on native code libraries, which probe CPU for calculating cycles by sampling. For example, Magpie [13] uses Event Tracing with the processor cycle counter in Windows operating system. Similar methods on ot her operating systems include the Linux Trace Toolkit [14] and Solaris DTrace [15], etc. Binder et al. [16] proposed a portable CPU-management framework for Java, which tracked the number of the JVM bytecode instructions, and transformed them to the CPU consumption. These methods all aim at the whole JVM instead of components. We transform the resource perspective to component level in the OSGi framework.

The most related work to ours was conducted by Miettinen et al. [8], which created a unique ThreadGroup object for every bundle deployed on OSGi. The task executed by one thread in the original software is executed by different threads belonging to different ThreadGroups sequentially. However, this method modifies the thread schedule model. Moreover, complex thread scheduling mechanism, frequent thread switching operations and maintenance of a large number of threads bring significant overhead. Therefore, as discussed in [8], this OSGi-based monitoring method is only suitable for off-line simulation test, but not applicable in the real deployment environment.

B. Anomaly Detection

System operators usually employ management system like IBM Tivoli [23] and HP OpenView [24] to collect a large volume of monitoring data and set rules to trigger alerts. Current methods use specific analytic forms (e.g. linear regression [17], Auto Regressive models with e Xogenous inputs [18], Gaussian Mixture Models [19]) to model correlations between metrics, and the anomalies are detected when the correlations are broken. These methods take the whole application as target, so it is not applicable for locating specific anomalous components in the OSGi-based software. Gama et al. [20] presented a self-healing sandbox for the execution of third party components in OSGi. In the sandbox, no faults are propagated to the trusted parts of the application. The protocol between the trusted platform and the sandbox platform brings considerable performance overhead, and the correct functioning is based on a set of assumptions which may not apply to some real applications as discussed in their own work.

VII. Conclusion

The OSGi framework provides support for the management of component-based software. It is important for improving the reliability of OSGi-based software to detect anomalies in the granularity of component. This paper proposed a thread-tracing method for monitoring components running in OSGi from the perspectives of CPU utilization. Furthermore, control charts are also employed to detect anomalous components online. The experimental results demonstrate that our method is able to monitor the CPU utilization in a high accuracy without significant overhead, and can detect the anomalous components effectively.

Acknowledgment

This work is supported by the National Grand Fundamental Research 973 Program of China under Grant No. 2009CB320704, the National High Technology Research and Development Program of China under Grant No. 2012AA011204, the National Natural Science Foundation of China under Grant No.61173004 and the National Science and Technology Major Project of China under Grant No.2011ZX03002-002-01.

References

[1] G. Heineman, B. Councill, et al., "Component-based software engineering and the issue of trust," in Proceedings of the 22nd international conference on Software engineering, Limerick, Ireland, 2000, pp.661-664.

[2] OSGi Alliance. Available: http://www.osgi.org/.

[3] OSGi Alliance. About the OSGi Service Platform. Revision 4.1. http://www.osgi.org/wiki/uploads/Links/OSGiTechnicalWhitePaper.pdf, 2007.

[4] R. Pressman, D. Ince, Software engineering: a practitioner's approach: McGraw-Hill New York, 2005.

[5] S. G. Swapna, "Architecture-Based Software Reliability Analysis: Overview and Limitations," IEEE Transactions on Dependable and Secure Computing , vol. 4, no. 1, pp. 32-40, 2007.

[6] K. Heiko, "Performance evaluation of component-based software systems: A survey," Performance Evaluation, vol. 67, no. 8, pp. 634-658, 2010.

[7] W. Zhang, B. Yang, B. Jin, et al., "Performance Tuning for Application Server OnceAS", in Proceedings of the Second international conference on Parallel and Distributed Processing and Applications, vol. 3358, Eds., ed: Springer Berlin / Heidelberg, 2005, pp. 451-462.

[8] T. Miettinen, D. Pakkala,M. Hongisto, "A Method for the Resource Monitoring of OSGi-based Software Components," in Proceedings of the 34th Euromicro Conference on Software Engineering and Advanced Applications, Parma,Italy, 2008, pp. 100-107.

[9] G. A. Barnard, "Control Charts and Stochastic Processes," Journal of the Royal Statistical Society, vol. 21, no. 2, pp. 239-271, 1959.

[10] T.Wang, X. Zhou, J. Wei,W. Zhang, "Towards Runtime Plug-and-Play Software," in Proceedings of t he 10th International Conference on Quality Software, Zhangjiajie, China, 2010, pp. 365-368.

[11] D. A. Menasce, "TPC-W: a benchmark for e-commerce," IEEE Internet Computing, IEEE, vol. 6, no. 3, pp. 83-87, 2002.

[12] S. Pertet, P. Narasimhan, "Causes of failure in web applications," Parallel Data L aboratory, Carnegie Mellon University, CMU-PDL-05-109, 2005.

[13] P. Barham, A. Donnelly, R. Isaacs, et al., "Using magpie for request extraction and workload modelling," in Proceedings of t he 6th conference on Symposium on Opearting Systems Design & Implementation - Volume 6, San Francisco, CA, 2004, pp. 18-31.

[14] K. Yaghmour, M. R. Dagenais, "Measuring and characterizing system behavior using kernel-level event logging," in Proceedings of the annual conference on USENIX Annual Technical Conference, San Diego, California, 2000, pp. 2-15.

[15] B. M. Cantrill, M. W. Shapiro, A. H. Leventhal, "Dynamic instrumentation of production systems," in Proceedings of the annual conference on USENIX Annual Technical Conference, Boston, MA, 2004, pp. 2-2.

[16] J. Hulaas, W. Binder, "Program transformations for portable CPU accounting and control in Java," in Proceedings of the ACM SIGPLAN symposium on Partial evaluation and semantics-based program manipulation, Verona, Italy, 2004, pp. 169-177.

[17] M. A. Munawar, P. A. S. Ward, "A comparative study of pairwise regression techniques for problem determination," in Proceedings of the international conference of the center for advanced studies on collaborative research. NY, USA, ACM, 2007, pp.152-166.

[18] G. Jiang, H. Chen, Y. Kenji, "Modeling and Tracking of Transaction Flow Dynamics for Fault Detection in Complex Systems," IEEE Transactions on Dependable and Secure Computing, vol. 3, no. 4, 2006, pp. 312-326.

[19] Z. Guo, G. Jiang, H. Chen, et al., "Tracking Probabilistic Correlation of Monitoring Data for Fault Detection in Complex Systems," in Proceedings of the 36th international conference on Dependable Systems and Networks. Philadelphia, PA, IEEE, 2006. pp. 259-268.

[20] K. Gama, D. Donsez, "A Self-healing Component Sandbox for Untrustworthy Third Party Code Execution", In Proceedings of the 13th International Symposium on Component-Based Software Engineering. Springer-Verlag, Berlin, Heidelberg, 2010. pp. 130-149.

[21] V. Chandola, A. Banerjee and V. Kumar, "Anomaly detection: A survey," ACM Comput. Surv., vol. 41, pp. 1-58, 2009.

[22] JVMTI, http://java.sun.com/j2se/1.5.0/docs/guide/jvmti/.

[23] IBM Tivoli, http://www.cdwg.com/content/brands/ibm/tivoli.aspx.

[24] HP OpenView, http://www.osalt.com/openview-network-node-manager.

[25] J. Dean, D. Grove and C. Chambers, "Optimization of object-oriented programs using static class hierarchy analysis," the 9th European Conference on Object-Oriented Programming. Aarhus, Denmark, 7–11 August, 1995. Springer. pp. 77-101.

An Exploratory Study of One-Use and Reusable Software Components

Reghu Anguswamy
Software Reuse Lab
Computer Science, Virginia Tech.
Falls Church, Virginia, USA
reghu@vt.edu

William B. Frakes
Software Reuse Lab
Computer Science, Virginia Tech.
Falls Church, Virginia, USA
frakes@cs.vt.edu

Abstract— **One hundred and seven implementations of a one-use stemmer component and the equivalent reusable programs were analyzed in this exploratory study. Subjects were given a list of 19 reuse design principles and asked to indicate which ones they used. A ranking of the design principles by frequency of use is reported. One-use and the equivalent reusable components were analyzed using measures of the pairs in terms of size in SLOC (source lines of code), effort in hours, number of parameters, and productivity as measured by SLOC/hours to develop. Reusable components were significantly larger than their equivalent one-use components and had significantly more parameters. The effort required for the reusable components was higher than for one-use components. The productivity of the developers was significantly lower for the reusable components compared to the one-use components.**

Keywords: Software reuse, software engineering, empirical study, reusable component, reuse design principles

I. Introduction

Many reuse design principles have been proposed [1-4] but there has been little empirical analysis of their use. The software reuse literature often refers to a one-use component and its reusable equivalent, but there has been little study of this concept. In this study, we examined the reuse design principles used by one-hundred and one subjects in converting a one-use s-stemmer component to make it reusable. Our purpose was to conduct an exploratory analysis to identify the most commonly chosen reuse design principles used to develop the components and to quantify the differences in size, number of parameters used and effort between one-use and reusable components.

Software reuse, the use of existing software artifacts or knowledge to build new systems, is pursued to realize benefits such as improved software quality, productivity, or reliability[5]. Approaches to measuring reuse and reusability can be found in [6]. Software reuse in industry has been studied and its benefits analyzed [7-12]. These papers document an improvement in software quality and productivity due to software reuse. There are also many types of software reuse [6]. In the paper by Mohagheghi and Conradi [8], it has been pointed out that although there is positive and significant evidence for improved productivity, further study is required to measure the actual gains in productivity. Our paper measures the effect on productivity of a software developer making a one-use component reusable.

Component based development in software reuse was presented as early as 1968 by McIlroy [13] suggesting that interchangeable pieces called software components should form the basis for software systems. Our study is based on the component based approach and the reuse design principles presented in this paper aim at the design and implementation of simple reusable software components. A software system developed with reusable components follows a *'with'* reuse process while a component designed to be reused in other systems follows a *'for'* reuse process.

A. One-use component vs. Reusable component

The distinction between a reusable and its equivalent one-use component is an intuitive concept that is not precisely defined. One-use components are generally written for specific applications and are not meant to be used again. Reusable components on the other hand are developed to be used more than once within a domain or across domains for various applications in various environments. Reusable components are generally developed by taking a one-use component and modifying it either to add more functionality or changing it to work in other environments following a re-engineering approach. They can also be developed from scratch. Our study involves reusable components built by re-engineering one-use components. It follows that compared to equivalent one-use components; reusable components tend to be larger, more complex and slower. They also should have more potential input/output types and more parameters.

As a simple example of a one-use versus a reusable component, consider the "hello world" program, `hello.c`, as a one-use component and the `anymessage.c` program as its reusable equivalent. Here is the code for each:

```
//hello.c (one-use component)
         main(){printf("hello world\n");}


//anymessage.c (reusable component)
main(argc, argv) /* print any message to output */
       int argc; char *argv[];
       {
              int i;
                for (i=1; i< argc; i++)
                     printf("%s ",argv[i]);
              printf("\n");
       }
```

Table I shows the relationships between `hello.c` and `anymessage.c` in terms of attributes such as size, complexity, etc. We ran the `wc` program in the Unix environment [http://www.computerhope.com/unix/uwc.htm] on `hello.c` and `anymessage.c`. As predicted, the reusable program is larger and has more parameters. We also hypothesize that `anymessage.c` will be more complex, require more testing, require more design knowledge, have higher execution speed, and require more time to develop. Thus Table I summarizes our theoretical model of the relationship between a one-use and a reusable component. In the remainder of this paper, we will empirically evaluate certain aspects of this model.

TABLE I: `hello.c` (one-use component) VS. `anymessage.c` (reusable component)

Attribute	`hello.c`	`anymessage.c`
Size (lines, chars, executable)	(1, 8, 9878)	(10, 25, 9931)
Parameters	0	2
Domain Knowledge	Low	Medium
Testing	<	>
Design	<	>
Execution Speed	<	>
Effort	<	>

B. Motivation

For practitioners and researchers, there are two motivations in our study. One is that even though the relation between software quality and reuse has been established, no empirical study has been found comparing one-use and equivalent reusable components. The other motivation is that practitioners and researchers need to address the problem of how to build reusable components. This exploratory study used a comprehensive list of reuse design principles presented in the past two decades for software reuse and identified the most used reuse design principles. This can be a guideline for building reusable components. One major limitation of our study is that the components studied are small in size. However, this is an exploratory study and with 107 subjects, nearly all of whom have some experience in software engineering. The sample size is adequate for comparing one-use and reusable components. Also, this exploratory study is a baseline for future study on designing, building, and measuring reusable components.

One study that is similar to ours is presented by Seepold and Kunzmann [14] for components written in VHDL . However, the major limitation in that study was that it involved only four components - two one-use and two equivalent reusable components. According to that study the complexity, effort and productivity were all higher for reusable components. The reasons identified were due to overhead in domain analysis, component verification and documentation.

This paper is organized as follows: section II provides an overview of the reuse design process and principles; section III presents the hypotheses analyzed in this study; section IV discusses the method employed, metrics used for evaluation, and the s-stemmer component used in our study; section V presents the results and discussion; and section VI presents the conclusions and future work.

II. REUSE DESIGN PRINCIPLES

How to make a software component reusable has been one of the key questions for software reuse research. Reusable components may be built from scratch or re-engineered from existing artifacts. As can be seen from Fig. 1., the quality of the reusable components may be measured in terms of safety (when implemented and/or merged with another component), execution speed (generally the reusable components are slower than the one-use components), cost, and size. Size may be measured by source lines of code (SLOC). We assume that the higher the SLOC, the higher will be the complexity resulting in higher numbers of faults and parameters.

Many reuse design principles have been proposed. These are summarized in the mindmap in Fig. 1 based on the work by [1]. The principles are at various levels of abstraction. The 3 C's model of reuse design [15], for example, was developed to explore the reuse design process in a general framework. It specifies three levels of design abstraction: 1. Concept – representation of the abstract semantics. 2. Content – implementation details of the code or software. 3. Context – environment required to use the component.

III. HYPOTHESES

This paper aims primarily to study and test four hypotheses related to the reusable components. Due to the higher complexity and functionality of the reusable components, their size (in SLOC - source lines of code), effort (in hours), the productivity (in source lines of code per hour) and number of parameters should be significantly higher than their equivalent one-use components. The choice of the metrics is discussed in section 4.3. These hypotheses are summarized in equations (1), (2), (3), and (4). $\boldsymbol{SLOC_{Reuse}}$ is the actual source lines of code in the reusable component while $\boldsymbol{SLOC_{ReuseDiff}}$ is the difference in the source lines of code between the reusable and one-use components. The difference is considered for the productivity of reusable components because the reusable components studied in this paper were not built from scratch; instead , they were reengineered by modifying the one-use components based on the reuse design principles given in Fig. 1.

$$SLOC_{Reuse} > SLOC_{one\text{-}use} \quad (1)$$
$$Effort_{Reuse} > Effort_{one\text{-}use} \quad (2)$$
$$SLOC_{ReuseDiff/hour} > SLOC_{one\text{-}use/hour} \quad (3)$$
$$Parameters_{Reuse} > Parameters_{one\text{-}use} \quad (4)$$

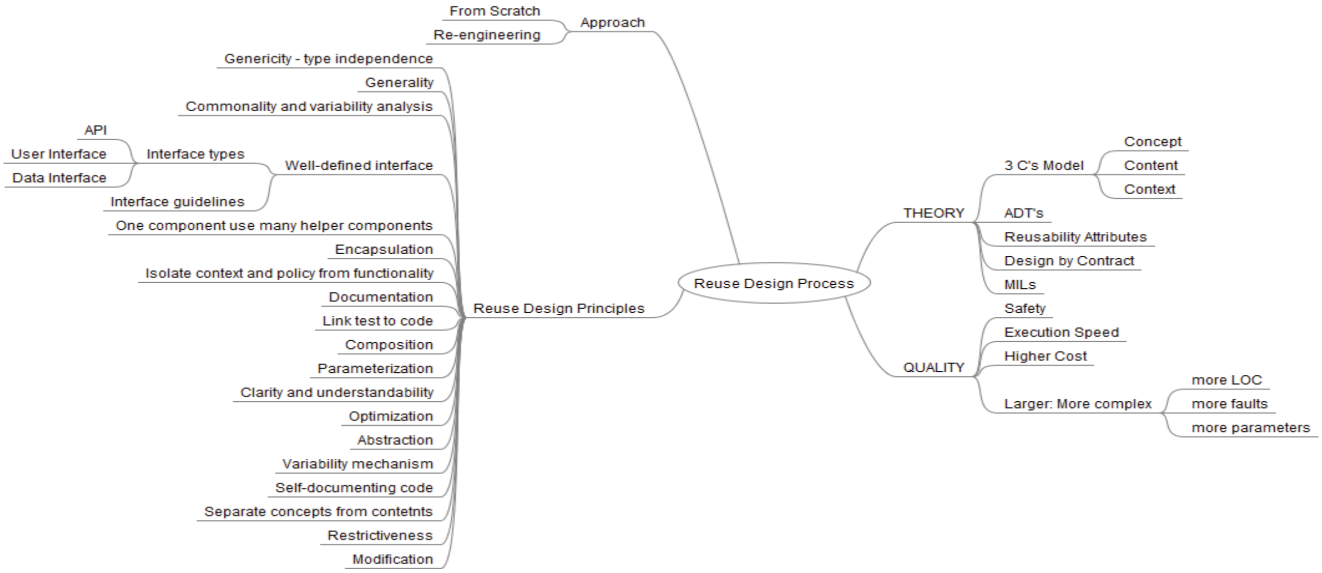


Figure 1. Mindmap of Reuse Design Process

IV. METHOD

Based on the faceted classification on types of software reuse by Frakes and Terry [6], the reuse design in our study involves development scope as internal, modification as white box, domain scope as vertical, management as ad hoc, and reused entity as code. A total of 107 subjects participated in this study. Nearly all the subjects were technical professionals with at least some experience in software engineering. The subjects were given an assignment to build a one-use software component implementing the s-stemming algorithm [16] and were later asked to convert their one-use stemmer component to a reusable component. The subjects were students either at Master's or Ph.D. level at Virginia Tech., USA.

A. A S-Stemmer Component

Three rules specify the s-stemming algorithm as follows (only the first applicable rule is used):

If a word ends in "ies" but not "eies" or "aies" then Change the "ies" to "y", For example, cities → city

Else, If a word ends in "es" but not "aes", "ees", or "oes" then change "es" to "e", For example, rates → rate

Else, If a word ends in "s", but not "us" or "ss" then Remove the "s", For example, lions → lion

The subjects were given lectures on the topics of software reuse, domain engineering and reuse design principles. The mindmap of the reuse design process as given in Fig. 1 was the basis of the lecture. One hundred and one of them converted their one-use components to an equivalent reuse component based on the reuse design principles in Fig. 1. The reuse design process followed was the reengineering method and not from the scratch method i.e. an existing component was modified to be reusable. The subjects were asked to follow a 'for' reuse design process i.e. design for future use.

The programming language used was Java. The reusable components were compared with one-use components based on the size (SLOC), effort (time in hours), number of parameters, and productivity (SLOC/hr).

B. Data Collection

For both the one-use and reusable components the subjects were asked to report the time required for developing the component. For the reuse components, the subjects were also asked to indicate and justify the reuse design principles (from Fig. 1.) that they used. One hundred and seven students successfully built the one-use components and 101 built the equivalent reusable component; six subjects did not build the equivalent reusable component and three of those who submitted did not report back the time required for building the component. All the components, both one-use and reusable components were graded as part of the assignment and required to satisfy two criteria: (1) the components must compile and execute error-free, and (2) the components must provide the right solutions for a set of test cases.

C. Evaluation Metrics

Source lines of code or SLOC is one of the first and most used software metrics for measuring size and complexity, and estimating cost. According to a survey by Boehm et al.[17], most cost estimation models were based directly on size measured in SLOC. Some of them are the COCOMO [18], COCOMO II [19], SLIM [20], SEER [21]. In COCOMO and COCOMO II the effort is calculated in man-hours while the productivity is measured in SLOC written per hour. Many empirical studies have also been based on measuring the complexity of software components by measuring SLOC [22-25]. There are also empirical studies where productivity of software components is measured in SLOC/hr [22, 23, 25, 26].

Herraiz et al. [27] studied the correlation between SLOC and many complexity measures such as the McCabe's cyclomatic complexity [28] and Halstead's metrics as given in

[29]. In their study they have presented empirical evaluations showing that SLOC is a direct measure of complexity, the only exception being header files which showed a low correlation with the McCabe's cyclomatic complexity measures. Research by Graylin et al. presented evidence that SLOC and cyclomatic complexity have a stable practically perfect linear relationship that holds across programmers, languages, code paradigms (procedural versus object-oriented), and software processes [30]. Linear models have been developed relating SLOC and cyclomatic xomplexity. Buse et al [31], for example, presented a study where they show a high direct correlation between the SLOC and the structural complexity of the code. A study by Gaffney [32] reported that the number of faults in a software component is directly correlated to source lines of code (SLOC). Krishnan et al. [33] also reported an empirical study that showed a direct correlation between SLOC and the number of defects in software components.

Based on these studies, we decided to compare the one-use and reusable components in our study based on the size (in SLOC), effort (man-hours) and productivity (in SLOC/hr).

V. Results and Analysis

A. Reuse Design Principles

Table II shows the summary of the usage of reuse design principles by the subjects. The mean number of principles used by a subject was 5.4. A *well-defined interface* (#1) was the most highly ranked principle and was used for about half of the reusable components. *Link to documentation* was ranked 2 and used in about 42% of the reusable components. Documentation has always been recommended and widely used in the programming world. *Clarity and understandability* of the code ranked next. This principle allows the users of the component a better and easier way of comprehending the code for future use. The next three principles were *generality, separate concepts from contents* and *commonality and variability*.

B. SLOC, Effort, Productivity and Parameters

The source lines of code for the one-use (N=107) and reusable components (N=101) were measured using the sloccount tool [http://www.dwheeler.com/sloccount/]. The notched box plots of the SLOC measured for the one-use and reusable components are shown in Fig. 2.

$$SLOC_{ReuseDiff} = (SLOC_{Reuse} - SLOC_{one\text{-}use}) \tag{5}$$

The effort taken in terms of time (hours) and the number of parameters are shown in Fig. 3 and 4 respectively. Fig. 5 compares the productivity (in terms of SLOC/hour) of the developers for one-use vs. reuse components. $SLOC_{ReuseDiff}$ is considered for the productivity of reusable components because the reusable components studied in this paper were not built from scratch; instead , they were reengineered by modifying the one-use components.

$$SLOC/hr_{ReuseDiff} = (SLOC_{Reuse} - SLOC_{one\text{-}use}) / Effort_{Reuse} \tag{6}$$

TABLE II. Ranking of reuse design principles used

Rank#	Reuse Design Principle	Count#
1	well defined interface	56
2	link to documentation	43
3	clear and understandable	42
4	generality	41
5	separate concept from contents	40
6	commonality and variability	31
7	linking of test to code	24
8	encapsulation	23
9	one component use many	21
10	composition	19
11	variability mechanism	13
12	parameterization	12
13	genericity	11
14	optimization	9
15	restrictiveness	7
16	modification	3
17	isolate context and policy	1
18	abstraction	1
19	self-documenting code	1

Understanding and interpreting box plots can be found in [34]. If the notches of boxplots of different groups overlap, then there is no significant difference between the groups and if they do not overlap, there is significant difference between the groups.

The median of SLOC significantly increased for the reusable components to 92 lines of code as compared to 51 for the one-use components, an increase of 80%. The average SLOC was 62 and 110 for the one-use components and the reusable components respectively. The notches of the two box plots do not overlap and this indicates a statistically significant difference between the sizes of the two components. This increase is due to incorporating more functionality in the reusable components. The boxplots in Fig. 2 also shows that there is much more variability in the SLOC measure for the reusable components. This may be because the reusable components have more functionalities and those functionalities vary from subject to subject based on the understanding of the reuse design principles; while for the one-use components the subjects may have had a more similar understanding of the functionality. From Fig. 3, the median of the time taken to implement the components was 5.0 hours and 8.0 hours respectively for the one-use and reusable components. Average time taken was 3.6 and 6.5 hours for one-use components and reusable components respectively. The notched areas of the box plots overlap for the two and this indicates no significant difference. As was the case for SLOC, the variability is higher for the reusable components. The inter quartile range for the one-use and reusable components are 3.0 hours and 5.5 hours respectively while the standard deviations are 3.0 hours and 6.8 hours respectively.

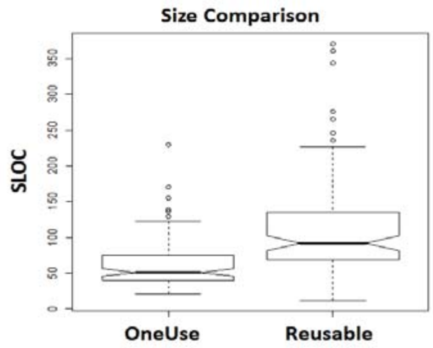


Figure 2. Comparison of actual size (SLOC)

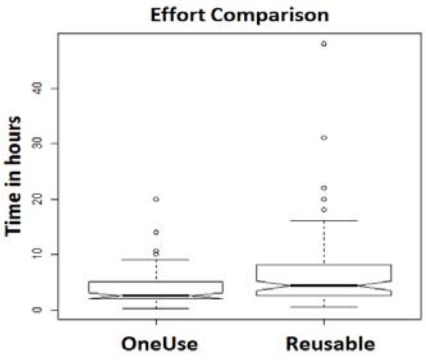


Figure 3. Comparison of Effort (hours)

As can be seen in Fig. 4, the number of parameters for the reusable components was significantly higher than for the one-use components. The medians were 2 and 5 for one-use and reusable components respectively. The mean number of parameters for the one-use components was 2.6 while the reusable components were 5.4. 40% of the one-use components had only a single parameter. In this case the variability is somewhat larger for the reusable components. As can be seen in Fig. 5, the median of the productivity was 21.0 and 6.45 SLOC/hr for the one-use and reusable components respectively. The mean for the productivity of one-use components (30.0 SLOC/hr) is almost three times the productivity the reusable components (10.6 SLOC/hr). The notches do not overlap and indicate significant difference. This may be because more time may have been spent on the design of the reusable component than on coding when compared to the one-use component. For the productivity, the variability of the one-use component is higher than the equivalent reusable components. The standard deviations are 29.2 SLOC/hr for one-use and 15.1 SLOC/hr for reusable components. The inter quartile range for the one-use components is 25.75 SLOC/hr while it is only 9.3 SLOC/hr for the reusable components.

An outlier in the effort for one-use component was the same subject who had an outlier in the SLOC (the subject who had the second most SLOC in one-use component). This indicates a higher effort for higher SLOC. The outliers for the number of parameters for one-use components and the same subject also caused outliers for reusable components. Using more parameters might be a programming style followed by the subjects.

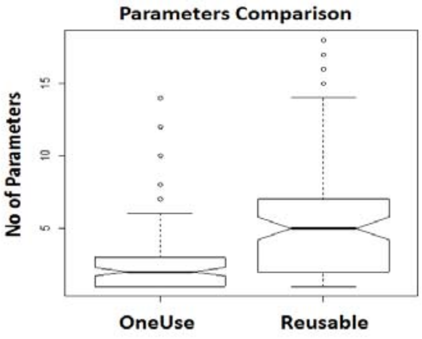


Figure 4. Comparison of #parameters

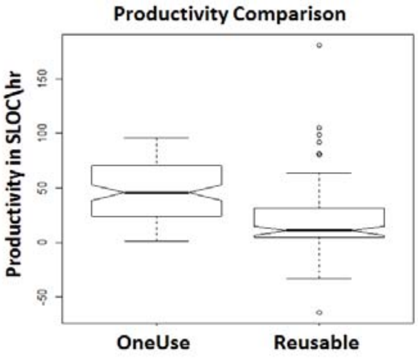


Figure 5. Comparison of productivity (SLOC/hr)

C. Matched Pair t-tests

SLOC, effort, productivity and the number of parameters were compared using the matched pair t-tests. For this analysis, the difference in the values of the one-use and reusable components was first calculated and this difference was then analyzed using one-sample t-test with a hypothetical test mean of zero. The results are shown in Table III. Table III shows that the SLOC, effort, and the number of parameters are statistically siginificantly higher. The productivity also shows a statistically significant difference. The reusable components have significantly lower productivity. Comparing the values of Cohen's d [35] the effect sizes are "*large*" for SLOC, number of parameters, and productivity, and "*medium*" for effort.

TABLE III. Matched pair t-test statistics (p<0.0001)

Variable	Mean	Std. Dev.	df	t	Cohen's *d*
$(\textbf{\textit{SLOC}}_{Reuse} - \textbf{\textit{SLOC}}_{one\text{-}use})$	48.6	53.7	100	9.09	0.89
$(\textbf{\textit{Effort}}_{Reuse} - \textbf{\textit{Effort}}_{one\text{-}use})$	3.09	5.9	97	5.1	0.56
$(\textbf{\textit{Parameters}}_{Reuse} - \textbf{\textit{Parameters}}_{one\text{-}use})$	2.76	2.87	100	9.64	0.88
$(\textbf{\textit{SLOC}}_{ReuseDiff/hr} - \textbf{\textit{SLOC}}_{one\text{-}use/hr})$	-20.5	31.67	97	-6.4	-0.81

VI. CONCLUSIONS AND FUTURE WORK

One-use components and their equivalent reuse components were analyzed for their sizes and the number of parameters. Both of these were found to be significantly higher for the reusable components. Programmer effort in terms of hours was also analyzed and observed to be significantly higher for reusable components. The productivity in SLOC/hour for the SLOC difference of reusable components was significantly lower than those for the one-use components. The higher SLOC for the reusable components is due to the additional functionality of the reusable components. The productivity was also affected by the additional functionalities introduced in the reusable components as the productivity was lower for the reusable components. This may be because the subjects might have spent more time in designing for the reusable components and modifying the existing one-use components as they are following the reengineering process for reuse design and not the build from scratch method. The effort required for reusable components was also significantly higher than those for the one-use components. The number of parameters was significantly higher for the reusable components than those for the one-use components. This may be because of more code within the component to realize the additional functionalities for reusability.

Nineteen reuse design principles were taught to the subjects. The most frequently used based on pareto ranking of the principles were presented. The six most frequently used reuse design principles were – *well-defined interface, link to documentation, clarity and understandability, generality, separate concepts from contents* and *commonality and variability.* The reuse design principles of *isolation of context and policy from functionality, abstraction* and *self-documenting code* were least used by the subjects. This may be because the components developed were relatively simple.

Future work may include an empirical study using a more complex algorithm. Future work may include components built in various other languages as well. *Design with reuse* may also be studied in future by having subjects use the reusable components developed in this study. The study will focus on identifying whether a correlation exists between the reuse design principles used and the ease of reusability.

[1] W. B. Frakes and D. Lea, "Design for Reuse and Object Oriented reuse Methods," presented at the Sixth Annual Workshop on Institutionalizing Software Reuse (WISR '93), Owego, NY, 1993.

[2] J. Sametinger, *Software engineering with reusable components*. Berlin Heidelberg, Germany: Springer Verlag, 1997.

[3] B. Weide*, et al.*, "Reusable software components," *Advances in computers,* vol. 33, pp. 1-65, 1991.

[4] M. Ezran*, et al.*, *Practical software reuse: the essential guide*. London: Springer Verlag, 2002.

[5] W. B. Frakes and K. C. Kang, "Software reuse research: status and future," *IEEE Transactions on Software Engineering,* vol. 31, pp. 529-536, 2005.

[6] W. Frakes and C. Terry, "Software reuse: metrics and models," *ACM Comput. Surv.,* vol. 28, pp. 415-435, 1996.

[7] R. van Ommering, "Software reuse in product populations ," *IEEE Transactions on Software Engineering,* vol. 31, pp. 537-550, 2005.

[8] P. Mohagheghi and R. Conradi, "Quality, productivity and economic benefits of software reuse: a review of industrial studies," *Empirical Software Engineering,* vol. 12, pp. 471-516, 2007.

[9] P. Mohagheghi and R. Conradi, "An empirical investigation of software reuse benefits in a large telecom product," *ACM Transactions on Software Engineering Methodology,* vol. 17, pp. 1-31, 2008.

[10] W. B. Frakes and G. Succi, "An industrial study of reuse, quality, and productivity," *Journal of Systems and Software,* vol. 57, pp. 99-106, 2001.

[11] M. Morisio*, et al.*, "Success and Failure Factors in Software Reuse," *IEEE Transactions on Software Engineering,* vol. 28, pp. 340-357, 2002.

[12] W. C. Lim, "Effects of Reuse on Quality, Productivity, and Economics," *IEEE Softw.,* vol. 11, pp. 23-30, 1994.

[13] M. D. McIlroy*, et al.*, "Mass produced software components," *Software Engineering Concepts and Techniques,* pp. 88–98, 1969.

[14] R. Seepold and A. Kunzmann, *Reuse techniques for VLSI design*. Netherlands: Springer 1999.

[15] L. Latour*, et al.*, "Descriptive and predictive aspects of the 3Cs model: SETA1 working group summary," 1991, pp. 9-17.

[16] W. B. Frakes and R. Baeza-Yates, *Information retrieval: data structures and algorithms*, 2nd ed. vol. 77. Englewood Cliffs, NJ: Prentice-Hall. , 1998.

[17] B. Boehm*, et al.*, "Software development cost estimation approaches — A survey," *Annals of Software Engineering,* vol. 10, pp. 177-205, 2000.

[18] B. W. Boehm, *Software engineering economics*. Upper Saddle River, NJ: Prentice-Hall, 1981.

[19] B. Boehm*, et al.*, "Cost estimation with COCOMO II," ed: Upper Saddle River, NJ: Prentice-Hall, 2000.

[20] L. H. Putnam and W. Myers, *Measures for excellence*: Yourdon Press, 1992.

[21] R. Jensen, "An improved macrolevel software development resource estimation model," in *5th ISPA Conference*, 1983, pp. 88–92.

[22] R. W. Selby, "Enabling reuse-based software development of large-scale systems," *IEEE Transactions on Software Engineering,* vol. 31, pp. 495-510, 2005.

[23] T. Tan*, et al.*, "Productivity trends in incremental and iterative software development," in *ESEM '09 Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement* Lake Buena Vista, Florida, USA, 2009, pp. 1-10.

[24] A. Gupta, "The profile of software changes in reused vs. non-reused industrial software systems," Doctoral Thesis, NTNU, Singapore, 2009.

[25] N. E. Fenton and M. Neil, "Software metrics: roadmap," presented at the Proceedings of the Conference on The Future of Software Engineering, Limerick, Ireland, 2000.

[26] M. Dinsoreanu and I. Ignat, "A Pragmatic Analysis Model for Software Reuse," in *Software Engineering Research, Management and Applications 2009*. vol. 253, R. Lee and N. Ishii, Eds., ed: Springer Berlin / Heidelberg, 2009, pp. 217-227.

[27] I. Herraiz and A. E. Hassan, "Beyond Lines of Code: Do We Need More Complexity Metrics?," in *Making Software: What Really Works, and Why We Believe It*, A. Oram and G. Wilson, Eds., 1st ed Sebastapol, CA: O' Reilly Media, Inc. , 2010, pp. 125-141.

[28] T. J. McCabe, "A Complexity Measure," *IEEE Transactions on Software Engineering,* vol. SE-2, pp. 308-320, 1976.

[29] S. H. Kan, *Metrics and models in software quality engineering*. India: Pearson Education India, 2003.

[30] J. Graylin*, et al.*, "Cyclomatic Complexity and Lines of Code: Empirical Evidence of a Stable Linear Relationship," *Journal of Software Engineering and Applications,* p. 137, 2009.

[31] R. P. L. Buse and W. R. Weimer, "Learning a Metric for Code Readability," *IEEE Transactions on Software Engineering,* vol. 36, pp. 546-558, 2010.

[32] J. E. Gaffney, "Estimating the Number of Faults in Code," *IEEE Transactions on Software Engineering,* vol. SE-10, pp. 459-464, 1984.

[33] M. S. Krishnan and M. I. Kellner, "Measuring process consistency: implications for reducing software defects," *IEEE Transactions on Software Engineering,* vol. 25, pp. 800-815, 1999.

[34] W. B. Frakes and T. P. Pole , "An empirical study of representation methods for reusable software components," *IEEE Transactions on Software Engineering,* vol. 20, pp. 617-630, 1994.

[35] J. Cohen, *Statistical power analysis for the behavioral sciences*, 2nd ed. Hillsdale, NJ: Lawrence Erlbaum Associates, Inc., 1988.

Choosing licenses in free open source software

Ioannis E. Foukarakis
Department of Telecommunications
Science and Technology
University of Peloponnese
Tripolis, Greece

Georgia M. Kapitsaki
Department of Computer Science
University of Cyprus
Nicosia, Cyprus

Nikolaos D. Tselikas
Department of Telecommunications
Science and Technology
University of Peloponnese
Tripolis, Greece

***Abstract*—Free/Libre/Open Source Software (FLOSS) has been lately on the focus of the software community in the attempt to provide wide access to software resources and promote their distribution. Open software resources may be linked directly or indirectly with more than one open source licenses. Due to license heterogeneity, incompatibilities may arise in such dependencies. For this reason, adequate support for decision making in license use is an important issue. Such systems can assist in determining the right licenses and avoid potential violations. We tackle the above aspects by giving an overview of the areas of license extraction from software resources, license information storing and license-related reasoning activities. We discuss about several attempts in these areas and present an initial prototype assistance system employing such techniques.**

Keywords- data dependencies; licensing; open source software

I. Introduction

Free/Libre/Open Source Software (FLOSS) [1] has influenced modern software engineering processes, allowing individuals to use software for free and software engineers to incorporate third party software libraries in their software products and implementations. The terms, under which the software has become available and is provided for use to the community, are captured in the licenses that accompany open source software. Licenses correspond to the characteristics of the intellectual property and usually express the rights and the obligations of the *copyright owner* and the one using the license (referred to as the *licensee*). Many different licenses have appeared: GNU General Public License (GPL), Apache License, BSD License, MIT License to name a few. Licenses fall into three main categories: permissive, weak copyleft or strong copyleft, showing different degrees of freedom in FLOSS use. The first category of permissive licenses states that licensed software can be adopted even in commercial software that is not distributed as open source. On the contrary, weak copyleft licenses demand that any changes in the licenses source code should be made available as open source software itself, whereas strong copyleft licenses mandate that any kind of further distribution be under open source principles.

Each license may have multiple versions making it difficult for developers to cope with incompatibilities that might exist due to the use of software libraries based on different licenses. Especially during the development phase, it is often the case that engineers include additional, and often redundant, dependencies in their code light-hearted, without examining possible licensing incompatibilities. As the number of components increases, so does the complexity of deciding which licenses can be applied to the final system, or of checking if violations exist among the terms defined in the licenses, leading to license incompatibility. Lindberg [2] describes license compatibility as a sim ilar approach with blood type compatibility: "*two blood types are compatible if donations from two different people can be used together.*" In the same way, licenses are compatible if software distributed under two different licenses can be used together in the same software product. Although no mature software for supporting decisions in license issues is available, many research activities and commercial efforts have recently appeared.

The current paper provides an overview of the aspects of license decisions by presenting the process of license compatibility detection that can be divided into the areas of license extraction information from software resources, license information storing, as well as modeling and reasoning actions. Some techniques are applied on the implementation of a prototype system extending a popular build tool. The rest of the paper is structured as follows. Section II briefly presents the steps of the license compatibilities process. Section III discusses works on extracting license information from existing components, whereas section IV is dedicated to the presentation of repository-based approaches. In section V ways for discovering licenses are presented. An initial prototype implementation of the compatibility detection is demonstrated in section VI. Finally, section VII concludes the paper.

II. The License Compatibility Process

Each software system is comprised of individual artifacts that may include source code, compiled programs, software libraries or other file types (i.e., images, multimedia files and data in general). An abstract model of the software includes the distinct artifacts and the way these are connected through the development process. For instance, a source code file may hold references to other source code files or libraries. In turn, the compiled version of the source code may have static or dynamic references to software libraries and so on.

The generic approach that can be followed for detecting license incompatibilities is depicted in Fig. 1. The first step towards making a decision for possible open source licenses is to identify the individual components of the software system. Afterwards, different *license extraction* techniques may be applied to detect the license of each artifact, varying from text matching to querying *license repositories*. In fact, the same

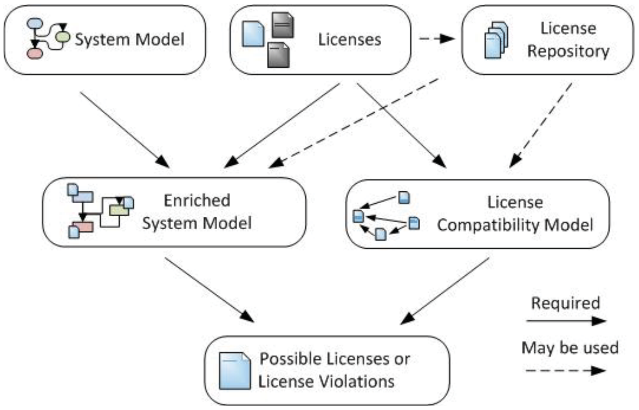


Figure 1. High level overview of detecting compatible licenses

license extraction techniques may be used to populate such repositories. By combining the system's model with license metadata, an enriched version of the system model can be derived, holding both information about the components that comprise the system and the license of each participating component. This enriched model can be seen as the license architecture of the software system, depicting artifact and license interconnections. The validity of these connections can be examined by checking whether two connected components have compatible licenses or not. In order to perform this task, the definition of compatibilities between licenses is required. This can be achieved by analyzing the licenses and modeling their compatibilities either as a whole or on individual permissions and restrictions. By combining the enriched system model with the *license compatibility model*, it is possible to detect permissible licensing schemas for the software system and/or possible licensing violations.

III. License Information Extraction

In FLOSS development, several methodologies and corresponding tools have been developed in an attempt to identify licenses from source code, software libraries in binary format, components or programs of different suppliers under FLOSS or proprietary licenses. A category of extraction methods is based on regular expressions that indentify the license of each file through string matching techniques. These patterns result from the analysis of the actual license text. Tools using this approach are Ohcount[1], OSLC (Open Source License Checker), and ASLA (Automated Software License Analyzer) [3]. According to this method, source files and files that might contain license information are first identified (i.e., commented parts of source files and whole files) and are then checked, in order to compare the content found against database files and conclude whether a specific license is present or not.

A similar approach is followed in LIDESC (Librock License Awareness System[2]), which is an awareness tool for preventing unintended violations of license terms of software that the developer writes or acquires. In LIDESC a license is modeled as a set of "license stamps" with each stamp corresponding to the entries encountered in the license text. It can be seen as a pair of a label and a string. The label defines a specific right or obligation in the license. The same label may be used in multiple licenses, whereas the string contains the license text that defines the corresponding right. By using text comparison, a set of license stamps is generated for each file, defined in two specific files for each license: a *.txt* file that contains the license text and a *.lh* file that uses the C language to define strings of symbolic terms, in order to describe the license.

An algorithm for automatic license identification from source code has been implemented in the Ninka tool [4]. The algorithm works by extracting the license statement from the file, which it breaks apart into textual sentences, and proceeds to find a match for each sentence. The list of matched sentences is analyzed to determine if it contains one or more licenses. This method is based on and requires a knowledge base with four sets of information: filtering keywords, sets of equivalence phrases, known sentence-token expressions and, license rules.

When focusing on binaries, the Binary Analysis Tool[3] (BAT) looks inside binary code, in order to find license compliance issues. BAT can detect bootloaders (such as Redboot, loadlin and uboot), open various compression archives (such as ZIP, RAR, tar, cpio and LZMA), search for Linux kernel and Busybox issues and identify dynamically linked libraries, while it reports the outcome in XML format. BAT uses symbol and string table comparisons to read binary code in firmware formats and compare it with source code, without undertaking reverse engineering actions. BAT can also compare the compiled version of the software under review with the corresponding source code, resulting in more accurate results. A different technique is based on automatic licenses' tracking of third-party artifacts [5]. Popular open-source build frameworks, i.e., Maven, Ivy and Gradle, describe artifacts and dependencies in terms of reusable declarative modules. Project Object Model (POM) files in Maven, as well as Descriptor files in Ivy and Gradle respectively, were designed to contain license information as part of the module metadata, giving the hope that the extraction of license information from module metadata can become fully automated. As we will describe, this is usually achieved by using an artifact repository.

IV. Licenses and Repositories

When talking about open source code repositories, many locations can be enumerated starting with SourceForge. Further repositories that aggregate project content from other locations can be found in commercial (e.g., Black Duck Knowledge Base[4]) or non-commercial solutions (e.g., Swik[5], Ohloh). Another approach can be found in FLOSSmole [6], a central repository containing data and analyses about FLOSS projects collected and prepared in a decentralized manner.

[1] http://www.ohloh.net/p/ohcount
[2] http://www.mibsoftware.com/librock/lidesc/
[3] http://www.binaryanalysis.org
[4] http://www.blackducksoftware.com/knowledgebase
[5] http://swik.net/

The important aspect is to provide repositories with "intelligence" by creating repositories that store not only source code, but also sufficient information that would allow drawing useful conclusions as described by the Fact Extractor Support of Kenyon [7]. Repositories of this kind should be accompanied with metadata in a machine understandable format that can be parsed, analyzed and categorized fast and efficiently. Metadata may indicate useful parameters, such as file names and sizes, versions and modification permissions, and license-related details. When concentrating on license information and dependencies, input to the software repository can be provided by agents that extract license information from source code, libraries and text files. On the other hand, the stored information is maintained in a suitable format, in order to be provided to tools that can reason over the license details. Reasoning mechanisms extract results that usually lead to an implementation decision for a specific information system or point out conflicts related with license use. Another important parameter in software repositories is related to how different versions of the same entry are handled. Information about changes in license dependencies needs to be kept up to date, since this may lead to changes in implementation decisions.

One attempt for storing license information in a repository can be found in FOSSology [8]. The *study of FLOSS* was initiated at Hewlett Packard more than a decade ago, but its final version was released recently. FOSSology presents a list with all license types identified for each project stored in the repository, while it contains a tool for analyzing the files given as input by matching text against license templates. Currently FOSSology supports about 260 different licenses (considering different license versions). A match percentage, indicating the probability that the discovered license is indeed the one included in the file under examination, is also provided. This happens, since in practice the project authors may have changed words or phrases of the original license text. The presence of such a percentage is, thus, desirable in all license repositories, as is the presence of license taxonomies that can help towards identifying which license category each project belongs to. Taxonomies can be further used for identifying licenses that are outside the acceptable license set, defined in policies imposed inside an organization. An example, where matching of licenses against a set of management rules is supported, can be found in Artifactory[6], an internal repository that acts as a proxy between a build tool (e.g., Maven, Ant) and the outside world. Artifactory supports five statuses for licenses found: unapproved, approved, unknown, not found (when no license information exists) or neutral (used to indicate unapproved licenses when an approved license also exists).

Based on the existing approaches and generic repository structures, an example of what should form part of the license association metadata for a software entry is illustrated in Fig. 2. This information links projects with one or more licenses and should be available for each license encountered (apart from the sixth one, which provides statistical information). Note that the repository can either be centralized or distributed [9].

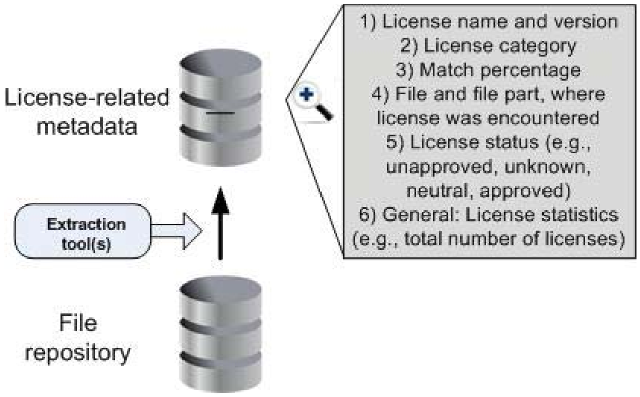

License-related metadata
Extraction tool(s)
File repository
1) License name and version
2) License category
3) Match percentage
4) File and file part, where license was encountered
5) License status (e.g., unapproved, unknown, neutral, approved)
6) General: License statistics (e.g., total number of licenses)

Figure 2. License metadata in a software repository

V. Discovering Possible Licenses

The previous techniques offer the means for extracting the license information and for storing this information in repositories. However, licenses and components cannot be viewed individually, but should be put in context with the rest of the entities that comprise the software system as part of the license compatibility process. This can be performed through a two-step procedure.

A. Modeling license relations

The first step lies in identifying the set of all possible licenses and modeling their relations. The whole set of open source licenses is quite large: 69 licenses have currently an approval from the Open Source Initiative[7] (OSI) and 88 from the Free Software Foundation[8] (FSF). Therefore, recent literature works focus on the most well-known and most widely used open source licenses, such as GNU GPL, GNU Lesser General Public License (LGPL) (version 2), Apache License (version 2.0), New BSD licenses, MIT license and Eclipse Public License. A practical approach that presents compatibility among some of the most popular open source licenses can be found in [10]. In general, license A can be considered to be *compatible* with license B, if software that contains components from both licenses can be licensed under license B. Under this line of though, all known licenses and their compatibilities can be represented through a directed graph *G(V, E)* as the one depicted in Fig. 3. Each node V of the graph represents one of the known licenses, while each edge E represents compatibility between the adjacent licenses. The direction of the edge represents which license should be used for software containing components with licenses from both nodes. This approach is based on manual interpretation of each license (through the corresponding license legal text) and provides general rules for reasoning on compatibility among licenses. A similar compatibility model is discussed in [11].

More detailed solutions, which integrate the notions of rights and obligations linked with a license, aim at modeling

[6] http://www.jfrog.com/

[7] http://www.opensource.org/

[8] http://www.fsf.org/

licenses in a machine understandable manner. Such approaches can assist in automated license compatibility checks. The aforementioned LIDESC uses the sets of "license stamps" to model licenses. If one license fulfills the requirements expressed in the license stamps of another license, then the two licenses can be considered compatible. In [12], the authors propose a metamodel for licenses refined using empirical data. A license is considered as a set of rights, where each right may refer to a set of obligations. A set of rules helps to extract license compatibilities. A similar approach is described in [13] in the framework of the Qualipso (Quality Platform for Open Source Software) project, where an ontology is used to model a set of well-known FLOSS licenses and their rights (8 FLOSS licenses were included). The ontology contains appropriate classes represented in a hierarchical model. The semantics of each license are modeled as a set of rights and conditions.

B. Inferring possible licenses

One of the most important issues in terms of deciding if two or more components are compatible lies on how the connection among them is modeled and under which conditions they can be connected. Software libraries may be linked statically or dynamically, whereas APIs may be used to hide client-server connections, middleware or protocol implementations. These different types of connections may affect specific rights or obligations defined within a license. For example, an email client needs to connect to an email server via the corresponding protocol. The license of the email server should not affect the possible licenses of the client. Additionally, when the email client needs to be tested, specific libraries can be used in the development of the test cases. Nevertheless, these libraries have no effect on the rights and obligations related to the distribution, since they are not integrated within the software but constitute external tools. Generally, license restrictions come to play, when software distribution needs to be addressed, but they should not affect actions performed in the framework of internal activities, such as testing. This does not exclude the presence of additional constraints imposed in policies within an organization that need to be considered. In order to tackle these issues and avoid conflicts, it is imperative to provide a model of the software system under question.

A simple approach for modeling the related licenses in a software system is to use the directed graph described previously. The possible licenses for the resulting software systems can be found among the ones that satisfy the compatibility criteria for the licenses of all system components. Consider for instance a project having two dependencies, one with license *C* and one with license *E* that may even belong to different license categories. Initially, all known licenses (*A* to *H* as in Fig. 3) can be regarded possible licenses. However, since the first dependency's license is *C* (Fig. 3(a)), the project can only have one of the licenses of the following set: *L1 = {C, F, H}*. In the same manner, since the second dependency's license is *E* (Fig. 3(b)), the project can only have one of the licenses among *L2 = {E, G, F, H}*. By using the intersection of the two license sets, it is concluded that the project is allowed to have one of the following licenses: *L1* ∩ *L2 = {F, H}*. If there is no information about the license of a dependency, the same process can be used recursively using transitive dependencies. Please note again that when organization policies on license use are present (e.g., use of strong copyleft licenses is prohibited), this simplified graph approach needs to be adapted in order to take into account the additional constraints.

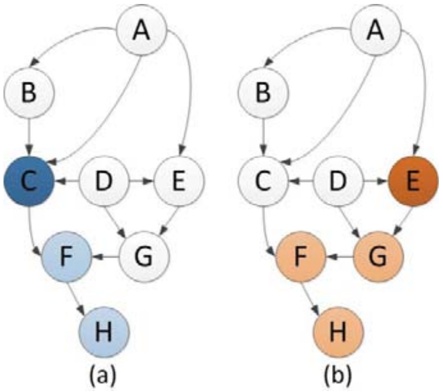


Figure 3. License compatibility graph

More detailed graph-based models for software systems can also be employed in this part of the process. Such models are useful in order to discover which rights and/or obligations may apply to the resulting software. Instead of checking for the possible licenses of the resulting system, all rights and obligations that result from using one of the license relation schemas described earlier are detected. In the Qualipso ontology [13], rules expressed in a legal rule language are applied to the model and possible licenses are detected automatically. Compatible licenses are found through the introduced reasoner that searches for the licenses that provide the required rights and/or belong to a specific class of the hierarchy.

VI. A Build Tool Case Study

The approaches presented are applicable to a large number of use cases. In order to evaluate the process of integrating the aforementioned techniques in a fully functional system, a prototype assistance system addressed mainly to software engineers that employ various components has been implemented. An interesting fact is that most of the above components are used for validating the result of the development process, and not for assisting during the actual implementation of the application. The prototype system has been implemented as part of a build tool, since build tools are widely employed in software engineering activities. By adding an extension to an existing build tool the system is able to support both stand-alone and team development, it can be used as part of continuous integration builds and can be easily incorporated in existing well known Integrated Development Environments (IDEs). In this section information on the design decisions and problems encountered during the development of the prototype are given.

Initially an appropriate build tool to support the process had to be selected. Since the community of Java software engineers is one of the largest, we decided to focus on Java-related technologies. Under the available Java-friendly solutions Maven[9] from Apache was selected. Maven's main advantage is

[9] maven.apache.org/

that holds a model of each software project internally. Using an XML file, Maven describes each project's model as a set of source code and third-party libraries (Java Archives or JARs). Moreover, it is able to detect transitive dependencies originating from the third-party libraries and include them in the project. These files are downloaded automatically from public or private repositories. Note that the model might include metadata for each dependency, such as the dependency's license name and text.

The detection of each dependency's license is perhaps the most difficult part in the process towards license decisions. As described, license discovery can be implemented either by searching each dependency's file(s) or by looking up from a repository. Although public repositories are available, the existing repositories are not designed for providing license information. Additionally, there's no guarantee that every JAR file contains license information. In order to evaluate the situation, a crawler that indexed a large number of JAR files in the central Maven repository was built. A list of the most common file names that include licensing information is presented in [4]. Out of the 31476 JAR files studied, exactly 14000 contained a file that might hold license information (approximately 44.5% of the whole set). The most commonly used word in the name of files that hold licensing information was the obvious suspect, i.e., "license", with 13601 of the JAR files containing at least one such file. The second name in order was "readme" with only 708 appearances. These numbers indicate that in most cases license detection can be performed without the need of a dedicated repository. However, there are cases, in which licensing information cannot be determined for a large number of dependencies. Consequently, although in the implemented case study a license repository is not necessary, it could offer a significant improvement in the results of the tool.

The final step in the case study was to create the enriched system model and extract possible licenses. The missing information for performing this task is the license compatibility model. In the implemented tool, the licenses and their compatibilities were captured in an XML format. Multiple models have been discussed in the current paper; however, in the current stage we opted for the approach of [10]. Although it does not focus on individual rights and obligations expressed in a license, it offers the required abstraction for understanding which licenses can be adopted and which not. The file can be modified, in order to introduce additional licenses or change the relations of existing ones. The XML representation is easily coupled with the simple graph approach captured in machine readable notation. Moreover, it can also offer a graphical representation of the system's individual components and the way their interconnections affect licenses allowing a custom XML parser to detect permissive combinations. Using this XML licenses dependencies file together with the set of licenses originating from the project's dependencies, as collected from the first phase of license detection, appropriate license decisions can be taken.

VII. Conclusions

In this paper licenses in free open source software have been discussed along with representative approaches that guide the license compatibility process. License identification deals primarily with the extraction of license information from existing source code and binary files, whereas software repositories can be exploited for assisting the storing and fast retrieval of license-related metadata linked to software products. License discovery is then feasible by modeling the relations among licenses and performing reasoning actions on these relations. These aspects were demonstrated through a prototype implementation for license identification in the Maven build tool. Through the presented study, it can be concluded that it is important for software engineers to detect the correct licenses and avoid integrating prohibited licenses or ones that cause violations in license dependency chains. This decision is vital especially at our times, where a wide variety of open source, already tested software is available at our fingertips. Through the use case demonstration it can be seen that such approaches can be combined with existing tools. We are currently working towards completing the prototype system with the aim to cover more file types and include intelligent reasoning capabilities towards the provision of a complete "license assistant" tool.

References

[1] K. W. Miller, J. Voas and T. Costello, "Free and Open Source Software," IT Professional , vol.12, no.6, pp.14-16, Nov.-Dec. 2010.

[2] V. Lindberg, "Intellectual Property and Open Source A Practical Guide to Protecting Code," O'Reilly Media, 2008.

[3] T. Tuunanen, J. Koskinen and T. Karkkainen, "Automated software license analysis," Automated Software Eng., vol. 16, no. 3-4, pp. 455-490, Dec. 2009.

[4] D. M. German, Y. Manabe and K. Inoue, "A sentence-matching method for automatic license identification of source code files," in Proc. of the IEEE/ACM int'l conf. on Automated software engineering, ACM Press, 2010.

[5] Y. Landman, "How to Use Continuous Integration to Protect Your Projects from Open-Source License Violations," http://weblogs.java.net/blog/yoavl/archive/2010/12/16/how-use-continuous-integration-protect-your-projects-open-source-licen, 2010.

[6] J. Howison, M. Conklin and K. Crowston, "FLOSSmole: A collaborative repository for FLOSS research data and analyses," Int'l Journal of Information Technology and Web Eng., vol. 1, no. 3, pp. 17-26, July 2006.

[7] J. Bevan, E .J. Whitehead Jr., S. Kim and M. Godfrey, "Facilitating Software Evolution Research with Kenyon," in Proc. of the 10th European software engineering conference held jointly with 13th ACM SIGSOFT int'l symposium on Foundations of software engineering, ACM Press, 2005.

[8] R. Gobeille, "The FOSSology project," in Proc. of the 2008 Int'l working conf. on Mining software repositories, ACM Press, 2008, pp. 47-50.

[9] M. Conklin, "Project Entity Matching across FLOSS Repositories," in Proc. of the 3rd int'l Conference on Open Source Systems, Springer-Verlag, vol. 234, 2007, pp. 45-57.

[10] D.A. Wheeler, "The Free-Libre/Open Source Software (FLOSS) License Slide," http://www.dwheeler.com/essays/floss-license-slide.pdf, 2007.

[11] D. M. German, and A. E. Hassan, "License integration patterns: Addressing license mismatches in component-based development," in Proc. of the IEEE 31st int'l Conference on Software Engineering, 2009, pp. 188-198.

[12] 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, vol. 11, no. 11, pp. 730-755, Nov. 2010.

[13] T. F. Gordon, "Report on a Prototype Decision Support System for OSS License Compatibility Issues," Qualipso (IST- FP6-IP-034763), Deliverable A1.D2.1.3, 2010.

A Unified Model for Server Usage and Operational Costs Based on User Profiles: An Industrial Evaluation

Johannes Pelto-Piri
Blekinge Institute of Technology
SE-371 79 Karlskrona, Sweden
Email: johannes.peltopiri@gmail.com

Peter Molin
Malvacom AB
Soft Center Fridhemsvägen 8
SE-372 25 Ronneby, Sweden
Email: peter.molin@malvacom.com

Richard Torkar
Blekinge Institute of Technology
SE-371 79 Karlskrona, Sweden
Email: richard.torkar@gmail.com

***Abstract*—Capacity planning is essential for providing good quality of service, for that reason we need to be able to predict the usage that the applications will impose on our servers. This paper presents a unified model that can predict the usage, hardware requirements and, ultimately, the operational costs. The goal of this study is to present a model for capacity planning. The model presented is developed and evaluated within the industry. The evaluation is used to analyze the possibilities of the proposed model. The models have been evaluated within a company using historical data that originates from production software. The evaluation was done by running the model against three applications and mapping the result to a selection of Amazon EC2 cloud instances. We then provided the same data to five developers and asked them which instance they would have chosen for the applications. Two of the developers suggested the same instance as the model, the second smallest on the scale. The remaining three developers chose the instance one step above the model's recommendation. The evaluation showed that the model is able to produce estimations that are comparable to expert opinions. The unified model in this paper can be used as a tool to estimate the usage, hardware requirements and the final cost of the servers. The model is easy to setup within a spreadsheet and contains parameters that are easy to obtain from access logs and various logging tools.**

I. Introduction

Mobile applications have gained a high market penetration [1] that is continuously growing. That, in combination with the rise of cloud computing, makes it easier than ever for small companies and independent developers to reach out to a large user population. For mobile applications it is more and more common to utilize context aware and multimedia services, which require more computational resources than their predecessors [2]. As cloud computing is expensive for applications with a heavy load and large data transfers [3], [4], small companies and independent developers need be able to make informed decisions about which infrastructure to use in order to provide cost-effective Quality of Service (QoS).

This study has been conducted at Malvacom AB, a growing startup in Sweden that develops a data synchronization service called mAppBridge. The goal of this study is to investigate how users and applications can be modeled and then derive the server usage for mobile applications and, consequently, set requirements for the infrastructure.

Capacity planning is the process in which we determine the capacity needed for our services in order to provide QoS. Literature has proposed many methods [5], [6], [7], [8], [9] for this task. Menascé and Almeida describes capacity planning as a series of steps where we identify the workload, forecast the performance and then do a cost analysis [5], in which it us up to us to implement and calibrate a workload and a performance model. However, Gunther [10] discusses the need for a simple to use capacity planning framework, arguing that the frameworks currently used are too complex. In this study we aim to create a unified model that can be easily implemented to provide an overview of applications' hardware requirements and final costs. Thus, the main focus of our study, and hence also for this paper are:

- Present a unified model that can be used to receive an initial estimate about the hardware requirements needed to sustain a certain size of user population (Section II).
- Show that the unified model is easy to calibrate and capable of modeling server usage, hardware requirements and costs (Section III).

II. A Unified Model

To be able to derive the cost for the applications, we will first model both the application and its users, and derive the traffic from those two entities. Secondly, we will also model the software's hardware requirement for that traffic. Finally, once we have the hardware requirements we will derive the operational costs.

We have created a unified model for this. We have created a User Model that models a user's profile, i.e. their availability. The User Model is then used by the Application Model that models the load imposed on the servers. Next, we use a Software Model that models how much hardware that the software will require under the load derived from the Application Model. The Hardware Model is then, in its turn, used to model the cost of the hardware requirements generated by the Software Model. An overview of the models can be seen

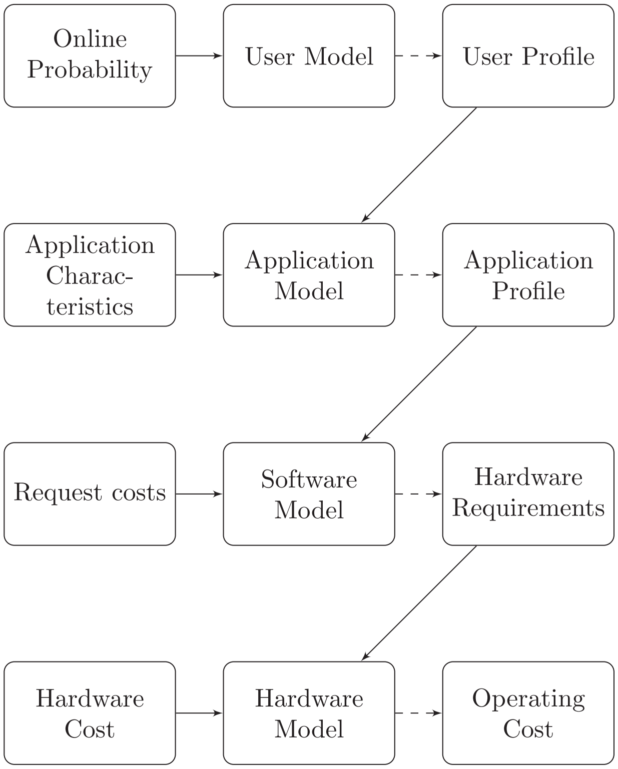


Fig. 1. Model overview.

in Fig. 1. (The boxes to the left represent the parameters to the model while the boxes on the right represents the outputs from the models.)

A. User Model

The user model consists of one parameter P that contains 24 values; each value represents an hour t of any given day i.e. $T = \{t_1, t_2, \ldots, t_{23}, t_{24}\}$, and describes the probability that a user will be online during t. The P parameter will later be used as an input to the Application Model.

$$P_t = \{P_1, P_2, \ldots, P_{23}, P_{24}\}$$

B. Application Model

The Application Model describes how many users the application has. It uses the User Model's probability function to determine the number of users online at a given moment. This is later used to derive the maximum concurrent users at a given hour and the total requests for one day. The Application Model takes the following parameters for modeling the application:

- **Data Pattern** (**DP_t**) The data activity for the hour t.
- **User Population** (**U_{pop}**) The user population of the application.

The DP parameter contains a vector of decimals that describes the usage profile for the application that we are modeling, $DP = \{DP_1, DP_2, \ldots, DP_{23}, DP_{24}\}$. Where 1 means that there is always something new to fetch from the server. This is used to model applications that rely heavily on push notifications, in which the server is responsible for initiating the communication between the server and the client. For example, an application that sends out four updates per hour is likely to have more traffic than an application that only sends out two. The U_{pop} parameter is a natural number that represents the number of users for our application. The output of the model is in the form of a vector R that contains 24 elements, one element for each hour that describes the number of requests in that hour. The equation for each element in the vector is described in (1).

$$R_t = U_{pop} \times DP_t \times P_t \tag{1}$$

$$R_{max} = max(R_t) \tag{2}$$

$$R_{sum} = sum(R_t) \tag{3}$$

Where R_{sum} (described in (3)) is the total number of predicted requests for one day and R_{max} (described in (2)) will be the maximum load that the servers will be expected to handle, hence the maximum concurrent users at any time in the system is not expected to exceed R_{max}.

C. Software Model

The Software Model models the amount of hardware resources that will be used by the application and its users. The Software Model requires the following parameters:

- **Request Size** (**R_{size}**)**:** The average size of each request in bytes.
- **Request Cost** (**R_{cost}**)**:** The cost of one request for the CPU in megahertz (MHz).
- **User Size** (**U_{size}**)**:** The size required per user.

As well as these parameters the Software Model also uses U_{pop} and R from the Application Model in order to calculate the requirements.

In short, the model needs to calculate the following requirements: CPU, storage, and bandwidth. We need to provide a CPU measurement strong enough for dealing with our peak loads. The equation for the CPU requirement can be found in (4).

The bandwidth requirement, on the other hand, is divided into two different sub-requirements: *a*) We need to derive the bandwidth that is needed to support the traffic peaks and, *b*) we want to know the daily load to know how much data we will handle per day. For the bandwidth required to we use R_{max} in (5) and for the total traffic per day we use R_{sum} in (6). The storage requirement can be found in (7).

$$CPU_{requirement} = R_{max} \times R_{cost} \tag{4}$$

$$Bandwidth_{peak} = R_{max} \times R_{size} \tag{5}$$

$$Bandwidth_{daily} = R_{size} \times R_{sum} \quad (6)$$

$$Storage_{requirement} = U_{pop} \times U_{size} \quad (7)$$

The models output are described in the formulas above. $CPU_{requirement}$ is defined in MHz, while outputs that are dealing with size (bytes) are defined in the same units as R_{size} and U_{size}. Thus defining the parameters in MB will yield the requirements in MB. Keep in mind that the Hardware Model uses GB as inputs, so by defining the size in MB the output will have to be converted to GB in a later stage.

D. Hardware Model

The main function of the Hardware Model is to calculate the cost of the hardware requirements derived from the Software Model. It calculates the cost for a one month period. The Hardware Model uses all but the $bandwidth_{peak}$ output from the Software Model as inputs. The Hardware Model uses the following parameters:

- **CPU Cost** : (C_{CPU}) The cost for the one MHz of CPU.
- **Storage Cost** : ($C_{storage}$) The cost of storing GB in the servers per month.
- **Bandwidth Cost** : ($C_{bandwidth}$) The cost of using GB of bandwidth per month.

The Hardware Model has the following outputs. The cost of fulfilling the CPU requirement (8), the network cost per month (9) and the storage cost (10).

$$CPU_{cost} = C_{CPU} \times R_{CPU} \quad (8)$$

$$Bandwidth_{cost} = C_{bandwidth} \times bandwidth_{daily} \times 30 \quad (9)$$

$$Storage_{cost} = C_{storage} \times Storage_{requirement} \quad (10)$$

Equation (8) is designed to derive the cost for the cost-performance of the modeled application on the given hardware. For applications hosted at Infrastructure as a Service provider (IaaS providers) such as Amazon or Rackspace we also pay for the bandwidth and storage of our applications. For the bandwidth cost described in (9) we start with calculating the daily cost and then converting it to a monthly basis. For the storage cost (described in (10)) it is common to charge a certain amount per GB for each month, hence we do not convert it to a monthly basis. We simply calculate the cost based on how many GB that will be required and the cost for those.

TABLE I
THE APPLICATIONS INCLUDED IN THE VALIDATION.

Application	U_{pop}	R_{size}
1	15140	351839
2	1334	2210956
3	599	676276

III. EVALUATION

This section contains an evaluation of the models. The evaluation was conducted in Malvacom AB; the parameters were estimated using historical data. The validation was done by modeling three applications that are currently in production. The selection of the applications was done by looking at the user population for the applications. We wanted to compare one small application, one medium-sized and one large from the available dataset in order to compare how the attributes such as the user population and the users availability impacts the actual cost. The selected applications can be seen in Table I. Application 1 is an application used for walking and social interactions while Application 2 and 3 is are social networking applications. The results were then analyzed and compared against developer opinions to see how the model performs compares against expert opinion.

In the first step of the evaluation we describe how we estimated the parameters. During the estimation we had access to data that allowed us to estimate the user's availability and various attributes about the requests such as request size. For the last step, when looking at the cost for deploying, we chose Amazon EC2 due their clear documentation regarding the pricing (obviously any other provider can be used as long as they provide clear data regarding pricing).

A. Parameter Estimation

For the parameter P in the User Model we summarized the traffic for mAppBridge, we grouped each request per hour and took the average number of requests based on the number of days we had collected traffic for and the average number of unique users per day. We chose to treat all requests as a virtual user, which results in 100% user activity when the number of requests was equal to the total population. Our extracted user pattern is described in the matrix below. The population and the average request size for each of the applications are described in Table I.

$$P = \begin{pmatrix} 0,113 & 0,091 & 0,087 & 0,135 & 0,260 & 0,383 \\ 0,530 & 0,613 & 0,626 & 0,652 & 0,660 & 0,675 \\ 0,679 & 0,711 & 0,695 & 0,651 & 0,624 & 0,658 \\ 0,645 & 0,655 & 0,597 & 0,439 & 0,266 & 0,165 \end{pmatrix}$$

Unfortunately, we did not have any good source regarding the data activity for the application so we assume that there were always something new for the user to fetch and, hence, we set the activity to 100% for each hour, i.e. $DP = \{t_1 = 1, t_2 = 1, \ldots, t_{24} = 1\}$.

TABLE II
THE PARAMETERS FOR THE HARDWARE MODEL.

Parameter:	C_{CPU}.	$C_{Storage}$.	$C_{Bandwidth}$.
Cost	$0.051.	$0.383.	$0.01.

For the request cost (R_{cost}) we have used (11). Unfortunately we did not have access to any historical data regarding the CPU utilization, therefore we made an assumption. We assume that we are currently maintaining one server with a single CPU with a processor speed on 2.2 GHz with an average load on 50% for a benchmark with 30,000 requests. We assume that the requests that the server is currently receiving is equivalent in terms of CPU utilization as the request from application we are trying to model. The estimation of the request cost parameter can be found in (12).

$$R_{cost} = \frac{\text{Processor speed (MHz)} \times \text{processor utilization}}{\text{Observed Requests}} \tag{11}$$

$$R_{cost} = \frac{22000 \times 0.5}{30000} = 0.04 \text{ MHz} \tag{12}$$

For the users we assume that our application will take up 5 megabytes of space per user in personal data, i.e. $U_{\text{size}} = 5\text{MB}$

For the Hardware Model we must estimate the cost for the processor, bandwidth and storage. We choose to model the cost for a single small instance on Amazon EC2. At the time of writing a small instance on Amazon with one EC2 unit, which is equivalent to 1.2 gigahertz and includes 160 gigabytes of storage, has a monthly cost of $61.2. Traffic that leaves amazon costs $0.01 per gigabyte. The cost for the storage and the CPU is done by dividing the processor speed and the storage with the cost. The parameters can be seen in Table II.

B. Results

The first step is to calculate the number of requests for each of the applications. This is done with (1). Table III shows the summary of the requests for each of the applications while the full request pattern for each of the application can be found in the below matrices.

$$R_1 = \begin{pmatrix} 1704 & 1376 & 1312 & 2045 & 3941 & 5793 \\ 8018 & 9283 & 9471 & 9866 & 9997 & 10219 \\ 10275 & 10767 & 10524 & 9861 & 9441 & 9963 \\ 9763 & 9913 & 9032 & 6645 & 4023 & 2493 \end{pmatrix}$$

$$R_2 = \begin{pmatrix} 150 & 121 & 116 & 180 & 347 & 510 \\ 706 & 818 & 834 & 869 & 881 & 900 \\ 905 & 949 & 927 & 869 & 832 & 878 \\ 860 & 873 & 796 & 585 & 354 & 220 \end{pmatrix}$$

$$R_3 = \begin{pmatrix} 67 & 54 & 52 & 81 & 156 & 229 \\ 317 & 367 & 375 & 390 & 396 & 404 \\ 407 & 426 & 416 & 390 & 374 & 394 \\ 386 & 392 & 357 & 263 & 159 & 99 \end{pmatrix}$$

TABLE III
THE PREDICTED REQUEST PEAK AND DAILY LOAD FOR EACH OF THE APPLICATIONS.

Application	R_{sum}	R_{max}
1	174,018	10,767
2	15,333	949
3	6,885	426
Total	196,236	12,141

When we have the request pattern for each of the applications we can move on and estimate the hardware requirements with the Software Model. The hardware requirements for the applications can be seen in Table IV. The equations used to calculate the requirements are described in Section II-C. The requirements that were in bytes have been converted to GB for convenience.

The last step of the execution is to derive the final cost for the hardware. We do this for each of the applications. The results can be found in Table V.

C. Analysis

When we analyze the results it is important to keep in mind which service provider we are using. In this case we have mapped the results to Amazon EC2 instances. By looking at the requirements (shown in Table IV) we can see that the requirements fit a small instance. We asked five developers to choose an instance for the applications based on the data given in Section III-A. Two of the developers thought that a small instance would suffice while three developers would have chosen a larger instance. This shows that the output of the model falls within the range of the developers' suggestions.

As can be seen, each application is modeled separately; this has resulted in different hardware requirements for each of the applications. Meaning, that in theory, this output can be used to distribute the applications across a set of servers so that we are utilizing each server in an optimal way, possibly leading to fewer resources allocated as we are keeping waste to a minimum. Also, by modeling existing applications on existing servers we can use the model to determine if we can fit a specified application to it, as we were predicting the CPU utilization of each application. This model also allows us to quickly see how the cost and hardware requirements would change if we change our service provider, as we can use one set of Software and Hardware Models for different service providers to compare the cost between them.

It should be noted that in order to gain more accurate results, more data is needed to calibrate the parameters. The parameters for these models are quite trivial in nature and there exists a wide array of software logging tools that can be installed to gather all the necessary data.

IV. VALIDITY THREATS

The goal of the evaluation was to see if the model is suitable for usage. We evaluated this by looking at how the model's final prediction was positioned in comparison with experts. We only compared against five experts, and we do not know how

TABLE IV
THE ESTIMATED HARDWARE REQUIREMENTS FOR EACH OF THE APPLICATIONS.

Application	$CPU_{requirement}$ (MHz)	$Bandwidth_{peak}$	$Bandwidth_{daily}$	$Storage_{requirement}$
1	394.779	3.528	57.022	75.70
2	34.784	1.953	31.572	6.67
3	15.619	0.268	4.339	2.995
Total	445.183	5.750	92.933	85.365

TABLE V
THE FINAL COST FOR DEPLOYING THE APPLICATIONS.

Final Cost

Application	CPU_{cost} (\$)	$Bandwidth_{cost}$ (\$)	$Storage_{cost}$ (\$)	Total
1	20.134	17.106	28.955	66.195
2	1.774	9.427	2.551	13.797
3	0.979	1.302	1.146	3.244
Total ($)	22.704	27.880	32.652	83.236

the comparison would have fared with more people. Also, the evaluation was targeted against mobile applications, furthering limiting the comparison.

The data at our disposal was able to give us the total number of unique users for the application, the user's activity pattern and the average size for each request. We were, however, unable to use the data to estimate parameters such as the user size (US) and various attributes in the Software Model such as the request cost.

V. Limitations

There are several limitations with this model. As for now the model does not help us to predict the QoS directly, as it simply models the hardware requirements and their cost. Also, the server's storage and network costs are not that accurate when it comes to the final cost since many service providers use predefined templates for the hardware where a certain amount of storage is already included in the price.

The requirements modeled by the Hardware Model does not take any kind of growth into consideration. This is a limitation when looking at the storage requirement. If we were modeling a data intensive application, in which the users are often uploading/downloading new content the storage will grow and hence the required cost for it.

In order to estimate the parameters in the best possible manner access to historical data is required. Also, different applications may target different users and have different user profiles as well, so in order for the historical data to be useful it must contain data from a similar application type.

The CPU requirement are modeled after R_{max}, the maximum numbers of users expected within one hour. This approach is a bit crude, a more realistic approach would be to model the arrival rate within each hour using a suitable probability distribution and then derive and use the mean or maximum arrival rate as R_{max}.

VI. Conclusions

In this study we have presented a unified model that can be used to predict the workload, hardware requirements and operational costs for a server infrastructure. The models are designed with simplicity in mind and they are easily implemented in a spreadsheet making the results easy to share.

Given a simple calibration of the model, the results are comparable to expert opinions. The result can also be used to compare the cost of service providers, and distribute applications based on their hardware requirements. In the end, these models can be used to help us make a more educated guess when we are performing capacity planning in order to assure cost-effective QoS.

We have conducted a static evaluation of the model. The evaluation has been focused on determining what the impact the model can have. Our future work will be focused on: *a*) Addressing some of the issues as described in Section V and also *b*) investigate the possibilities of adding an easy to use QoS model and finally *c*) validate the models from a business perspective

References

[1] M. Meeker, J. Dawson, J. Lu, B. Lu, R. Ji, S. Devitt, S. Flannery, N. Delfas, and M. Schneider, "The Mobile Internet Report," 2009.

[2] C. Canali, M. Colajanni, and R. Lancellotti, "Performance Evolution of Mobile Web-Based Services," *IEEE Internet Computing*, vol. 13, no. 2, pp. 60–68, 2009.

[3] B. C. Tak, B. Urgaonkar, and A. Sivasubramaniam, "To move or not to move: The economics of cloud computing," in *Third USENIX Workshop on Hot Topics in Cloud Computing (HOTCLOUD 2011)*, 2011, pp. 1–5.

[4] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the Clouds : A Berkeley View of Cloud Computing Cloud Computing," EECS Department, University of California, Berkeley, Tech. Rep., 2009.

[5] D. A. Menascé and V. A. F. Almeida, *Capacity Planning for Web Services: metrics, models and methods*. Prentice Hall, Upper Saddle River, 2001.

[6] M. Koorsse, L. Cowley, and A. Calitz, "Network Application Performance Modelling," in *Southern African Networks and Applications Conference*, vol. 27, no. 0, 2004.

[7] R. Lopes, F. Brasileiro, and P. D. Maciel, "Business-driven capacity planning of a cloud-based it infrastructure for the execution of Web applications," *2010 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW)*, pp. 1–8, 2010.

[8] R. Garg, H. Saran, and R. Randhawa, "A SLA framework for QoS provisioning and dynamic capacity allocation," in *Quality of Service, 2002. Tenth IEEE Internatinoal Workshop on*, 2002, pp. 129–137.

[9] J. Allspaw, *The Art of Capacity Planning: Scaling Web Resources*. O'Reilly Media,, 2008.

[10] N. Gunther, “Hit-and-run tactics enable guerrilla capacity planning,” *IT professional*, vol. 4, no. 4, pp. 40–46, 2002.

A Model-centric Approach for the Integration of Software Analysis Methods

Xiangping Chen[1,2,3,4], Jiaxi Chen[3,5], Zibin Zhao[3,6], Lingshuang Shao[7]
[1] Institute of Advanced Technology, Sun Yat-sen University, Guangzhou, 510006, China
[2] Research Institute of Sun Yat-sen University in Shenzhen, Shenzhen, 518057, China
[3] National Engineering Research Center of Digital Life, Guangzhou, 510006, China
[4] Key Laboratory of High Confidence Software Technologies (Peking University), Ministry of Education, Beijing, 100871, China
[5] School of Software, Sun Yat-sen University, Guangzhou, 510006, China
[6] School of Information Science and Technology, Sun Yat-sen University, Guangzhou, 510006, China
[7] Computer School of Wuhan University, Wuhan, 430072, China
Chenxp8@mail.sysu.edu.cn, chenjiaxi2011@foxmail.com, zzbloving@qq.com, shaolsh@gmail.com

Abstract— **Software analysis technologies are widely used in software quality assessment and system property prediction. The application of analysis technologies costs extra learning effort and domain-specific skill, and requires integration of software analysis methods. Current works on the integration of analysis methods highlight the abstraction effect of model and mostly focus on the integration of software modeling and model-based analysis methods. However, integration of analysis methods for other software artifacts, such as source code analysis methods, is not considered. In addition, less attention has been paid to the integration of the analysis results back to the software model. In this paper, we proposed a model-centric approach for the integration of software analysis methods. Our approach starts from a high-level software model provided by user input or generated from other artifact. We build the input adaptation rules in a platform specific way. As a result, the analysis methods can be integrated with input specification according to the model and adapted to the required input automatically. Considering different user interaction requirements, our framework provides facilities for execution of analyzers implemented as Eclipse plug-in and analyzers with command line interface in the execution stage. The traceability information from model element, analysis input to output is recorded in input adaptation and execution stage, and used for the integration of analysis results. We implement the integration framework as an Eclipse plug-in, and integrate 13 analysis methods in our framework to show the usability of our approach.**

Keywords-integration;software analysis;model-centric

I. INTRODUCTION

With the increasing importance of quality requirements in software systems, software analysis technologies are widely used for software quality assessment and system property prediction [12]. Because an analysis method is usually developed for specific quality concern, the software system with multiple quality requirements may require several analysis methods. Considering the chan ging analysis requirements during software lifecycle, applications of analysis technologies cost a lot of learning effort and domain-specific skill.

The integration of analysis methods into a wider process basically requires three steps: the extraction of input required by the analysis; the con trol of analysis execution; and the integration of analysis results back into the core model [1]. To ease the use o f analysis methods, approaches and frameworks for integration of analysis methods are proposed [2,3,4,5,6,7]. These works highlight the abstraction effect of model during analysis for user understanding. However, most of these approaches are developed for model-based analysis methods without considering analysis methods for other software artifacts, such as source code analysis methods. The input adaptation and analyzer execution of analysis methods for other kinds of artifacts should be consi dered. In addition, most approaches focus on the construction of quality attribute model through model transformation [3,4] to ease the heavy modeling effort of model-based analysis methods. However, less attention has been paid to the integration of the analysis results back to the so ftware model [7]. Integrating analysis results of different formats in the model level involves the traceability from model, analysis input to analysis result during analysis.

In this paper, we proposed a model-centric approach for the integration of software analysis methods. Software models provide abstraction of software systems. They are usefu l for selecting analysis method, selecting analysis object, and understanding analysis result. Our approach starts from a high-level software model provided by user input or generated from other artifacts. We build the input adaptation rules in a platform specific way. As a result, the analysis methods can be integrated with input specification according to the model and adapted to the required input automatically. Considering different user interaction requirements, our framework provides facilities for execution of analyzers implemented as Eclipse plug-in and analyzers with command line interface in the execution stage. During input adaptation and execution stages, the traceability information from model element, analysis input to analysis results is recorded. With the traceability information, our approach integrates the analysis results back to the model. We implement the integration framework as an Eclipse plug-in, and integrate 13 analysis methods to show usability of our approach.

The remainder of the paper is organize d as follows: Section 2 gives a gen eral overview of our solution, which is further detailed in the Section 3. Section 4 presents the use of our approach to integrate software analysis methods. Finally, Section 5 discusses some related works before Section 6 concludes our work.

II. Approach Overview

Models are widely used to provide abstraction of the software systems during software lifecycle. Considering providing a high-level view during analysis, we proposed a model-centric approach for the integration of software analysis methods. There are two kinds of users: analysis method integrators and analysis method users. The integrators are analysis method or analyzer developers who are familiar with the analysis methods. They specify the integration definition. The analysis method users are end-users of analysis methods. They choose analysis method and analysis object according to their analysis task at hand. The approach overview is shown in Fig. 1.

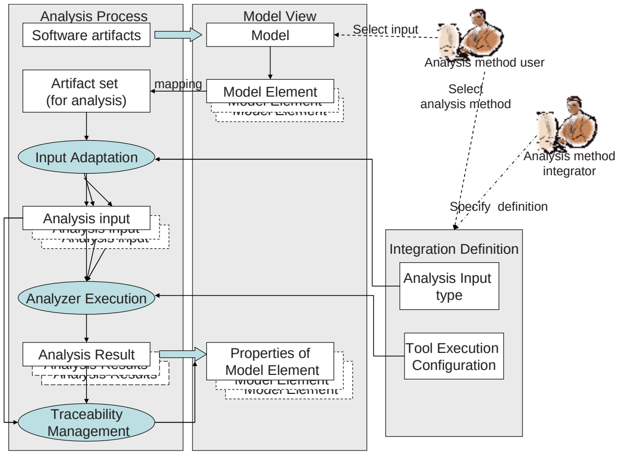


Figure 1. Approach overview

The software model is contained in the software artifacts provided by analysis method user, or generated from other artifacts. Model is used to guide the analysis users in choosing analysis input and using the analysis results.

According to the input, execution and output of the analysis methods, our approach includes three aspects: input adaptation, analyzer execution and traceability recording for analysis result integration. We build the input adaptation rules in a platform specific way. As a result, the analysis object in the model level can be mapped to an artifact set. With the specification of required input of the analysis method, the artifact set for analysis is adapted to the required analysis inputs automatically. During input adaptation stage, the traceability information from model element to analysis inputs is recorded. Considering different user interaction requirements, our framework provides facilities for execution of analyzers implemented as Eclipse plug-in and analyzers with command line interface. In the execution stage, our framework provides input for the analyzer, runs the analyzer, and acquires the analysis result. The input-output relationship is recorded during execution. With the traceability information from model element, analysis input to analysis results, our approach integrates the analysis result back to the model.

III. Model-centric Integration of Software Analysis Methods

A. Input Adaptation

Our approach starts from the generation of a software model. We choose the Unified Modeling Language (UML) [8] model as the high-level model. UML is a standardized general-purpose modeling language in the field of object-oriented software engineering. Most artifacts in object oriented development could be abstracted as elements or attributes of elements in the model.

When analyzing a software system, the analysis method user selects the whole project or part of the project according to the software model. The input adaptation includes two steps: first, the selected elements and attributes are mapped to a corresponding software artifact set. Because the arrangement and format of software artifacts rely on their development platforms, we build the mapping rules from UML model to software artifacts in a platform specific way. Secondly, the artifact set is adapted to proper inputs for the analysis method. Because an analysis method has its input format considering minimizing the required input or widely used input format, there may be mismatch between the artifact set and the required input of the analysis method. The adaptation may generate one or more inputs for analyzer execution. The adaptation is in a model-centric way with platform specific configuration and pre-define input requirement.

The platform specific adaptation rules are saved as pre-defined rules in our integration framework. We build the mapping rules from UML class diagram to Java project in Eclipse platform. Class diagram is the most important and most widely used structural view. Java is an object oriented programming language. The artifacts in a Java project can be abstracted as elements or attributes in the UML class diagram. In the Eclipse platform, the resources in a Java project are arranged using a tree structure called JavaModel. The key elements in the Java model are:

IJavaProject: represents a Java Project. It contains IPackageFragmentRoots as child nodes.

IPackageFragmentRoot: holds source files or binary files, can be a folder or a library (zip / jar file).

IPackageFragment: represents a single package. It contains ICompilationUnits or IClassFiles, depending on the files in the IPackageFragmentRoot. IPackageFragments are not organized as parent-children.

ICompilationUnit: represents a Java source file. Its children are IImportDeclaration, IType, IField, IMethod and so on.

IClassFile: represents a compiled (binary) file.

IType: represents either a source type inside a compilation unit, or a binary type inside a class file.

The mapping rules are defined based on rules for reversing design views [11], as shown in Table 1. We can see that the rules for project and package elements are simply one-to-one mapping. The mapping rules for class/interface elements consider the implementation in a Java file that: (1) a Java file may contain only one class/interface definition; (2) a Java file may contain several class/interface definitions; (3) a class may contain inner class definitions. For the mapping result of the element IType, the IType can be extracted and used to form a new ICompilationUnit which contains this IType as the only element. An artifacts of the type I ImportDeclaration, IField, IInitializer or IMethod is seldom used as analysis input because its definition is meaningful only in the context of an IType. For the mapping result of the type IField or IMethod, we extract its parent element which is of IType as the analysis artifact.

TABLE I. THE MAPPING RULES

UML Element	Pre-condition	Mapping type	Eclipse artifact
Project	--	1-1	IJavaProject
Package	--	1-1	IPackageFragment
Class/ Interface	If the ICompilationUnit contains only one IType which is of the same name	1-1	ICompilationUnit
	If the ICompilationUnit contains more than one IType, or the IType is child of another IType	1-1	IType
Function	--	1-1	IMethod
Variable	--	1-1	IField

In the integration stage, the integrator describes the required input of analysis method. We provide option list based on the artifact formats used in a Java application. If there is mismatch between the artifact set and the analysis input, we choose the analysis input which is the maximum subset of the artifact set compared to other validate analysis inputs included in the artifact set. Because artifacts are arranged as a tree structure, the input is adapted by a breadth-first search of analysis input from the root of the tree.

B. Analyzer Execution

In the input adaptation stage, analysis inputs are generated according to the input requirements. In the analyzer execution stage, the analysis task includes providing input for the analyzer, running the analyzer, and acquiring the analysis results.

There are two kinds of analysis execution requirements: automatic analysis and analysis with user interaction. Generally, the goal of analysis is to acquire the analysis results for use. This kind of analysis requires analyzing the input without user interaction. However, with the increasing analysis complexity, some analysis methods may require user configuration during analysis based on some intermediate results. This kind of analysis methods are usually implemented with graphical user interface so as to help understanding the intermediate and final results. Considering these two kinds of analysis requirements, our framework provides infrastructures for integration of analyzer with command line interface and analyzer implemented as Eclipse plug-in, respectively.

Analysis tools with command line interface are featured by its structured commands and black-box-like execution. The tool can be executed using command following pre-defined structure. The command structure usually includes the command name with several parameters, such as input and output. In addition, some more parameters may be needed to drive the analyzer in different modes.

For the integration of this kind of analyzer, we provide #INPUT# and #OUTPUT# to stand for the input and output in the command string. The analysis integrator can define the command structure using these parameters. To drive the analyzer, our framework generates a concrete command with the input and the output of certain analysis task. The #INPUT# will be replaced by the path of the artifact found in the input adaptation. The output path can be user-defined or using default value generated by our framework. In some cases, we find that the analyzer's command structure does not include the parameter for analysis output and its analysis results are returned in standard output stream which is printed in texture terminal. In this case, our framework saves the output as a file in the output file path through output redirection. Take the integration of an analysis method JDepend [9] as an example. Its command structure is defined as "java jdepend.xmlui.JDepend -file #OUTPUT# #INPUT#". When running the analyzer, our framework generates a command "java jdepend.xmlui.JDepend -file C:\result C:\input" if the input path is "C:\input" and the result path is "C:\result".

For an analyzer implemented as Eclipse plug-in, it usually provides graphical user interface and supports user interaction during analysis. A plug-in has a unique identifier registered in Eclipse plug-in registry. We assume that an analysis method in the plug-in can accept command triggered by end-user from an item in the user interface. Considering the Eclipse's plug-in mechanism, the command activating an analysis method is called an action. A plug-in may include several analysis methods and provide the invocation interfaces of different analysis methods through different actions. It should be noted that not all the actions provide analysis service.

The definition for analyzer execution includes the plug-in identifier, the extension type and the action class name. The integration framework requires the installation of plug-in analyzer before using the analysis method. When an analysis method is started, our framework finds the plug-in according to its unique identifier. And then, our framework instantiates an extension object of the extension type to run the action. Because the Eclipse platform loads all its plug-ins in a lazy manner, the integrated analysis methods will not be instantiated and activated until it is used. It means that the integrated plug-ins will not affect the performance of Eclipse platform.

When execution of an analysis method is finished, the relationship between user input in UML model and analysis result is recorded and stored in an XML file. The traceability information is provided for integration of analysis results.

C. Analysis Result Integration based on Traceability Information

Analysis is the process of breaking a complex topic or substance into smaller parts to gain a better understanding of it. In an analysis method, its input is the analysis object, and the analysis result is the property of the analysis object. Because the analysis input can be abstracted as elements in the software model, the analysis result can be abstracted as the property of an element or a set of elements. The goal of analysis result integration is to build the relationship between the software model and the analysis results. The integration is based on the traceability information recorded during analysis process.

The traceability information from the analysis object selected in the UML model, the artifact set, the analysis inputs and analysis results is kept by the integration framework during input adaptation and runtime execution stage. Relationship between UML model and the input is decided by the platform-specific rules and the required input format of the analysis method. During analyzer executeion, relationship is recorded into an XML file. After these two steps, the relationship between UML model and analyzer output can be deduced. The traceability recording process is shown in Fig. 2.

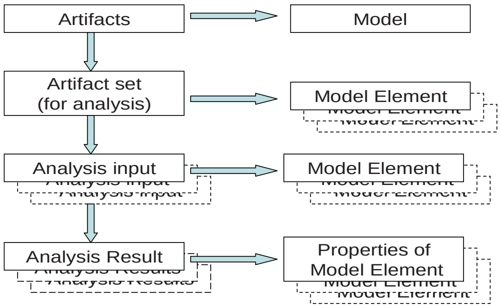


Figure 2. Traceability recording

With the traceability information recorded, the UML model will be extended with the analysis results as attributes of the analysis objects. To display the integrated analysis results, we provide a friendly user interface to show relationship between UML models and analysis results. In the UML diagrams, the extended properties will be linked to the analysis results.

IV. CASE STUDIES

We implement the supporting tool of our framework as an Eclipse plug-in. Using the tool, we have already integrated 13 analysis methods. We first introduce the integrated analysis methods, and then use an analysis method JDepend[9] as an example to introduce how to integrate an analysis method and how to use an integrated analysis method in our framework.

A. The Integrated Analysis Methods

We have already integrated 13 analysis methods in our framework. The integrated analysis methods are shown in Table 2. We assume that the analysis method integrators are analysis method developers who can provide detail information about the analysis methods. As a result, we choose analysis methods with open source analyzers, so as to provide detail integration information.

In the table, we can see that an analyzer may implement one or more analysis methods. An analysis method is a relatively standalone functional unit providing analysis service in the analyzer. There may be several analysis methods in its inner structure. An analysis method has its specific input format requirement. However, in the table, we combined the analysis methods with the same invocation interface but different input requirements as one analysis method for space limit.

TABLE II. INTEGRATED ANALYSIS METHODS

Analysis method	Analyzer	Input Format	Analyzer implementation
JDepend	JDepend [9]	IPackageFragmentRoot/ IPackageFragment	Eclipse plug-in/ Command line
Classycle	Classycle [13]	IJavaProject	Eclipse plug-in/ Command line
Code Analysis	CodePro Analytix [14]	IPackageFragmentRoot/ IPackageFragment/ ICompilationUnit	Eclipse plug-in
Similar Code Analysis	CodePro Analytix	IPackageFragmentRoot/ IPackageFragment/ ICompilationUnit	Eclipse plug-in
Metrics	CodePro Analytix	IPackageFragmentRoot/ IPackageFragment/ ICompilationUnit	Eclipse plug-in
Code Coverage	CodePro Analytix	IPackageFragmentRoot/ IPackageFragment/ ICompilationUnit	Eclipse plug-in
Dependency Analysis	CodePro Analytix	IPackageFragmentRoot/ IPackageFragment/ ICompilationUnit	Eclipse plug-in
Findbugs	Findbugs [15]	IJavaProject/ IPackageFragmentRoot/ IPackageFragment/ ICompilationUnit	Eclipse plug-in
Checkstyle	Checkstyle [16]	IJavaProject/ IPackageFragmentRoot/ IPackageFragment/ ICompilationUnit	Eclipse plug-in /Command line
PMD	PMD[17]	IPackageFragmentRoot/ ICompilationUnit	Command line
Yasca	Yasca [18]	IPackageFragmentRoot/ ICompilationUnit	Command line
JLint	JLint [19]	IPackageFragmentRoot/ ICompilationUnit	Command line
JavaNCSS	JavaNCSS [20]	IPackageFragmentRoot/ ICompilationUnit	Command line

B. Integrating the JDepend Analysis Method

JDepend[9] is an open source analysis tool for Java application. Its analysis result is design quality metrics. The metrics include number of classes and interfaces, afferent couplings, efferent couplings, abstractness, etc. It has two versions of implementations and could be run as Eclipse plug-in or from command line interface. In this section, we use JDepend as an example to introduce how to integrate analysis method and use integrated analysis method in our framework.

(1) Integrating an analysis method

The JDepend integrator provides the integration definition. The elements of integration definition are shown in Table 3. The elements *analysis method plug-in identifier*, *type of extension point*, and *plug-in action class* are used for integrating analyzer implemented as Eclipse plug-in. The element *command structure* is used for integrating analyzer with command line interface. It should be noted that the elements for integrating analyzers are optional. An analysis method can be integrated with any kind of analyzer implementations.

TABLE III. THE INTEGRATION DEFINITION

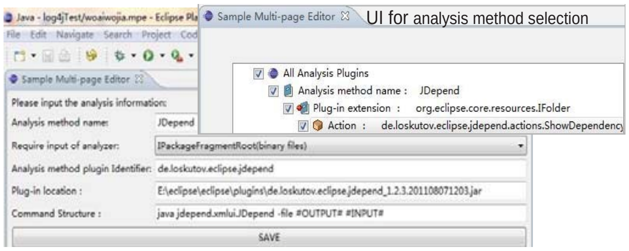
Element Name	Description
Analysis Method Name	The name of analysis method being integrated
Required input of analyzer	The input format accepted by the analyzer. A list of supported formats are provided.

Analysis method plug-in identifier	Unique identifier of the plug -in in Eclipse platform
Type of extension point	The type of extension point associated with the action
Plug-in action class	The class for invocation of an action
Command structure	The abstracted structure for generating concrete command

The integrator provides the integration definition in the user interface, as s hown in Fig. 3. The analysis method name is “JDepend”. Its required input is of the type “IPackageFragmentRoot(binary files)” for Java application.

To integrate JDepend implemented as Eclipse plug-in, our framework provides facility to ease the definition specification. If the integrator provides t he analyzer implementation, our framework interprets the plug-in configuration file and lists all the actions as analysis method candidates. The JDepend plug-in includes only one analysis method candidate, as shown in Fig. 3. The integrator can add the definition by selecting the JDepend analysis method or inputting the definition directly. Its extension type is “popupmenu”. The action class name is “de.loskutov.eclipse.jdepend.actions.ShowDependecy”.

To integrate JDepend analyzer with command line interface, we refer to its documents and find that its command follows the syntax “java j depend.xmlui.JDepend [-components <components>] [-file <output file>] <directory> [directory2 [directory 3] ...] ”. By abstracting its input and output, one of the running mode of JDepend command structure is defined as “java jdepend.xmlui.JDepend -file #OUTPUT# #INPUT#”.


Java - log4jTest/woaiwojia.mpe - Eclipse Pla
File Edit Navigate Search Project Cod
Sample Multi-page Editor
UI for analysis method selection
SAVE

Please input the analysis information:

Analysis method name: JDepend

Require input of analyzer: IPackageFragmentRoot(binary files)

Analysis method plugin Identifier: de.loskutov.eclipse.jdepend

Plug-in location : E:\eclipse\eclipse\plugins\de.loskutov.eclipse.jdepend_1.2.3.201108071203.jar

Command Structure : java jdepend.xmlui.JDepend -file #OUTPUT# #INPUT#

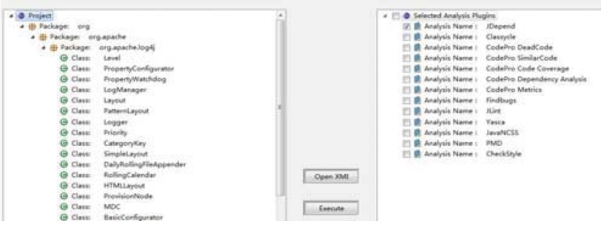
- ☑ All Analysis Plugins
  - ☑ Analysis method name : JDepend
    - ☑ Plug-in extension : org.eclipse.core.resources.IFolder
      - ☑ Action : de.loskutov.eclipse.jdepend.actions.ShowDependency

Figure 3. User interface for integration sepecification and selection of analysis method in Eclipse plug-in analyzer

(2) Using an integrated analysis method

The user plan to analyze the software project Log4j [10]. Log4j is an open source project of the Apache Software Foundation. Log4j enables logging at runtime without modifying the application binary so as to avoid a heavy performance cost for logging. The artifacts of Log4j contain about 20000 lines of source code and 188 classes.

Because the artifacts of Log4j do not contain a so ftware model, our framework first generates a UML model based on the source code of Log4j using a reverse tool EclipseUML [21]. We build a tree view of the UML model in our framework. The user selects the whole project as analysis object in the tree view of the Log4j U ML model. The integration definition of an analysis method is saved in an XML file. The integration definitions form a repository of reusable analysis methods. Based on the integration definitions, our framework provides a list of integrated analysis methods for the analysis method users. The user interface for selecting analysis object and analysis method is shown in Fig. 4.



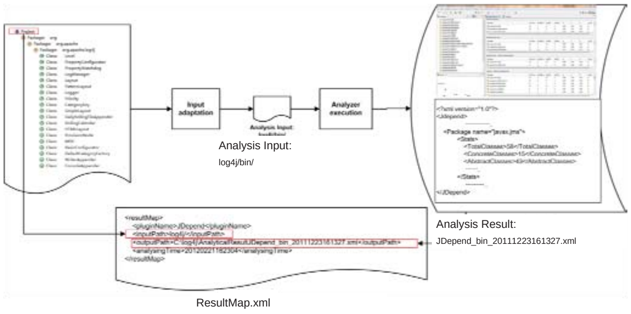
- Project
  - Package: org
    - Package: org.apache
      - Package: org.apache.log4j
        - Class: Level
        - Class: PropertyConfigurator
        - Class: PropertyWatchdog
        - Class: LogManager
        - Class: Layout
        - Class: PatternLayout
        - Class: Logger
        - Class: Priority
        - Class: CategoryKey
        - Class: SimpleLayout
        - Class: DailyRollingFileAppender
        - Class: RollingCalendar
        - Class: HTMLLayout
        - Class: ProvisionNode
        - Class: MDC
        - Class: BasicConfigurator

- □ Selected Analysis Plugins
  - ☑ Analysis Name : JDepend
  - □ Analysis Name : Classycle
  - □ Analysis Name : CodePro DeadCode
  - □ Analysis Name : CodePro SimilarCode
  - □ Analysis Name : CodePro Code Coverage
  - □ Analysis Name : CodePro Dependency Analysis
  - □ Analysis Name : CodePro Metrics
  - □ Analysis Name : Findbugs
  - □ Analysis Name : JLint
  - □ Analysis Name : Yasca
  - □ Analysis Name : JavaNCSS
  - □ Analysis Name : PMD
  - □ Analysis Name : CheckStyle

Figure 4. User interface for analysis object and analysis method selection

Our framework accepts UML model elements and maps the input to the artifact set IJavaProject of Log4j project. Because the required input of JDepend is an IPackageFragmentRoot folder holding binary files, our framework breadth-first searches IPackageFragmentRoot in the project and find two folders: “src” folder holding source code and “bi n” folder holding binary files. The “bin” f older is the proper analysis input. Our framework invokes the analyzers with the adapted inputs. For analysis method with one analyzer implementation, our framework runs the analyzer by default. For analysis method with two analyzer implementations, user can s elect to run one or two analyzer implementations.

The user selects to run two JDe pend analyzer implementations. Our framework first pre pares proper inputs for analyzers. The inpu t of plug-in analyzer is an ob ject with “IFolder” interface representing the IPackageFragmentRoot folder “bin” in Eclipse workspace. The input of analyzer with command line interface is the folder location. After invocation of the pl ug-in analyzer, the command for analyzing the Log4j is triggered and the analysis results are shown in the graphical user interface. For the a nalyzer with co mmand line interface, our framework generates a concrete command “java jdepend.xmlui.JDepend -file C:\log4j\AnalyticalResult\JDepend_bin_20111223161327.xml C:\log4j\bin” automatically when the log4j project locates at C:\log4j. Our framework uses runtime class to create a c hild process for command execution.

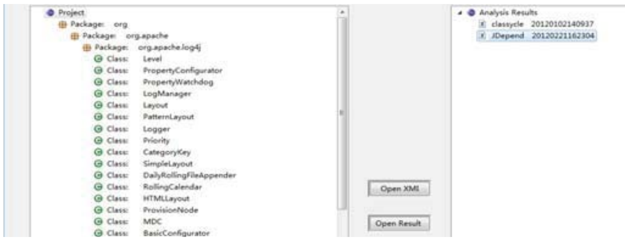


```
<?xml version="1.0"?>
<JDepend>
  <Package name="javax.jms">
    <Stats>
      <TotalClasses>58</TotalClasses>
      <ConcreteClasses>15</ConcreteClasses>
      <AbstractClasses>43</AbstractClasses>
      ......
    </Stats>
    ......
</JDepend>
```

```
<resultMap>
  <pluginName>JDepend</pluginName>
  <inputPath>log4j/</inputPath>
  <outputPath>C:\log4j\AnalyticalResult\JDepend_bin_20111223161327.xml</outputPath>
  <analysingTime>20120221162304</analysingTime>
</resultMap>
```

Figure 5. The traceability recording process

In the input adaptation and analyzer execution stage, our framework saves tr aceability information to dedu ce the relationship between UML elements and analysis results. The relationship between the e lement “project” in the UML m odel and the analysis result file generated by JDepend is saved in an XML file “ResultMap.xml”. The process is shown in Fig. 5. Our framework extends the UML model with th e JDepend analysis result. The user can interpret the analysis results in the tree view of UML model, as show in Fig. 6. All the analysis results of the selected element are in the table. Their corresponding analysis result files will be open if selected.



```
Project
  Package:   org
    Package:   org.apache
      Package:   org.apache.log4j
        Class:   Level
        Class:   PropertyConfigurator
        Class:   PropertyWatchdog
        Class:   LogManager
        Class:   Layout
        Class:   PatternLayout
        Class:   Logger
        Class:   Priority
        Class:   CategoryKey
        Class:   SimpleLayout
        Class:   DailyRollingFileAppender
        Class:   RollingCalendar
        Class:   HTMLLayout
        Class:   ProvisionNode
        Class:   MDC
        Class:   BasicConfigurator
```

```
Analysis Results
  classycle   20120102140937
  JDepend   20120221162304
```

Figure 6. User interface of analysis result interpretation

V. RELATED WORKS

Many approaches and frameworks have been developed for complex analysis of software systems. Most of these approaches focus on the integration of software modeling with quality analysis methods [1]. Model interchange approaches have been proposed to make it possible to create a software model in a tool, then automatically transform the model description for conducting specific analysis, and obtain the results for use. DULLY [3] and XTEAM [4] are developed for facilitating the development of model interpreters. DULLY [3] 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 different modeling tools. XTEAM [4] is an interpreter framework focusing on solving the mismatch between software model and required input of analysis methods. KAMI [2] is a framework for run-time model adaptation. It provides facilities to simplify the development model updaters. However, these approaches focus on the integration of model-based analysis method. They assume that all the artifacts, no matter analysis input or output, are all models. Our work considers the integration of analysis methods for other software artifacts, such as source code analysis methods. In addition, integration of analysis result is also supported in our framework.

Several approaches are proposed to facilitate the use of multiple analysis methods. ADD [5] and SOFAS [6] provide standard interfaces for integration of analysis methods. ADD supports integrating analysis results. SOFAS is developed based on the idea of software analysis as a service. SOFAS follows the principles of a RESTful architecture so as to make analyses easily accessible and composable. However, few analysis methods can be integrated, because existing methods cannot fulfill the design and implementation requirements of these frameworks. Our approach aims at enabling effective reuse of existing analysis methods.

VI. CONCLUSION AND FUTURE WORK

Considering providing a high-level view during analysis, we proposed a model-centric approach for the integration of software analysis methods. Our approach includes three aspects: input adaptation, analyzer execution and traceability recording for analysis result integration, so as to support fully integration of analysis methods.

In the future, we will define more input adaptation rules so as to support analyzing different kinds of software system, such as C++ application, web application, and so on.

ACKNOWLEDGMENT

This effort is supported by the National Natural Science Foundation of China (Grant No. 61100002), Guangdong Natural Science Foundation (Grant No. S2011040005180); Shenzhen Technology R&D Program for Basic Research (Project Name: Research on Software Architecture centric Analysis for Internetware and its Application in Digital Home); the National Key Technology R&D Program (Grant Nos. 2011BAH27B01 and 2011BHA16B08); the Major Science and Technology Projects of Guangdong (Grant No. 2011A080401007), and the Industry-academy-research Project of Guangdong (Grant No. 2011A091000032).

REFERENCES

[1] Dobrica L. Exploring Approaches of Integration Software Architecture Modeling with Quality Analysis Models. Proceedings of Ninth Working IEEE/IFIP Conference on Software Architecture, 2011: 113-122.

[2] 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

[3] 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.

[4] Edwards G, Medvidovic N. A methodology and framework for creating domain-specic development infrastructures. Proc of the 23rd IEEE ACM Int'l Conference on Automated Software Engineering. 2008: 168-177.

[5] Bass L, Clements P, Kazman R. Software Architecture in Practice. 2nd ed. Addison-Wesley. 2003.

[6] Ghezzi G and Gall HC. SOFAS: A Lightweight Architecture for Software Analysis as a Service. Proceedings of the 2011 Ninth Working IEEE/IFIP Conference on Software Architecture, 2011: 93-102.

[7] 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

[8] Object Management Group. UML Version 2.4.1. http://www.omg.org/spec/UML/2.4.1/

[9] Jdepend. http://www.clarkware.com/software/JDepend.html

[10] Apache Log4j. http://logging.apache.org/log4j/

[11] P. Tonella and A. Potrich. Reverse Engineering of Object Oriented Code. Springer-Verlag, Berlin, Heidelberg, New York, 2005.

[12] Jackson D, Rinard M. Software Analysis: A Roadmap. In The Future of Software Engineering, Anthony Finkelstein (Ed.), pp. 215-224, ACM Press 2000.

[13] Classycle. http://classycle.sourceforge.net/index.html

[14] CodePro Analytix. http://code.google.com/intl/zh-CN/javadevtools/codepro/doc/index.html

[15] Hovemeyer D, Pugh W. Finding Bugs Is Easy. Companion of the 19th Ann. ACM SIGPLAN Conf. Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA '04), Oct. 2004.

[16] Checkstyle. http://checkstyle.sourceforge.net/

[17] PMD. http://pmd.sourceforge.net/

[18] Yasca. http://www.yasca.org/

[19] JLint. http://artho.com/jlint/

[20] JavaNCSS. http://www.kclee.de/clemens/java/javancss/

[21] EclipseUML.http://www.ejb3.org/index.ht

CATESR: Change-aware Test Suite Reduction Based on Partial Coverage of Test Requirements

Lijiu Zhang†, Xiang Chen‡, Qing Gu†*, Haigang Zhao†, Xiaoyan Shi†, Daoxu Chen†

† State Key Laboratory for Novel Software Technology
Nanjing University, Nanjing, China
Email: jssyzlj@gmail.com

‡ School of Computer Science and Technology
Nantong University, Nantong, China
Email: xchencs@ntu.edu.cn

***Abstract*—Test suite reduction is an effective technique to save the cost of regression test. This technique is performed by identifying and eliminating redundant test cases from regression test suite. However, fault detection ability of original test suite may also be seriously weakened due to excessive reduction in test suite. In this paper, we propose a new change-aware test suite reduction approach CATESR, based on the conjecture that test cases which are more likely to cover the changes after code modification can gain a higher probability to reveal potential faults in the modified code. To assess the effectiveness of our approach, we implement CASTER and conduct a set of empirical studies on eight real C programs. The results show that our approach can not only greatly decrease the size of reduced test suite but also keep, sometimes even improve, the fault detection ability compared to HGS approach.**

***Index Terms*—Regression Testing, Test Suite Reduction, Test Requirement Classification, Change-aware**

I. Introduction

During software development and maintenance, developers often modify codes and then trigger software evolution. Code modification may be caused by new user requirements, faults detected, code refactoring, or performance improvement. Regression testing is an important method to guarantee the quality of software during its evolution. Statistical data shows that the cost of regression testing accounts for over 1/3 of total development cost [15]. Main research issues in regression testing include test suite reduction, test case selection, test case prioritization, and test suite augmentation [8].

In this paper, we mainly concentrate on test suite reduction issue. Some of previous research mainly focus on proposing novel approaches to reduce test suites according to a single test coverage criterion [2] [3] [7] [16] [21]. However, Rothermel et al. observed that test suite reduction can significantly decrease the size of test suites but at the cost of seriously losing original fault detection ability (FDA) [19]. To solve this issue, researchers have proposed some approaches based on multiple test coverage criteria. For example, Jeffrey and Gupta proposed a selective redundancy concept [13]. Black et al. performed test suite reduction using bi-criteria binary ILP model [1]. Then Hsu and Orso extended this work [1] and developed a general tool MINTS [11].

Differing from previous research, we propose a change-aware test suite reduction approach. According to the previous approaches analysis, we find these approaches seldom analyzed the relationship between code changes and test requirements. In our research, we want to focus on test cases that are more likely to cover the changes after code modification in hope of detecting potential faults. Based on this conjecture, we propose CATESR approach to perform test suite reduction base on partial coverage of test requirements. In particular, given a test suite TS and a set of test requirements R, we firstly label R into three categories after code change analysis. Secondly, we classify TS into three sub test suites based on the requirement label result. Finally, we design a strategy named *Partial*-CATESR, which only performs test suite reduction over the sub test suite with the highest ability to cover the code changes. However, this strategy maybe lose partial test requirement coverage in most cases. Therefore, to verify the effectiveness of *Partial*-CATESR, we also design a strategy named *Full*-CATESR, which performs test suite reduction over all three sub test suites respectively and combine these test suites into a single test suite.

In our empirical study, we choose 7 real C programs in Siemens suite and 1 large-scale C program named *space* as our experiment subjects. In experiment setup, we choose HGS approach as our test suite reduction approach and make comparisons among HGS, HGS-based *Partial*-CATESR, and HGS-based *Full*-CATESR. After data analysis, we find: (1) *Partial*-CATESR can not only greatly decrease the size of reduced test suite, but also keep, sometimes even improve, the FDA of reduced test suite compared to HGS approach. (2) The fault detection ability of reduced test suite are not weakened due to the partial coverage of test requirements.

The main contributions of this paper include:

- we propose a change-aware test suite reduction approach CATESR based on partial coverage of test requirements.
- we conduct an empirical study to verify the effectiveness of our approach.

* Correspondence author. Email: guq@nju.edu.cn

II. Background

In this section, we firstly introduce the preliminaries of test suite reduction and then show the motivation of our approach by a simple example.

A. Test Suite Reduction

In regression testing, a simple strategy is to use all previous test cases to test the modified software, however this strategy is not always feasible. Researchers proposed different effective approaches to solve this problem [8]. This paper mainly focuses on the issue of test suite reduction, which was originally formalized as follows by Harrold et al. [7]:

Given: A test suite $TS = \{tc_1, tc_2, \cdots, tc_m\}$, a requirement set $R = \{r_1, r_2, \cdots, r_n\}$ generated by a test adequacy criteria C, and subsets of TS: $T_1, T_2, \cdots, T_n$, for each associated with a r_i such that any test case $tc_j \in T_i$ can be used to cover requirement r_i.

Problem: To find a subset of T with the minimum cardinality that covers all the requirements covered by T.

This problem has been demonstrated to be NP-Complete and we summarize these related work in Section V.

B. Motivating Example

To illustrate the limitation of previous approaches and show the motivation of our approach, we design a simple example shown in Table I.

TABLE I
A Motivating Example

Program Segments	Test Cases		
bool ***TestAndSubtract***(m, n)	$< 3, 4 >$	$< 4, 3 >$	$< 13, 13 >$
1: if ($m < 10$)	X	X	X
2: *alert(m);*	X	X	
3: if ($m <= n$)	X	X	X
4*: *alert(n);*	-	-	-
4: return *false*;	X		X
5: else		X	
6: $m -= n$;		X	
7: return *true*;		X	
Fault Detection Ability	T	F	F

The left-most column in Table I shows a segment of program under test. The segment implements a function ***TestAndSubtract*** to test a non-negative integer n and subtract n from a integer m. In particular, it firstly performs a low-value checking for m (Lines 1-2). Then if m is not large enough to be subtracted by n, this function will return $false$ (Lines 3-4). Otherwise, this function will perform a subtraction operation and return $true$ (Lines 5-7).

The rightmost three columns in Table I show the statement coverage information of three test cases. For example, after running the test case $< 3, 4 >$, this test case can cover statements from line 1 to line 4.

Suppose a new requirement is proposed to alert m when m is less than or equal to n. However this requirement is incorrectly implemented by using statement alert(n) (Line 4*) while the right implementation is alert (m). The bottom row of Table I shows the fault detection ability of all three test cases. Here only the test case $< 3, 4 >$ can detect this fault. If we use existing test suite reduction approaches, such as HGS [7], we can get a reduced test suite $\{< 3, 4 >, < 4, 3 >\}$ or $\{< 4, 3 >, < 13, 13 >\}$. In this case, it only has 50% probability to detect this fault. However, if we further analyze the code change, we can find that we do not need to re-execute the test case $< 4, 3 >$ since this test case do not cover change related statements (Lines 5-7). Therefore we can remove this test case and then perform a test suite reduction on $\{< 3, 4 >, < 13, 13 >\}$. Finally we can choose the test case $< 3, 4 >$ and this test case can detect the fault in the modified function. Meanwhile the size of reduced test suite is decreased from 2 to 1.

III. CATESR Approach

In this section, we firstly introduce the framework of our CATESR approach and then introduce the implementation detail.

A. The Framework of CATESR approach

We use Figure 1 to show the framework of CATESR approach. In particular, CATESR takes two consecutive versions of source code and coverage information of test cases on i-th version as its input, and produces a reduced test suite to $(i + 1)$-th version as its output.

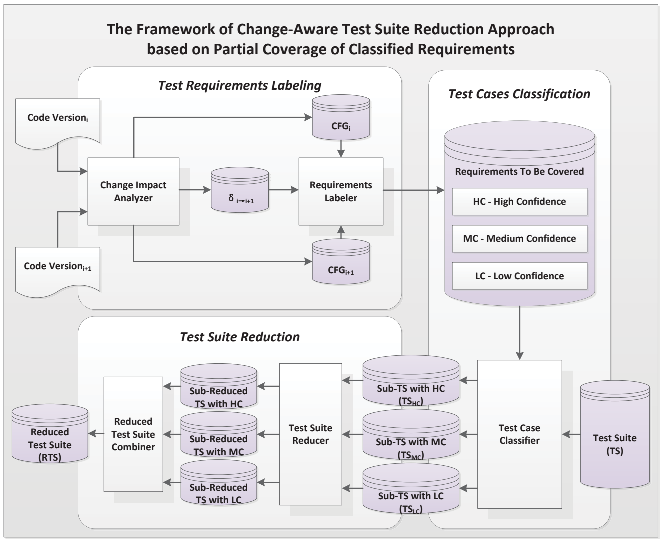


Fig. 1. The framework of CATESR

CATESR consists of three modules: Test Requirements Labeling Module (M_1), Test Case Classification Module (M_2), and Test Suite Reduction Module (M_3). Firstly, we perform a code change analysis in M_1 and label original test requirements into three categories: low confidence (LC) category, medium confidence (MC) category, and high confidence (HC) category. This test requirement labeling process is performed based on the dependence analysis of code change. Meanwhile these three categories conform the following total ordering relation.

$$LC \prec MC \prec HC. \tag{1}$$

Secondly, we divide the original test suite (TS) into three sub test suites in M_2 according to the ability to cover the requirements with highest confidence category. Finally, we perform test suite reduction over some or all sub test suites independently in M_3 and design a strategy to combine these reduced sub test suites into a reduced test suite.

B. The Implementation detail of CATESR approach

1) Test Requirements Labeling Module (M_1): This module aims to label the test requirements into discriminating categories according to code change analysis. These categories indicate the ability to be aware of the changes for test cases who cover requirements in respective category. This module includes two components: Change Impact Analyzer Component (C_1) and Requirements Labeler Component (C_2).

Firstly, given two consecutive versions $Version_i$ and $Version_{i+1}$, we can use C_1 to separately generate their control flow graphs (CFGs) CFG_i and CFG_{i+1}. Then we can analyze the difference $\Delta i \rightarrow i+1$ between these two versions. An item δ in set $\Delta i \rightarrow i+1$ can be defined using the following tuple:

$$\delta \triangleq (type, line_{ob}, line_{oe}, line_{nb}, line_{ne}). \tag{2}$$

The tuple δ contains five elements: $type$, $line_{ob}$, $line_{oe}$, $line_{nb}$, and $line_{ne}$. The $type$ element denotes code change type and the valid value set is {*ADD*, *DELETE*, *UPDATE*, *UNCHANGED*}. To assistant code change analysis, four additional elements are needed. The $line_{ob}$ and $line_{oe}$ elements specify the beginning and end line-numbers of old statements in $Version_i$, and the $line_{nb}$ and $line_{ne}$ elements specify the beginning and end line-numbers of new statements in $Version_{i+1}$.

Then we can use C_2 to label test requirements[1] according to the information collected by *C1*. We use Algorithm 1 to describe this requirement labeling process.

Given a statement-level test requirements set $Reqs_{state}$, CFG_i, CFG_{i+1}, and change information $\Delta_{i \rightarrow i+1}$ from $Version_i$ to $Version_{i+1}$, Algorithm 1 can label each requirement in $Statements$ by a proper category indicating the confidence of being covered by our approach.

Algorithm 1 mainly has two steps: requirements labeling preparation and requirements labeling. In requirements labeling preparation step, we iteratively collect the affected blocks using an auxiliary function GetBlocks (Lines 4-10), and build the line-number one-one mapping from new to old statements for *UNCHANGED* code segments (Lines 11-16).

The signature of function GetBlocks is designed as follows:

$$blocks\ GetBlocks(CFG, line_{begin}, line_{end});$$

This function can return all blocks in CFG that contain any of the statements indexed with the line-number ranging from $line_{begin}$ to $line_{end}$. Due to the limitation of the space, we omit the implementation details of this function.

Algorithm 1 ***Requirements Labeling*** algorithm

```
Require:
    $Reqs_{state}$: *statement*-level test requirements;
    $CFG_i$: control-flow graph of $Version_i$;
    $CFG_{i+1}$: control-flow graph of $Version_{i+1}$;
    $\Delta_{i \rightarrow i+1}$: software evolution from $Version_i$ to
        $Version_{i+1}$.
Ensure:
    Each requirement in $Reqs_{state}$ will be marked with a
    proper confidence category (i.e., HC, MC, or LC).
1:  $Map_{new \rightarrow old}, Blocks_{new}, Blocks_{old} \leftarrow \emptyset$;
2:  // Step 1: Calculate Affected Blocks and build the 1-1
    line-number map from new to old version.
3:  for all $\delta \in \Delta_{i \rightarrow i+1}$ do
4:      if $\delta.type =$ "ADD" then
5:          $Blocks_{new} \leftarrow Blocks_{new}$
                $\cup\ GetBlocks(CFG_{i+1}, \delta.line_{nb}, \delta.line_{ne})$;
6:      else if $\delta.type =$ "DELETE" then
7:          $Blocks_{old} \leftarrow Blocks_{old}$
                $\cup\ GetBlocks(CFG_i, \delta.line_{ob}, \delta.line_{oe})$;
8:      else if $\delta.type =$ "UPDATE" then
9:          $Blocks_{new} \leftarrow Blocks_{new}$
                $\cup\ GetBlocks(CFG_{i+1}, \delta.line_{nb}, \delta.line_{ne})$;
10:         $Blocks_{old} \leftarrow Blocks_{old}$
                $\cup\ GetBlocks(CFG_i, \delta.line_{ob}, \delta.line_{oe})$;
11:     else if $\delta.type =$ "UNCHANGED" then
12:         $bias \leftarrow \delta.line_{ob}$ - $\delta.line_{nb}$;
13:         for $line \leftarrow \delta.line_{nb}$ to $\delta.line_{ne}$ do
14:             $Map_{new \rightarrow old}$.Put($line$, $line$ + $bias$);
15:         end for
16:     end if
17: end for
18: // Step 2: Mark Test Requirements with proper confi-
    dence category.
19: Initialize all $s \in Reqs_{state}$ with $Confidence$ LC;
20: for all $block \in Blocks_{old}$ do
21:     Mark all $s \in block.Statements$ with Confidence HC;
22: end for
23: for all $block \in Blocks_{new}$ do
24:     MarkStatements($block$, $Map_{new \rightarrow old}$, HC);
25: end for
```

When calculating *Affected Blocks*, for *ADD*-type changes, we only account blocks affected in CFG_{i+1} (Lines 4-5). We treat *DELETE*-type changes in the same way (Lines 6-7). However, for *UPDATE*-type changes, we will account blocks in both CFG_i and CFG_{i+1} (Lines 8-10). Here, blocks collected from CFG_i and CFG_{i+1} are separately denoted by $Blocks_{old}$ and $Blocks_{new}$.

When building the line-number mapping $Map_{new \rightarrow old}$ for each *UNCHANGED* code segment δ (Line 11), we firstly calculate the bias of line-number from old to new statement, denoted by $bias$ (Line 12). Then we iteratively put line-number pairs $< line, line + bias >$ into $Map_{new \rightarrow old}$ for each $line$

[1] In this paper, we take statement coverage as our test criterion, thus statements can be treated as test requirements needed to be covered.

Algorithm 2 ***MarkStatements*** function

```
Require:
    $block$: program block to be processed;
    $Map_{new \rightarrow old}$: 1-1 line-number mapping from new to old statements for unchanged code segments;
    $CONFIDENCE$: confidence category (i.e., LC, MC, or HC).
Ensure:
    Requirements W.R.T $block$ will be properly labeled.
$Stmts_{new} \leftarrow block.Statements \cap Map_{new \rightarrow old}.Keys$;
if $Stmts_{new} \neq \emptyset$ then
    $Stmts_{old} \leftarrow Map_{new \rightarrow old}$.MapToValues($Stmts_{new}$);
    if $Stmts_{old}.Confidence \prec CONFIDENCE$ then
        Mark all $s \in Stmts$ with $CONFIDENCE$;
    end if
else
    for all $block_{pre} \in block.PreBlocks$ do
        MarkStatements($block_{pre}$, $Map_{new \rightarrow old}$, MC);
    end for
    for all $blcok_{succ} \in block.SuccBlocks$ do
        MarkStatements($block_{succ}$, $Map_{new \rightarrow old}$, MC);
    end for
end if
```

ranging from $\delta.line_{nb}$ to $\delta.line_{ne}$ (Lines 13-15).

During the requirements labeling phase, all requirements from $Reqs_{state}$ are initially assigned to the lowest confidence category *LC* (Line 19). If test cases can cover those statements in affected blocks $Blocks_{old}$ and $Blocks_{new}$, they can be aware of changes and further reveal potential faults with high probability. In CATESR, for blocks in $Blocks_{old}$ collected from $Version_i$, their statements are also in $Reqs_{state}$ as partial requirements, will be reassigned to the highest confidence category *HC* (Lines 20-22). However, for blocks in $Blocks_{new}$ collected from $Version_{i+1}$, their statements may not exist in the older version $Version_i$. To label these requirements, we design a recursive function *MarkStatements* (Algorithm 2) (Lines 23-25). When calling *MarkStatements* for the first time, the statements in $Version_i$ will be marked with the highest confidence category *HC*. If no such statements exist, the algorithm will recursively check such statements in predecessors and successors of current block, and mark them with a lower confidence category *MC*.

2) Test Cases Classification Module (M_2): We use this module to classify test cases into proper categories. Test cases in the same category can at most cover the same level of requirements' confidence. After performing M_1, all test requirements are labeled by a confidence category (i.e., *HC*, *MC*, or *LC*). Then, based on the total order relation given by formula 1, for each test case tc in test suite TS, M_2 computes the highest confidence of requirements which tc can cover, denoted by $Confidence$, and put tc into one proper subset of TS in accordance with $Confidence$. We use algorithm 3 to show the process of this module.

In Algorithm 3, we define three sets, TS_{HC}, TS_{MC}, and TS_{LC} to store test cases that can cover requirements with confidence *HC*, *MC*, and *LC* at most respectively. Initially, we set these result sets empty (Line 1). Then, for each test case tc, we compute the highest confidence of requirements tc can cover (i.e., $Confidence$)(Lines 3-8). Finally, we add tc to TS_{HC}, TS_{MC}, or TS_{LC} according to $tc.Confidence$ (Lines 9-15). After classification, each test case can be in only one of TS's subsets (i.e., TS_{HC}, TS_{MC}, or TS_{LC}).

Algorithm 3 ***Test Case Classification*** Algorithm

```
Require:
    $Requirements$: labeled statement-level requirements;
    $TS$: original test suite contains test cases to be classified;
Ensure:
    $TS$ is divided into 3 pieces, $TS_{HC}$, $TS_{MC}$, and $TS_{LC}$, in which test cases can only get the highest confidence HC, MC, and LC respectively.
 1: $TS_{HC}, TS_{MC}, TS_{LC} \leftarrow \emptyset$;
 2: for all $tc$ in $TS$ do
 3:     $tc.Confidence \leftarrow$LC;
 4:     for all $req$ in $Requirements$ do
 5:         if $tc.Cover(req)$ &&
                $tc.Confidence \prec req.Confidence$ then
 6:             $tc.Confidence \leftarrow req.Confidence$;
 7:         end if
 8:     end for
 9:     if $tc.Confidence =$ HC then
10:         $TS_{HC}$.Add($tc$);
11:     else if $tc.Confidence =$ MC then
12:         $TS_{MC}$.Add($tc$);
13:     else if $tc.Confidence =$ LC then
14:         $TS_{LC}$.Add($tc$);
15:     end if
16: end for
```

3) Test Suite Reduction Module (M_3): We use M_3 to generate the final reduced test suite. After classification by M_2, original test suite TS is divided into three sub test suites TS_{HC}, TS_{MC}, and TS_{LC}. Then we can reduce these three sub test suites by using a classical test suite reduction approach HGS [7]. Three reduced test suite with each confidence category *HC*, *MC*, and *LC* are respectively denoted by RTS_{HC}, RTS_{MC}, and RTS_{LC}. Finally, we propose a strategy named *Partial*-CATESR to generate the final reduced test suite. Meanwhile to illustrate the effectiveness of *Partial*-CATESR, we also propose another strategy named *Full*-CATESR for comparison.

Partial-CATESR is motivated by research work of Gu et al. [5]. They conjectured that not all test requirements need to be covered during regression testing. Their empirical results show that properly not covering irrelevant test requirements could not only reduce the size of test suite, but also keep the FDA. Similarly, our *Partial*-CATESR strategy pays more attention to the test cases covering the most related requirements (i.e., TS_{HC}) and ensure the same requirement coverage of TS_{HC}.

Based on the former analysis, reduced test suite RTS_{HC} has the highest confidence to cover the code changes after modification, RTS_{MC} the second, and RTS_{LC} the lowest.

Therefore, the calculation of RTS under *Partial*-CATESR can be given by the following process.

$$RTS \leftarrow \begin{array}{l} \textbf{if } TS_{HC} \neq \emptyset \textbf{ then } RTS_{HC} \\ \textbf{else if } TS_{MC} \neq \emptyset \textbf{ then } RTS_{MC} \\ \textbf{else } RTS_{LC} \end{array} \tag{3}$$

Full-CATESR is a naive strategy by simply combining three sub test suites into one test suite. The construction process of RTS can be defined as follows:

$$RTS \leftarrow RTS_{HC} \cup RTS_{MC} \cup RTS_{LC} \tag{4}$$

IV. Empirical Studies

To evaluate the effectiveness of our CATESR approach, we perform empirical studies to answer the following research questions:

RQ1: To what extent can CATESR approach decrease the size of reduced test suite compared to HGS approach?

RQ2: Can the FDA be kept, when CATESR approach generates a relatively smaller reduced test suite, comparing to HGS approach?

RQ3: Can the FDA be weakened due to the partial coverage of test requirements?

A. Subjects and Experiment Setup

1) Experiment Subjects: We adopt seven small C programs in Siemens suite and one large-scale C program named *space* in our empirical study. The Siemens suite is originally contributed by Ostrand et al. for a study of the fault detection abilities of control-flow and data-flow coverage criteria [12], and has been partially modified by researchers for further studies. *space* is a program for interpreting statements written in some specific array definition language (ADL). Each subject contains a single correct version and a set of versions with a single fault.

TABLE II
Experiment Subjects

Subject	# Test Cases	# Versons	LOC	Description
printtok	4130	7	402	Lexical Analyzer
printtok2	4115	10	483	Lexical Analyzer
replace	5542	29	516	Pattern Replacement
schedule	2650	9	299	Priority Scheduler
schedule2	2710	9	297	Priority Scheduler
tcas	1608	40	138	Altitude Separation
totinfo	1052	23	346	Information Measure
space	13585	35	6218	ADL Interpreter

The characteristics of these subjects are summarized in Table II. Each subject contains a large test pool with at least 1052 test cases, and 13585 at most. Each subject contains multiple single-faulty versions, with the count between 7 and 40. For subjects in Siemens suite, the number of lines of code (LOC) ranges from 138 to 516, which is relatively small comparing to practical programs. Therefore, we introduce *space*, which contains 6218 LOC, to verify the scalability of our CATESR approach. These subjects are available from Subject Infrastructure Repository (SIR) at University of Nebraska-Lincoln [2] [4].

2) Experiment Setup: Since there exists randomness in our approach, we independently perform the experiment 1000 times on each faulty version for each subject. During each iteration, we firstly generate a test suite by randomly choosing test cases from the test pool, then we adopt both *Partial*-CATESR and *Full*-CATESR in test suite reduction. To show the effectiveness of our approach, we also implement HGS algorithm [7] as a baseline. When randomly generating a test suite, we firstly determine the size n of test suite. The value of n is determined by LOC of the subject timing a random number ranging from 0 to 0.5. Then we randomly choose n test cases from the test pool and construct TS. Finally, when test cases in TS cannot cover all feasible requirements covered by the test pool, we will add additional test cases by randomly choosing test cases from the remainder of the test pool. If no test case in TS can detect the fault in this faulty version, we will discard TS and regenerate the test suite. This experiment design is motivated by Rothermel et al. [19].

B. Results and Analysis

In this subsection, we analyze the data gathered from our experiments to answer **RQ1**, **RQ2**, and **RQ3**.

1) Experiment Metrics: In our empirical study, we mainly focus on two metrics: time cost and FDA.

In real testing scenarios, time cost includes test environment setup, test case execution, and test result examination. Since test suite reduction can significantly reduce the time cost of regression testing, here we only consider the size of test suite as a metric to measure the time cost. Consequently, we can adopt the average extent of reduction to original test suite in percentage over 1000 times independent experiments, given by the following formula:

$$TS_{Reduced} = \frac{|TS|_{avg} - |RTS|_{avg}}{|TS|_{avg}} * 100 \tag{5}$$

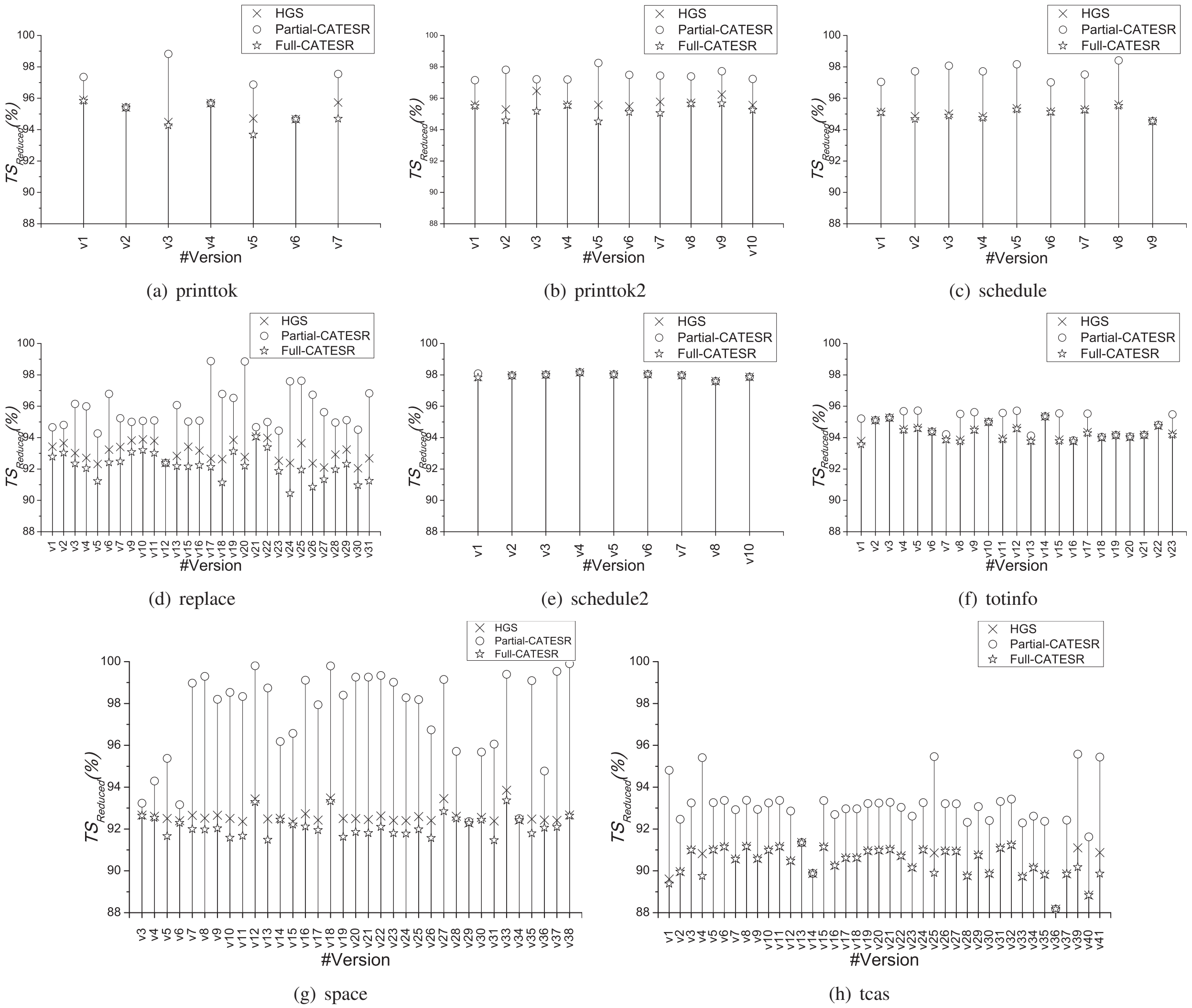
To measure the effectiveness of our approach, we also need to check FDA after test suite reduction. Since each faulty version contains only one fault, we define an indicator function g_f as follows:

$$g_f(TS) = \begin{cases} 1 & \text{if fault } f \text{ can be detected by } TS, \\ 0 & \text{other wise.} \end{cases} \tag{6}$$

Consequently, we can take the average value $\overline{g_f(TS)}$ of indicator over 1000 experiments as the metric of fault detection ability, denoted by FDA.

2) Test Suite Size Analysis: To answer **RQ1**, we analyze the data, and fetch the size of reduced test suites. The reduction extent is measured by formula 5 and visualized in Figure 2. For each subject over all available faulty versions, we mark three key values to show the reduction extent $TS_{Reduced}$ for HGS, *Partial*-CATESR, and *Full*-CATESR,respectively in the same vertical drop line.

[2] http://sir.unl.edu/php/index.php



(a) printtok

(b) printtok2

(c) schedule

(d) replace

(e) schedule2

(f) totinfo

(g) space

(h) tcas

Fig. 2. The size of test suite reduction for each subject

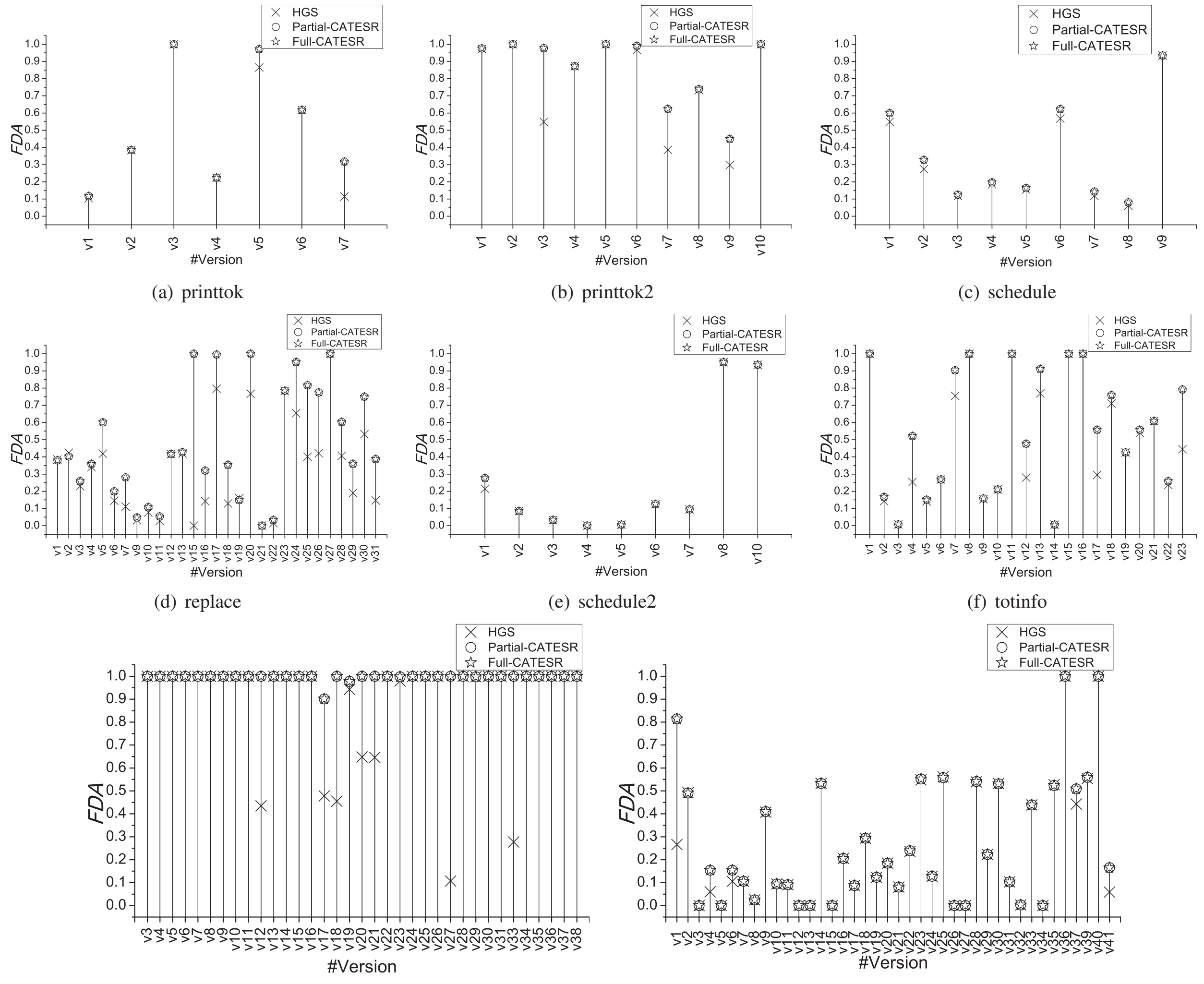
We can find that our *Partial*-CATESR can at least gain the same reduction extent as HGS performs. At most of the cases, we can get more reductions. Within subject *space*, for instance, original test suite are reduced by around 99 percentage using our *Partial*-CATESR. In this case, the size of reduced test suite is much smaller than reduced by around 93 percentage using pure HGS approach. For those versions on which our *Partial*-CATESR performs as well as pure HGS, we seriously check both the source codes and their test cases. We find that all test cases are classified in the same category, because either they all have a strong confidence to cover the changes, or they are insensitive to the changes through our CFG-based change analysis.

Note that the performance of *Full*-CATESR, including reduction extent and FDA, will be discussed at the end of this subsection.

3) Fault Detection Ability Analysis: To answer **RQ2**, we also check the FDA using the metric FDA as we have discussed above. Similar to the visualization of reduction extent, we summarize FDA for each subject over all faulty versions in Figure 3.

Clearly, FDA among all our experiments are not weakened at all, and even strengthened many times. For some faults in many subjects, such as *replace* and *space*, the probabilities of being detected, used to be below 0.1 using pure HGS approach, are increased to 1 when adopting our *Partial*-CATESR.

4) Partial Coverage Affect Analysis: To answer **RQ3**, we design the *Full*-CATESR, to assure a 100 percentage coverage of original test requirements. As we can see in Figure 2, the reduction extent are decreased due to additional test cases added to RTS, which is generated by *Partial*-CATESR. However, the FDA among all experiments are not enhanced at all (See Figure 3). Our experiment results show that, it is meaningless to re-execute test cases who have few relevance with the changes during regression testing, though they may cover some irrelevant test requirements. Therefore, we conclude that the FDA will not be weakened by the insufficient coverage of original test requirements.



(a) printtok (b) printtok2 (c) schedule

(d) replace (e) schedule2 (f) totinfo

(g) space (h) tcas

Fig. 3. The FDA of test suite reduction for each subject

C. Threats to Validity

We mainly discuss some potential threats for our empirical studies in this subsection. Threats to internal validity mainly come from the incorrect program implementation or corrupted data. To avoid these threats, we prepared our data carefully and tested our programs with simple programs. Threats to external validity affect the result generalization. The primary threat is the representativeness of our subjects. However, these subjects are commonly used by other researchers. Another threat is that we only adopt HGS as our test suite reduction approach. Using other reduction approaches may affect the conclusion of our paper. Threats to conclusion validity arise when accounting for the random variation in our approach. To overcome these threats, we need more sophisticated statistical hypothesis tests to have a higher confidence in our results.

V. Related Work

Test suite reduction is an important research topic in regression testing. It has been successfully applied to different application domains, such as web applications [20], GUI applications [18], and fault localization [23]. We summarize previous research work into three categories.

The first category of research focuses on proposing effective approaches given a single test coverage criterion. Horgan and London used linear programming to conduct test suite reduction based on data flow coverage [10]. Harrold et al. proposed a classical greedy approach called HGS [7]. Chen and Lau proposed two greedy algorithms called GE and GRE [2]. Jones and Harrold developed two techniques according to the modified conditon/decision coverage (MC/DC) criterion [14]. Tallam and Gupta proposed an approach based on formal concept analysis [21]. Since test suite reduction is a typical combinatorial optimization problem, Mansour et al. applied meta-heuristic search techniques to this problem, such as simulated annealing and genetic algorithm [16]. Zhang et al. performed test suite reduction for JUnit test suites [24]. Chen et al. proposed a test reduction approach based on the pairwise interaction of test requirements [3].

The second category of research focuses on performing test suite reduction based on multiple coverage criteria. Jeffrey and

Gupta extended HGS and proposed the concept of selective redundancy. They attempted to select additional test cases that are redundant to a particular coverage criterion but are not redundant with respect to one or more other coverage criteria [13]. Black et al. proposed a reduction approach using bi-criteria binary ILP model [1]. Hsu and Orso extended the work [1] and could solve a wide range of multi-criteria test suite reduction problems [11].

The remaining work can be categorized into the third category of research. Vaysburg et al. used EFSM dependence analysis to conduct requirement-based test suite reduction [22]. Harder et al. generated, augmented, and minimized test suites by the operational difference technique [6]. Martina and Bertolino introduced the notion of spanning sets of entities and this notion can be used to perform test suite reduction [17]. Heimdahl and George conducted test suite reduction with respect to given specification-based criteria that are based on formal models of the software [9].

Different from previous research, we consider the relationship between code changes and test requirements. We attempt to focus on test cases that are more likely to be aware of the changes after code modification, and give a preliminary empirical study for our proposed approach.

VI. Conclusion

Test suite reduction can significantly decrease the size of test suites but at the cost of seriously losing original fault detection ability. Based on the conjecture that test cases who are more likely to cover the changes after code modification can gain a higher probability to reveal potential faults, we propose a new change-aware test suite reduction approach CASTER. Our empirical studies show the effectiveness of our approach.

In the future, we want to further consider the following issues. Firstly we want to apply data-flow analysis to our approach, as a complement of current control-flow based change analysis. Secondly we want to apply our approach to more subjects written by other programming languages. Last but not least we want to apply our approach to larger-scale real-world applications.

Acknowledgment

This work is supported in part by the National High-Tech Research and Development Plan of China (863) under Grant No. 2006AA01Z177, the 973 Program of China under Grant No. 2009CB320705, and the NSFC Projects under Grant No. 60873027 and Grant No. 61021062.

References

[1] J. Black, E. Melachrinoudis, and D. Kaeli, "Bi-criteria models for all-uses test suite reduction," in *Proceedings of the International Conference on Software Engineering*, 2004, pp. 106–115.

[2] T. Chen and M. Lau, "A new heuristic for test suite reduction," *Information and Software Technology*, vol. 40, no. 5-6, pp. 347–354, 1998.

[3] X. Chen, L. Zhang, Q. Gu, H. Zhao, Z. Wang, X. Sun, and D. Chen, "A test suite reduction approach based on pairwise interaction of requirements," in *Proceedings of the Symposium on Applied Computing*, 2011, pp. 1390–1397.

[4] H. Do, S. Elbaum, and G. Rothermel, "Supporting controlled experimentation with testing techniques: an infrastructure and its potential impact," *Empirical Software Engineering*, vol. 10, pp. 405–435, 2005.

[5] Q. Gu, B. Tang, and D. Chen, "Optimal regression testing based on selective coverage of test requirements," in *Proceedings of International Symposium on Parallel and Distributed Processing with Applications*, 2010, pp. 419–426.

[6] M. Harder, J. Mellen, and M. D. Ernst, "Improving test suites via operational abstraction," in *Proceedings of the International Conference on Software Engineering*, 2003, pp. 60–71.

[7] M. J. Harrold, R. Gupta, and M. L. Soffa, "A methodology for controlling the size of a test suite," *ACM Transactions on Software Engineering and Methodology*, vol. 2, pp. 270–285, 1993.

[8] M. Harrold and A. Orso, "Retesting software during development and maintenance," in *Proceedings of Frontiers of Software Maintenance*, 2008, pp. 99 –108.

[9] M. P. E. Heimdahl and D. George, "Test-suite reduction for model based tests: effects on test quality and implications for testing," in *Proceedings of the International Conference on Automated Software Engineering*, 2004, pp. 176–185.

[10] J. R. Horgan and S. London, "Data flow coverage and the c language," in *Proceedings of the Symposium on Testing, Analysis, and Verification*, 1991, pp. 87–97.

[11] H.-Y. Hsu and A. Orso, "Mints: A general framework and tool for supporting test-suite minimization," in *Proceedings of the International Conference on Software Engineering*, 2009, pp. 419–429.

[12] M. Hutchins, H. Foster, T. Goradia, and T. Ostrand, "Experiments of the effectiveness of dataflow- and controlflow-based test adequacy criteria," in *Proceedings of the International Conference on Software Engineering*, 1994, pp. 191–200.

[13] D. Jeffrey and N. Gupta, "Improving fault detection capability by selectively retaining test cases during test suite reduction," *IEEE Transactions on Software Engineering*, vol. 33, pp. 108–123, 2007.

[14] J. A. Jones and M. J. Harrold, "Test-suite reduction and prioritization for modified condition/decision coverage," *IEEE Transactions on Software Engineering*, vol. 29, pp. 195–209, 2003.

[15] H. K. N. Leung and L. White, "Insights into testing and regression testing global variables," *Journal of Software Maintenance*, vol. 2, pp. 209–222, 1990.

[16] N. Mansour and K. El-Fakih, "Simulated annealing and genetic algorithms for optimal regression testing," *Journal of Software Maintenance*, vol. 11, pp. 19–34, 1999.

[17] M. Marré and A. Bertolino, "Using spanning sets for coverage testing," *IEEE Transactions on Software Engineering*, vol. 29, pp. 974–984, 2003.

[18] S. McMaster and A. Memon, "Call-stack coverage for gui test suite reduction," *IEEE Transactions on Software Engineering*, vol. 34, pp. 99–115, 2008.

[19] G. Rothermel, M. J. Harrold, J. Ostrin, and C. Hong, "An empirical study of the effects of minimization on the fault detection capabilities of test suites," in *Proceedings of the International Conference on Software Maintenance*, 1998, pp. 34–43.

[20] S. Sprenkle, S. Sampath, E. Gibson, L. Pollock, and A. Souter, "An empirical comparison of test suite reduction techniques for user-session-based testing of web applications," in *Proceedings of the International Conference on Software Maintenance*, 2005, pp. 587–596.

[21] S. Tallam and N. Gupta, "A concept analysis inspired greedy algorithm for test suite minimization," in *Proceedings of the Workshop on Program Analysis for Software Tools and Engineering*, 2005, pp. 35–42.

[22] B. Vaysburg, L. H. Tahat, and B. Korel, "Dependence analysis in reduction of requirement based test suites," in *Proceedings of the International Symposium on Software Testing and Analysis*, 2002, pp. 107–111.

[23] Y. 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*, 2008, pp. 201–210.

[24] L. Zhang, D. Marinov, L. Zhang, and S. Khurshid, "An empirical study of junit test-suite reduction," in *Proceedings of the International Symposium on Software Reliability Engineering*, 2011, pp. 170–179.

A Process Model for Human Resources Management focused on increasing the Quality of Software Development

Flávio E. A. Horita, Jacques D. Brancher and Rodolfo M. de Barros
Computer Department
State University of Londrina, UEL
Londrina, Brazil
feahorita@gmail.com; {jacques, rodolfo}@uel.br

Abstract — **The lack of quality in the production process of software development isn't attributed only to the techniques and technologies, but also to the lack of attention and importance placed on its members. Thus, this paper presents a process model for Human Resources Management focused on improving the quality of software development. Its preparation was based on areas and expected results of the process of human resource management present in the Reference Model for Brazilian Software Process Improvement (MR-MPS)[1]. In order to contribute to its understanding and use, it is presented a comparative study with other models present in the literature and identify their benefits and problems with an application in two software development projects.**

Keywords: Human Resources Management; Process Quality; Training Management; Performace Management; Human Factor.

I. Introduction

The high dependence of human resources for the development of a software project has demonstrated the importance of its management. This is due to the fact that some studies have demonstrated that they generate and strengthen innovation, produce, take decisions, lead, motivate, communicate, supervise, manage and direct the business [5], [8], [9], [13], [14], [15].

However, even nowadays, these resources are still losing focus on software development processes, which tend to give more importance to the technical and practical areas [2], [14]. Moreover, it has been ever more common to find members doing exhaustive tasks (working overtime, on weekends or vacation) which end up resulting in their exhaustion, dissatisfaction and demotivation.

Thus, the software development process is not only about the use of software and hardware to generate systems. The development and maintaining is its connected to people.

As a consequence, its success or failure is intimately related to the way in which they are allocated and controlled within your budget and time [15]. Professionals with the right skills to perform their tasks, execute them in a most efficient way, which reflect in the quality of the software development process and in the final product.

Considering what was previously exposed, the aim of this work is to present a model of process focused on the increase of the quality of the software development process. For its elaboration, we based on what Amâncio et al. [1] and Morais [11] suggested and on the expected areas and results of the human resources management process presents in the maturity level E of MR-MPS.

This article is divided in six elementary sections, including this introduction. In Section 2 it is presented the theory related to the paper. In Section 3 it is presented the research methodology that was used. Section 4 it is presented the process model for Human Resources management. In Section 5 it is presented the results of the research. Finally, Section 6 it is presented the conclusions and suggestions for future works.

II. Theory

A. Human Resources Management on Software Development

Several studies demonstrate that holding the best technological tools, using the most efficient techniques and work models is not enough to guarantee the success of a software project [9], [14], [15].

It is necessary the existence, in parallel, of a human resources management able to develop skills and guarantee the effective allocation of its members, in order to increase the quality of its process [12].

However, several managers attribute more importance to the technical and practical areas rather than the human resources, which end up by losing the focus in software development processes [2]. A manager must act in order to encourage the developing staff to work together as a team, concentrating in the customers' needs and product quality.

This context made managers responsible not only for the leadership in planning, organization and control of the efforts expended during the project. They had to develop other skills like manage people, e.g. ability to lead and stimulate people's development, abilities to solve problems and excellent interpersonal competence [15].

Moreover, during the development of a software project, the dynamic in business processes and the high turnover of technologies and his members highlights the importance to

[1] Modelo de Referência para Melhoria do Software Brasileiro (MR-MPS)

manage intellectual knowledge with creating mechanisms to collect, store and share within the organization [6], [12].

B. *Related Work*

There are several authors working hard in new models and processes to improve people management. We have made a deep analysis to find the good and bad points of them.

Amâncio et al.'s work [1] presents the definition of a process model grounded on the Human Resources Management, area of knowledge of PMBoK. The author presents its application in the software factory of a public education institution which main focus is the development of its members in the academic and professional spheres.

This process is divided into four activities, two of them *Planning Needed Human Resources* and *Hiring or Mobilizing Project Team* were developed in the initial phases of the project, while the other two *Develop Project Team* and *Manage Project Team*, must be executed in parallel until its end.

According to the author, although partial, the institutionalization of the process, besides achieving its main objective, the decrease in turnover, managed to increase the quality of its services and products through training and continuous assessment of the skills and performance of the people involved.

A human resource management process was also suggested in De Carvalho's work [7]. This work is divided into three complementary areas: Planning, Monitoring and Assessment, in each one of them its work instructions, resources and roles present in its execution were carefully defined and detailed.

Moreover, the author also defines some "external" activities which must be accomplished in a more administrative sphere. Among them, we can cite *Effectivenes of Professionals Hiring*, that is not part of the IT manager competence and must be carried out by the human resources department.

Also, in this context, Morais [11] presents a human resources management process focused on the improvement of the knowledge identification, storage and sharing process within the organization.

Developed to be adherent to the MR-MPS, this process is composed by six activities that aim to work broadly the organization's needs, its trainings, manageable knowledge and performance, besides controlling the dismissal of its members.

Apart from this presentation, the author performed a preliminary validation of the process through its implementation in the human resources area of a system development organization. According to the author, the model demonstrated to be efficient in this context.

Unlike the three first models, People Capability Maturity Model (P-CMM) maturity model is a variant of Capability Maturity Model (CMM) which has as focus to help in human resources management. To do so, it offers a set of good manners to make provisions for the continuous growing of workforce abilities in the organization [6].

According to Curtis and Hefley [6], the workforce abilities are defined as knowledge level, ability and capability to perform activities within the project. In order to monitor and improve these competences, the model is divided in five maturity levels, so gradually each one of them will be identified, developed and worked. Thus, the advantages identified when implementing the P-CMM vary in function to the maturity level in which the company finds itself [6].

C. Reference Model for the Brazilian Software Improvement Process[2]

Developed in 2003 by the SOFTEX[3] as part of the MPS.Br[4] program, the MR-MPS consists of a reference model with the definition of prerequisites for the improvement of the quality of the software process. Besides it, the program is composed by an Assessment Method (MA-MPS) and a Business Model (MN-MPS), each one of them described by guides and/or document models.

In accordance with Capability Maturity Model Integration for Development (CMMI-DEV) and following the described headlines in its main program, this model was divided into seven maturity levels. These levels define steps to improvement processes in the organization [10]. Moreover, this division aims to enable its implementation and assessment in micro, small and medium enterprises.

These maturity levels are composed by processes which define what the expected results are, and capabilities which express its institutionalization level and implementation in the organization. Thus, it is noteworthy that the development among these levels happens cumulatively and only when all demands were found.

III. RESEARCH METHODOLOGY

The research methodology used in this article was a case study. According to Yin [17], case studies offer an empirical research that investigates a contemporary phenomenon and offers researchers an object of applied study in its natural context. And, in addition, new facts and research issues about this environment can be identified [17].

In order to work on the case study, we selected a project of a software factory in a public university. Their teams were composed by undergraduate and master's students. Because of this, the company suffers with the seasonality issues in periods of academic activity, lack of commitment, interest and a low rate of productivity in its members.

Another problem of this company is the lack of a process of preservation of intellectual capital generated during the projects. Figure 1 show the development process used in the factory.

As shown in the Figure 1, the process begins with the macro activity, *Initial Analysis*, when the project's scope is defined through meetings with the customer. Then, in the macro activity *Analysis and Planning* it begins with the project planning and the record of its information in the Project Plan.

[2] Modelo de Referência para Melhoria do Software Brasileiro (MR-MPS)

[3] Associação para Promoção da Excelência do Software Brasileiro

[4] Programa para Melhoria do Processo do Software Brasileiro (MPS.Br).

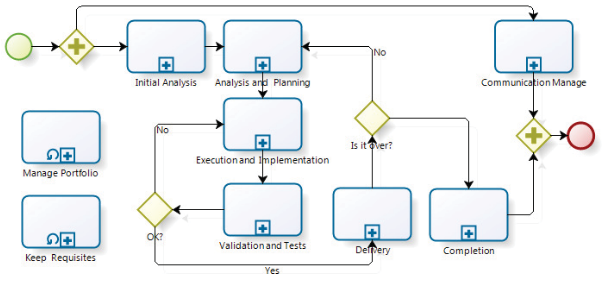


Fig. 1. Study Case Development Process

After this stage, it starts its execution and implementation and, subsequently, its validation and tests. During this stage, a direction is defined, either turning to the previous macro activity in order to correct inconsistencies and problems or, follow to the *Delivery* and *Project Completion*.

Unlike others, the macro activity *Keep Requisites* and *Manage Portfolio* occur in an asynchronous way and hold responsibility for ensuring the coherence of system's requisites and managing information between projects and top management, respectively.

Moreover, in each macro activities, a set of activities and artifacts, workflow of the works and tasks to be accomplished for each role of the process is defined.

IV. Process Model Proposed

Aiming the elaboration of the process model, we based on Amâncio et al. [1], Morais [11] and on the expected areas and results present on maturity level E of MR-MPS. The notation and elements used in the process modeling came from *Business Process Modeling Notation* (BPMN), since it is a notation language with standard icons for process design, which facilitates the understanding of the customer [4]. Figure 2 holds a more detailed model.

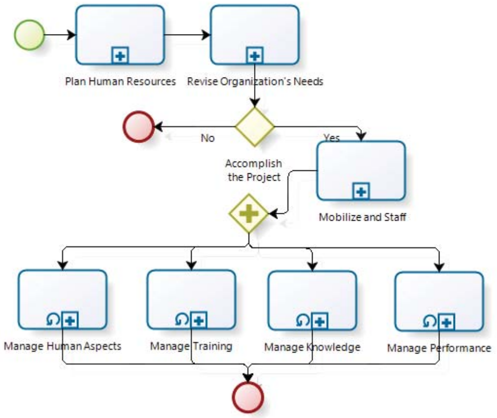


Fig. 2. Process Model for Human Resources Management

As shown in Figure 2, the process begins with the planning of the human resources. Then, it is conducted an analysis in domestic assets, in order to identify the partners who will be part of the project team. Based on this definition, it is carried out the recruitment and member's mobilization.

When the team is formed up and starts working on the accomplishment of the project, its members are constantly assessed towards knowledge, performance, training and human aspects, to minimize the problems and difficulties and maximize and improve the abilities. Next, we present each activity of the process.

A. *Human Resources Plan*

Accomplishing the human resources planning, from the very beginning of the project, has a big importance for its manager. The reason is easy: this stage will be defined and planned to identify factors that could influence the human resources management.

Its workflow begins with the definition of the needed resources for the accomplishment of the project. Subsequently, based on this definition, a set of four management policies[5] must be defined. Then the organizational chart is elaborated.

After its definition, in parallel, we must work on the development of the career planning and on the detailing of the pre requisites for roles allocation. Completed these two stages, all this information will be grouped and will compose the Human Resources Management Plan (HRMP).

B. *Review the Business Needs*

After planning, it is essential for the projects manager to work on the formation of their team. And, for that, they must base on the analysis of the organization's environmental factors (physical and social environments and people's attitude) and the organizational processes assets [1].

However, it is noteworthy that the revision of the organization's needs must encompass not only human needs, but also those needs connected to technical and support factors, such as: expenses with support people (e.g. accountants and administrators), travels, materials and trainings.

C. *Hire and Relocate Members*

The goal of this activity consists in identifying, from an organizational chart, which roles and attributes will be needed to the execution of the project. Based on this, abilities and pre requisites can be defined and used to help in the selection of the member of the team.

In order to form the team, it is needed that, based on roles and attributes defined on the organizational chart, the manager is able to choose between hiring new members or relocating internal members. After this definition, the training needs must be identified, defined and recorded in the Tactical Training Plan (TTP).

During the development of this activity, the project manager must have access to artifacts that present which are the knowledge level, skills, experiences and availability of the members of the organization, so these information will help not only on the definition of whom is going to be part of the team.

[5] Policies of Development, Knowledge, Availability and Communication Management.

D. Manage Training

When any training needs in the team members are identified, they must be carried out according to the rules and specifications defined on TTP.

However, during their accomplishment, it is important to constantly monitor and assess their members. So that, improvements in teaching infrastructure and new training will be identified and implemented in order to guarantee the increase in the abilities and capabilities of their members.

E. Manage Human Aspects

This activity has as purpose to identify the environmental and social factors which may influence the good development of the project.

For this, first of all, the needs of personal meetings, structural and geographical factors of the working environment and actions to minimize them must be defined. Subsequently, it must be defined the needs of personal meetings, exchange of staff and socialization needs.

All these factors must be constantly assessed, either through interviews or surveys, so the satisfaction and motivation levels of its members could be identified and improved. During this activity, a psychological can help to interpret data and collect information.

F. Manage Performance

Managing performance consists in constantly assess, formally or informally, the member's performance, during the execution of their tasks, based on the assessment criteria pre-defined in the HRMP.

This assessment is important, because people are not always able to develop what is expected from them, resulting in a discrepancy between the planning and the accomplishment. Due to this, when identified, it's important that the manager adopts actions to correct or minimize them.

For that, with the definition of the criteria used for the performance assessment, the manager identifies if there were any discrepancy between what was planned and what was accomplished. Then, from this, he will be able to evaluate reasons and suggest corrective solutions for these problems.

G. Knowledge Management

The knowledge management consists in adopting measures, techniques and tools to help the identification, retention and sharing of knowledge. This management focuses in to improve the quality and productivity in future projects or development [16].

This knowledge can be formed by a group of data generated throughout the projects, obtained from individuals, from the organization culture, organizational transformations and internal and external processes.

H. Artifacts and Positions

During the application of the project, a set of **artifacts** must be elaborated and kept during the project. These must be generated either during the execution of the activities. The proper and constant update of these will serve as a help for other activities or for the development of future projects.

Moreover, it's important define the **management roles**, which throughout the completion of the model have to ensure the proper accomplishment of the activities. Besides this they have to identify possible improvements in the processes and suggesting corrective measures.

A summary of these artifacts and the roles related to the activities is presented in Table I.

TABLE I. ARTIFACTS AND MANAGEMENT ROLES CONNECTED TO THE ACTIVITIES OF THE PROCESS MODEL

	Consumed Artifacts	**Generated Artifacts**	**Roles**
Human Resources Planning	1) Organizational Policies; 2) Scope Document.	1) Human Resources Management Plan (HRMP); 2) Organizational Chart; 3) Plan of Positions and Functions; 4) Career Plan.	1) Project Manager; 2) Human Resources Manager.
Organization's Needs	1) Project Requisites; 2) Organizational Structure; 3) Economic Conditions.	1) Organizational Chart; 2) Career Plan; (CP).	1) Project Manager; 2) Human Resources Manager.
Training Management	1) Tactical Training Plan (TTP).	1) Report of Effectiveness of Team Training.	1) Project Manager; 2) Human Resources Manager; 3) Course Coordinator.

As shown in Table I, for each activity, besides a set of artifacts, there is a set of roles that must be associated and can be used during its execution.

However, due to the diversity in project's context, it is worthy to highlight that the acceptance and utilization of these roles must to attend to their needs. For more details about this process, access http://www.gaia.uel.br/gaia_rh/.

V. RESULTS

In order to validate the process model, some performance indicators for information and data collection were defined and applied. Through the analysis of these sources, it was possible to identify its advantages and limitations. Next, the results obtained with this research are described.

The first indicator shows the variation in the rate of rework from the training sessions. This is because high levels of rework pose major problems during the development of a project.

Through training exercises, we try to eliminate them by improving the skills and knowledge of members. Thus, solving the rework, this metric helps the project manager to identify the level of effectiveness of training. Figure 3 show this indicator for the case study.

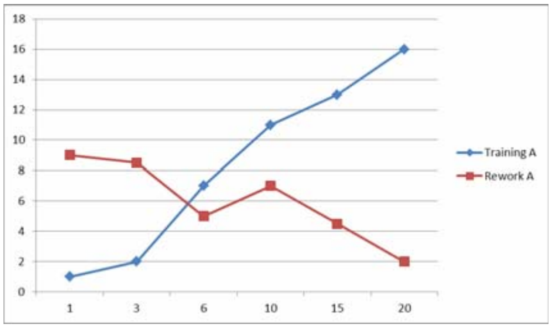


Fig. 3. Training Time vs. Rework Index

As can be observed in Figure 3, the training carried out with the team members have strong relationship with the improvement of your content to rework. After the implementation of the framework, this fact is evidenced by the decrease in 78% of this index in Project, that research contributes to the quality in the development process.

Besides this, the indicator for the analysis of improvement in the performance of members by conducting training, it is also important for improving quality in the development process. This is because the effectiveness of the performance actually contributes to the effectiveness of the members.

Thus, by conducting trainings, seeks to empower and qualify them so they can increase this indicator. Furthermore, through this measure, makes it possible to project manager to analyze the performance of its members and, if necessary, take steps to improve them. Figure 4 has the graphics prepared for analysis of this indicator.

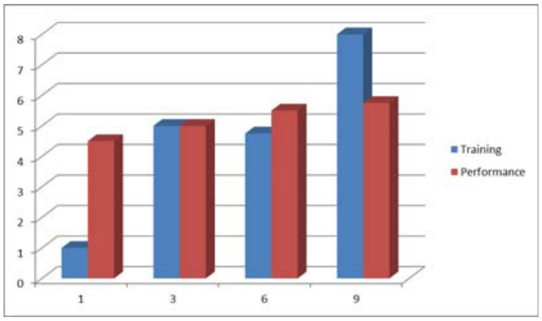


Fig. 4. Training Time vs. Performance Index

Also the rate of rework, the training carried out also maintains a strong relationship with the improvement in the performance index of members of the team. This fact strongly evidenced by analyzing the graphs shown in Figure 4.

Through them, it is noted that with the completion of training, implemented by the framework, an increase of 22% in performance of the project members. And that contributes not only to meet the deadline and measurement of the team, but also to improve the quality of coding, and especially the ease of maintenance.

Thus, in a general way, through the analysis of the performance indicators and collected information, the following advantages were identified:

- **Increasing member's motivation:** Main factor for the decrease in productivity and quality during the accomplishment of the tasks. Their motivation is constantly analyzed during the project, and actions for his improvement be identified, suggested and worked;
- **Improvement in the development process:** The continuous process to analysis and monitoring the knowledge, performance and members' abilities, aims to guarantee the allocation of the appropriate member for each task and with this increase the development;
- **Improvement in selection and allocation of members:** These activities are improved by the selection and utilization of members based on their performance, knowledge and stored experiences in the History Database (HD);
- **Decrease of member's turnover:** Aiming to keep the integrity and consistency of the team, problems with respect, inclusion, motivation and alignment of member's personal objectives with organization's objectives are identified and worked during the development of the project;
- **Increase of the organizational memory of the organization:** The storage of experiences, estimates, knowledge and performance of team's members, during the development of the projects, in the suggested HD, has as objective to keep this information available in the beginning of every project in order to facilitate the definition of the workforce;

Moreover, through continuous monitoring of performance and human aspects in projects' teams, it can be stated that the mentioned advantages have contributed significantly for the decrease of its member's turnover and for the increase in motivation and improvement of its activities development.

All these factors, besides contributing significantly for the application of the process model, also collaborates to the establishment of a specialist's network within the organization.

A. *Comparative Analysis*

In order to provide a comparative analysis between the models presented in Section 2a and the process model, Table II was elaborated with questions and activities from materials that generally cover staff management [3], as well as specific studies for the software development area [5], [12] [17], [18].

According to Table II, the process model is able to find all questions and activities defined for this analysis. We can observe the importance placed on the human resources present in the organization, the appreciation of the organization's intellectual capital from a specific level (individual) to a more organizational one (company) and the work of monitoring and evaluation performed on the human aspects of its members.

VI. Conclusions and Future Works

Analyzing the results obtained during the case study development, we can evaluate the success in the implementation of the process model. It is highlighted, mostly, the increase in motivation of members, their skills and capabilities, resulting in a significantly improvement in its development process and decrease in member's turnover.

Thus, focused on increase these human factors, the process model presented was developed to attend, provide and add

TABLE II. COMPARISON BETWEEN THE EXISTENT MODELS AND THE PROCESS MODEL

	Amâncio *et al*.	**De Carvalho**	**P-CMM**	**Morais**	**Process Model**
Planning Human Resources	Yes, during the initial phase of the project.	Yes, realized in the beginning of the project.	Yes, defined in the initial phase of the model.	No.	Yes, it is work in the beginning of the project.
Revise Organization's Needs	Yes, approached superficially during the planning.	Yes, it uses a repository of knowledge and abilities.	No.	Yes, revised based on organizational factors.	Yes, it uses organizational factors.
Opt for Internal Mobilization to External Staffing	N/A	N/A	N/A	N/A	Yes.
Work the Mobilization and Staffing	No, the model only defines functions and timesheets.	Yes, performed based on profiles of competences demanded for the project.	Yes, performed based on project's needs and availability in the organization.	Yes, use an interview database and a recruitment process.	Yes, performed based on setting of the organizational chart.
Training Management	High. It has a specific activity and an action and monitoring plan.	High. It defines and constantly monitors.	High. It is worked during the development of the model.	High. Proposes a continuous process of improvement.	High. Worked on according to the project's needs.
Performance Management	High. Definition and monitoring of indicators.	High. It monitors with project repository.	High. Its analysis is realized during the whole model.	High. Monitoring using predefined indicators.	High. Constant monitoring during the project.
Organizational Knowledge Management	No.	No.	No.	Yes, the knowledge is identified, storage and sharing with the company.	Yes, focused in the individual sharing with the company.
Uses Historical Database	No.	Yes, it has an organizational and a project one.	Yes, used in all phases of the project.	No.	Yes, used in all phases of the project.
Human Aspects Management	No.	No.	Yes.	No.	Yes, monitored during the development of the project.

more value to human resources of software project through planning and continuous development of his team's members.

Finally, as presented in Table II, this process model differs from other existing process models in literature, with the appreciation of the intellectual capital of the organization and the work of monitoring and evaluation performed on the human aspects of its members.

As future study suggestion is to development a tool to help in planning, recruitment and selection, monitoring, development and analysis of human resources during the development and project planning.

REFERENCES

[1] Amâncio, S. F., Costa, H. A. X., De Camargo, V. and Penteado, R. A. D, "Gerência de recursos humanos para uma fábrica de software de pequeno porte". In *X Workshop Um Olhar Sóciotécnico sobre a Engenharia de Software*, Ouro Preto, Brazil, 2008.

[2] André, M., Badoquín, M. G. and Acuña, S. T, "Formal model for assigning human resources to teams in software projects". *Information and Software Technology* 53, 2011, pp. 259-275.

[3] Chiavenato, I, Gestão de Pessoas: e o novo papel dos recursos humanos nas organizações. 2nd ed, Rio de Janeiro, Elsevier, 2004.

[4] Cruz, T, BPM & BPMS: Business Process Management & Business Process. 2nd ed, Rio de Janeiro, Brasport, 2010.

[5] Cibotto, R. A. G., Tait, T. F. C., Malucelli, A. and Reinher, S, "O fator humano no desenvolvimento distribuído de software". In *VII Workshop Um Olhar Sociotécnico sobre a Engenharia de Software*, Curitiba, Brazil, 2011.

[6] Curtis, B., Hefley, B. and Miller, S, P-CMM: People Capability Maturiry Model. *Software Engineering Institute*, June, 2009.

[7] De Carvalho, L. R, Planejamento da alocação de recursos humanos em ambientes de desenvolvimento de software orientados à organização, Master Thesis, Universidade Federal do Rio de Janeiro, COPPE, Brazil, 2003.

[8] Denning, P. and Riehle, R, "The Profession of IT: Is Software Engineering Engineering?". *Commun. ACM 52*, 2009, pp. 24-26.

[9] Hazzan, O. and Hadar, I, "Why and how can human-related measures support software development processes?". *The Journal of Systems and Software*, 2008, pp 1248-1252.

[10] MR-MPS (Modelo de Referência para Melhoria de Processo do Software Brasileiro). Associação para Promoção da Excelência do Software Brasileiro, August, 2011.

[11] Morais, S. R. G., Uma abordagem para a gerência de recursos humanos de organizações de software, Master Thesis, Universaidade de Fortaleza, Brazil, 2009.

[12] Qiu. Y., "Human resources management based on human capital in enterprises". In *International Conference on Management and Service Science*, Wuhan, China, 2011.

[13] Reis, K., "Fatores Humanos: A influência na qualidade de software". *Revista Engenharia de Software*, ed. 18, 2, 2009, pp. 24-29.

[14] Shan, X. Jiang and G. Huang, T., "The optimization research on the human resource allocation planning in software projects". In *International Conference on Management and Service Science*, Wuhan, China, 2010.

[15] Tohidi, H., "Human resources management main role in information technology project management". In *World Conference on Information Technology*, Antalya, Turkey, 2011.

[16] Wei, S., Qingpu, Z., and Chen, L., "The model of knowledge integration management for IT corporation and its operating mechanism," in Information Management and Engineering (ICIME), 2010 The 2nd IEEE International Conference on, 2010, pp. 85–89.

[17] Yin, R. K, Case Study Research: Design and Method, Third Edition, Applied Social Research Methods Series, v. 5, Sage Publications, Inc, 2002.

Verification of Cyber-Physical Systems Based on Differential-Algebraic Temporal Dynamic Logic

Xiaoxiang Zhai, Bixin Li, Min Zhu, Jiakai Li, Qiaoqiao Chen, Shunhui Ji

School of Computer Science and Engineering, Southeast University, Nanjing, China
Email: {xxzhai, bx.li}@seu.edu.cn, kongs@139.com, jiakai_li@seu.edu.cn,
joe_0701@126.com, shunhuiji@163.com

Abstract

Differential temporal dynamic logic (dTL) is an approach for specifying and verifying properties of cyber-physical systems (CPS) and it can handle with temporal behaviors for CPS. The hybrid programs (HP), as operating model of dTL, only contain differential equations that can be solved in polynomial arithmetic, which results that dTL can only specify and verify CPS of simple dynamics. However, differential-algebraic dynamic logic (DAL) solves the problem through the introduction of differential invariants, but lacks verification capabilities for properties with temporality. This paper combines the advantages of dTL and DAL, and proposes differential-algebraic temporal dynamic logic (DATL). We have achieved the following results: a trace semantics for differential-algebraic programs (DAP), four new rules based on the rules of dTL and DAL, a proof of the soundness of the new rules, and the specification and verification of safety of aircraft collision avoidance system with DATL. Our theory together with a case study demonstrates that DATL overcomes the constraints that differential equations must be solvable in polynomial arithmetic and can be used to specify and verify temporal properties of CPS.

***Keywords*—Cyber-physical systems; property verification; differential temporal dynamic logic; differential-algebraic dynamic logic; differential-algebraic temporal dynamic logic; aircraft collision avoidance system;**

I. Introduction

Cyber-physical systems (CPS) are integrations of computation and physical processes. Embedded computers and networks monitor and control the physical processes, usually with feedback loops where physical processes affect computations and vice versa [1]. Many approaches have been proposed to verify properties of CPS. They are primarily divided into two categories: model checking and theorem proving. Because CPS do not admit equivalent finite-state abstractions [2] and due to general limits of numerical approximation, model checkers are still more successful in falsification than in verification. Differential temporal dynamic logic (dTL) and differential-algebraic dynamic logic (DAL) are approaches falling the scope of theorem proving.

As operating model of dTL, HP have limited expressiveness and cannot model complex CPS. For example, fluctuations and errors in the physical processes cannot be expressed. As operating model of DAL, DAP make up for the shortcomings of HP via the introduction of quantifiers and differential invariants. However, DAL cannot verify properties with temporality, and dTL has the ability by introducing temporal operator and expanding the relevant calculus rules. This paper combines the advantages of dTL and DAL and proposes differential-algebraic temporal dynamic logic (DATL) to verify properties of CPS with complex dynamics and temporality.

This paper is organized as follows: next section first introduces an aircraft collision avoidance system, then defines trace semantics of DAP under temporal behavior. The aircraft collision avoidance system is modeled using DAP and safety of the system is specified as a DATL formula. Section 3 introduces four new calculus rules for DATL by inheritance, expansion and improvement of the rules of the DAL and dTL, and proves the correctness of the new rules. The property of the aircraft collision avoidance system is specified as a DATL formula and we use DATL sequent calculus to formally verify the property. Finally, a conclusion of research in this paper is drawn and some expectations are brought forward for the future.

II. CPS Modeling and Property Specification

A. Aircraft Collision Avoidance System

There are a number of aircrafts flying in the air, their flight dynamics can be described by a group of differential equations. The differential equations of $(*)$ denote flight dynamics of any two aircrafts X and Y, where point $x(x_1, x_2, x_3)$ and point $y(y_1, y_2, y_3)$ denote three-dimensional coordinates of aircrafts X and Y respectively, $d(d_1, d_2, d_3)$ and $e(e_1, e_2, e_3)$ denote speed of

This work is supported partially by National Natural Science Foundation of China under Grant No. 60973149, partially by the Open Funds of State Key Laboratory of Computer Science of Chinese Academy of Sciences under Grant No.SYSKF1110, partially by Doctoral Fund of Ministry of Education of China under Grant No. 20100092110022, and partially by the College Industrialization Project of Jiangsu Province under Grant No.JHB2011-3.

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

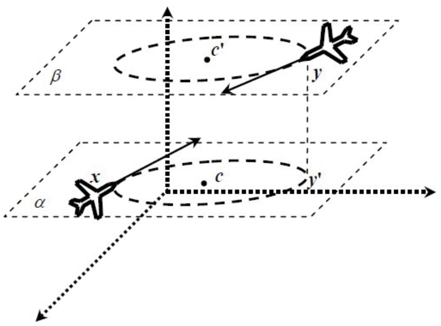


Fig. 1. Aircraft Collision Avoidance System

X and Y along x-axis, y-axis and z-axis respectively, ω and ϖ denote angular velocity of X and Y in horizontal direction, and a, b denote acceleration of X and Y in vertical direction respectively.

$$(*)\begin{cases} x_1{}' = d_1 \wedge x_2{}' = d_2 \wedge x_3{}' = d_3 \wedge d_1{}' = -\omega d_2 \wedge d_2{}' = \\ \quad \omega d_1 \wedge d_3{}'{=}a \wedge a' = 0 \quad (\mathscr{F}(\omega, d_3, a)) \\ y_1{}' = e_1 \wedge y_2{}' = e_2 \wedge y_3{}' = e_3 \wedge e_1{}' = -\varpi e_2 \wedge e_2{}' = \\ \quad \varpi e_1 \wedge e_3{}'{=}b \wedge b' = 0 \quad (\mathscr{G}(\varpi, e_3, b)) \end{cases}$$

If the distance between aircrafts X and Y at some time is less than some value p, it should be considered that X and Y have the risk of collision. At this point, X and Y should adopt a strategy to avoid the collision. Platzer [3] proposed an aircraft collision avoidance system of two-dimensional case, we extended it into the three-dimensional case shown in Figure 1. As the coordinate of X is point $x(x_1, x_2, x_3)$, and through the point x, there exists only one plane α which is parallel to the ground. By the same argument, through the point $y(y_1, y_2, y_3)$, there is only one plane β which is parallel to the ground. We can get point y' in plane α by projecting Y on α (ignoring the size of the aircraft itself), then there must exist a circle (noted as *circle*1) whose center is $c(c_1, c_2, x_3)$ and in which point x and point y' stay, as shown in Figure 1. Similarly, there exists another circle (noted as *circle*2) whose center is $c'(c_1, c_2, y_3)$ and in which point y stays. The aircraft collision avoidance system can be described as follows: when the distance between aircraft X and Y is less than some value p, X and Y respectively change their flight direction into the tangential direction in point x and y of *circle*1 and *circle*2 in the plane α and β. And then X and Y fly several time in accordance with dynamics described by differential equations shown in (#) (Note: during the entire collision avoidance process, the height of aircrafts is constant, in (#), differential equations describing height are omitted). The aircrafts entered free flight after the collision avoidance process is completed. If the risk of collision happens again, the same collision avoidance strategy should be adopted.

$$(\#)\begin{cases} x_1{}' = d_1 \wedge x_2{}' = d_2 \wedge d_1{}' = -\omega d_2 \wedge d_2{}' = \omega d_1 \\ y_1{}' = e_1 \wedge y_2{}' = e_2 \wedge e_1{}' = -\omega e_2 \wedge e_2{}' = \omega e_1 \end{cases}$$

B. Trace Semantics of DAP

DATL introduces DAP as its operating model. Because the semantics of state transition of DAP under temporal behavior changes, it is necessary to redefine the semantics of state transition of DAP under temporal behavior , which is given as follows:

Definition 1 (Trace Semantics): The trace semantics, $\tau(\alpha)$, of a DAP α, is the set of all its possible hybrid traces and is defined inductively as follows:

1. $(\hat{v}, \hat{\omega}) \in \tau(\mathscr{J})$ iff $(\hat{v}, \hat{\omega}) \models \mathscr{J}$, where $\mathscr{J}$ denotes DJ-constraint, and according to different structures of $\mathscr{J}$, several cases should be distinguished as follows:

1) $(\hat{v}, \hat{\omega}) \models x := \theta$ iff $val(\omega, x) = val(v, \theta)$.
2) $(\hat{v}, \hat{\omega}) \models \theta_1 \geq \theta_2$ iff $val(v, \theta_1) \geq val(v, \theta_2)$.
3) $(\hat{v}, \hat{\omega}) \models \phi \wedge \psi$ iff $(\hat{v}, \hat{\omega}) \models \phi$ and $(\hat{v}, \hat{\omega}) \models \psi$.
4) $(\hat{v}, \hat{\omega}) \models \phi \vee \psi$ iff $(\hat{v}, \hat{\omega}) \models \phi$ or $(\hat{v}, \hat{\omega}) \models \psi$.
5) $(\hat{v}, \hat{\omega}) \models \neg\phi$ iff it is not the case that $(\hat{v}, \hat{\omega}) \models \phi$.
6) $(\hat{v}, \hat{\omega}) \models \phi \to \psi$ iff it is not the case that $(\hat{v}, \hat{\omega}) \models \phi$ or it is the case that $(\hat{v}, \hat{\omega}) \models \psi$.
7) $(\hat{v}, \hat{\omega}) \models \forall x\phi$ iff $(\hat{v}_x, \hat{\omega}) \models \phi$ for all states v_x that agree with v except for the value of x.
8) $(\hat{v}, \hat{\omega}) \models \exists x\phi$ iff $(\hat{v}_x, \hat{\omega}) \models \phi$ for some state v_x that agrees with v except for the value of x.

2. $\tau(\mathscr{D}) = \{(\bar{\varphi}) : \bar{\varphi}$ is a differentially augmented state flow and some duration $r \geq 0$ such that for all $\zeta \in [0, r]$, $\bar{\varphi}(\zeta) \models \mathscr{D}$, and $val(\bar{\varphi}(\zeta), z) = val(\bar{\varphi}(0), z)$ for all variables z that are not changed by $\mathscr{D}$, where $\mathscr{D}$ denotes DA-constraint$\}$.

3. $\tau(\alpha \cup \beta) = \tau(\alpha) \cup \tau(\beta)$.

4. $\tau(\alpha; \beta) = \{\sigma \circ \varsigma : \sigma \in \tau(\alpha), \varsigma \in \tau(\beta)$ when $\sigma \circ \varsigma$ is defined$\}$, the composition of $\sigma = (\sigma_0, \sigma_1, \sigma_2, \ldots)$ and $\varsigma = (\varsigma_0, \varsigma_1, \varsigma_2, \ldots)$ is

$$\sigma \circ \varsigma = \begin{cases} (\sigma_0, \ldots, \sigma_n, \varsigma_0, \varsigma_1, \ldots) & \text{if } \sigma \text{ terminates at } \sigma_n \text{ and} \\ & \quad \text{last } \sigma = \text{ first } \varsigma \\ \sigma & \text{if } \sigma \text{ does not terminate} \\ \text{not defined} & \text{otherwise} \end{cases}$$

5. $\tau(\alpha*) = \cup_{n\in\mathbb{N}}\tau(\alpha^n)$, where $\alpha^{n+1} = (\alpha^n; \alpha)$ for $n \geq 1$, as well as $\alpha^1 = \alpha$ and $\alpha^0 = (?\text{true})$.

C. Modeling of Aircraft Collision Avoidance System

We use DAP to model the whole process of collision avoidance, and the model is $trm*$, where $*$ denotes that the execution of trm which is described as $(\sim)$ is repeated 0 time or at least one time. The trm consists of three phases: free flight phase (described with $free$), changing phase of heading(assuming that the phase is completed at once, without considering the delay, described with $tang$) and flight limiting phase (aircrafts must meet dynamic $\mathscr{F}(\omega, 0, 0) \wedge \mathscr{G}(\omega, 0, 0)$ relative to the free flight phase). In trm , ϕ means the distance between X and Y is greater than or equal to p, $d := \omega(x - c)^{\perp}$ is a shorthand for $d_1 := -\omega(x_2 - c_2), d_2 := \omega(x_1 - c_1)$, and $e := \omega(y - c)^{\perp}$ is a shorthand for $e_1 := -\omega(y_2 - c_2), e_2 := \omega(y_1 - c_1)$.

$$(\sim)\begin{cases} trm \equiv free; tang; \mathscr{F}(\omega, 0, 0) \wedge \mathscr{G}(\omega, 0, 0) \\ free \equiv \exists\omega\exists a\mathscr{F}(\omega, d_3, a) \wedge \exists\varpi\exists b\mathscr{G}(\varpi, e_3, b) \wedge \phi \\ \phi \equiv (x_1 - y_1)^2 + (x_2 - y_2)^2 + (x_3 - y_3)^2 \geq p^2 \\ tang \equiv \exists u\omega := u; \exists c(d := \omega(x - c)^{\perp} \wedge e := \omega(y - c)^{\perp}) \end{cases}$$

D. Safety Specification of Aircraft Collision Avoidance System

We want to verify the property: aircrafts are safe at any time, i.e., any time that aircrafts will not collide. This property has temporal characteristics, that is, any time. DAL apparently cannot specify this property of the CPS with temporal characteristics. This paper specifies the property with DATL that combined with the advantages of modeling complex CPS in DAL and specifying properties of CPS with temporal behaviors in dTL, as follows:

We have used DAP to model aircraft collision avoidance system and got an operating model $trm*$. Here we will use DATL formula to specify safety property of the system, we specify the property "the aircrafts will not collide at any time" as a DATL formula: $\psi \equiv \phi \rightarrow [trm*]\Box\phi$, this formula denotes that ϕ implies $[trm*]\Box\phi$. $[trm*]\Box\phi$ denotes that ϕ is satisfied in any state of any execution path of the system. That is, the formula ψ denotes that if the distance between the two aircrafts is greater than or equal to p, the constraint is satisfied at any time in the execution of $trm*$.

III. Property Verification of CPS

So far, we have used DAP to model aircraft collision avoidance system, used DATL formula to specify safety of the system and got the formula $\psi \equiv \phi \rightarrow [trm*]\Box\phi$. Based on the definition of trace semantics of DAP, we add four new rules, prove the soundness of them, and then use DATL sequent calculus to verify the formula.

A. New Rules and Proof

We add four new rules to DATL as follows:

$$([\mathscr{J}]\Box)\frac{\phi \wedge [\mathscr{J}]\phi}{[\mathscr{J}]\Box\phi} \qquad (\langle\mathscr{J}\rangle\Diamond)\frac{\phi \vee \langle\mathscr{J}\rangle\phi}{\langle\mathscr{J}\rangle\Diamond\phi}$$

$$([\mathscr{D}]\Box)\frac{[\mathscr{D}]\phi}{[\mathscr{D}]\Box\phi} \qquad (\langle\mathscr{D}\rangle\Diamond)\frac{\langle\mathscr{D}\rangle\phi}{\langle\mathscr{D}\rangle\Diamond\phi}$$

Other rules and proof calculus in dTL and DAL are maintained in DATL. We call any formula ϕ provable in the DATL if we can find a DATL proof for it that starts with axioms (rule ax) at the leaves and ends with a sequent $\vdash \phi$ at the bottom. While constructing proofs, however, we would start with the desired goal $\vdash \phi$ at the bottom and work our way backwards to the subgoals until they can be proven to be valid as axioms (ax). Once all subgoals have been proven to be valid axioms, they entail their consequences, which, recursively, entail the original goal $\vdash \phi$. Thus, while constructing proofs, we work bottom-up from the goal. When we have found a proof, we justify formulas from the axioms top-down to the original goal.

Next, we prove the soundness of rules $[\mathscr{J}]\Box$ and $[\mathscr{D}]\Box$, and the proof of other two rules are similar.

(1) $([\mathscr{J}]\Box)\ \frac{\phi\wedge[\mathscr{J}]\phi}{[\mathscr{J}]\Box\phi}$

Proof: It is proved inductively according to the structure of $\mathscr{J}$.

1) when $\mathscr{J}$ is the form $x := \theta$, we want to prove $\frac{\phi\wedge[x:=\theta]\phi}{[x:=\theta]\Box\phi}$, and it has been proved in dTL [3].

2) when $\mathscr{J}$ is the form $\theta_1 \geq \theta_2$, we want to prove $\frac{\phi\wedge[\theta_1\geq\theta_2]\phi}{[\theta_1\geq\theta_2]\Box\phi}$, i.e., assume that ϕ and $[\theta_1 \geq \theta_2]\phi$ are satisfied, i.e., current state $v \models \phi$ and $v \models [\theta_1 \geq \theta_2]\phi$, then prove that $[\theta_1 \geq \theta_2]\Box\phi$ is also satisfied. Let $\forall\sigma \in \tau(\theta_1 \geq \theta_2)$, then the conclusion needed to prove is equal to $\sigma \models \Box\phi$, and $\sigma = \{(\hat{v}) : val(v, \theta_1) \geq val(v, \theta_2)\} \cup \{(\hat{v}, \hat{\Lambda}) : val(v, \theta_1) < val(v, \theta_2)\}$. If $val(v, \theta_1) \geq val(v, \theta_2)$, $\sigma = \{(\hat{v})\}$ and the conclusion becomes $\{(\hat{v})\} \models \Box\phi$, i.e., $v \models \phi$. To have already known, $v \models \phi$ is satisfied, then proof finishes. And if $val(v, \theta_1) < val(v, \theta_2)$, $\sigma = \{(\hat{v}, \hat{\Lambda})\}$. Then conclusion satisfies obviously. And accordingly $\theta_1 = \theta_2, \theta_1 \leqslant \theta_2, \theta_1 < \theta_2, \theta_1 > \theta_2$.

3) when $\mathscr{J}$ is the form $\varphi \wedge \psi$, we want to prove $\frac{[\varphi]\Box\phi\wedge[\psi]\Box\phi}{[\varphi\wedge\psi]\Box\phi}$. According to the third point of trace semantics of DAP, we can conclude that $(\hat{v}, \hat{\omega}) \models \varphi \wedge \psi$ iff $(\hat{v}, \hat{\omega}) \models \varphi$ and $(\hat{v}, \hat{\omega}) \models \psi$, i.e., $(\hat{v}, \hat{\omega}) \in \tau(\varphi\wedge\psi)$ iff $(\hat{v}, \hat{\omega}) \in \tau(\varphi)$ and $(\hat{v}, \hat{\omega}) \in \tau(\psi)$, i.e., $[\varphi\wedge\psi]\Box\phi$ iff $[\varphi]\Box\phi \wedge [\psi]\Box\phi$.

4) when $\mathscr{J}$ is the form $\varphi \vee \psi$, we want to prove $\frac{[\varphi]\Box\phi\vee[\psi]\Box\phi}{[\varphi\vee\psi]\Box\phi}$. The proof is similar with 3).

5) when $\mathscr{J}$ is the form $\neg\psi$, we want to prove $\frac{\neg([\psi]\Box\phi)}{[\neg\psi]\Box\phi}$. According to the fifth point of trace semantics of DAP, we can conclude that $(\hat{v}, \hat{\omega}) \models \neg\psi$ iff $\neg((\hat{v}, \hat{\omega}) \models \psi)$, i.e., $(\hat{v}, \hat{\omega}) \in \tau(\neg\psi)$ iff $\neg((\hat{v}, \hat{\omega}) \in \tau(\psi))$, i.e., $[\neg\psi]\Box\phi$ iff $\neg([\psi]\Box\phi)$.

6) when $\mathscr{J}$ is the form $\varphi \rightarrow \psi$, we want to prove $\frac{[\varphi]\Box\phi\rightarrow[\psi]\Box\phi}{[\varphi\rightarrow\psi]\Box\phi}$. According to the sixth point of trace semantics of DAP, we can conclude that $(\hat{v}, \hat{\omega}) \models \varphi \rightarrow \psi$ iff $\neg((\hat{v}, \hat{\omega}) \models \varphi) \vee ((\hat{v}, \hat{\omega}) \models \psi)$, i.e., $(\hat{v}, \hat{\omega}) \in \tau(\varphi \rightarrow \psi)$ iff $\neg((\hat{v}, \hat{\omega}) \in \tau(\varphi)) \vee ((\hat{v}, \hat{\omega}) \in \tau(\psi))$, i.e., $(\hat{v}, \hat{\omega}) \in \tau(\varphi \rightarrow \psi)$ iff $((\hat{v}, \hat{\omega}) \in \tau(\varphi)) \rightarrow ((\hat{v}, \hat{\omega}) \in \tau(\psi))$, i.e., $[\varphi \rightarrow \psi]\Box\phi$ iff $[\varphi]\Box\phi \rightarrow [\psi]\Box\phi$.

7) when $\mathscr{J}$ is the form $\forall x\varphi$, we want to prove $\frac{[\varphi_{\{x:=\forall\theta\}}]\Box\phi}{[\forall x\varphi]\Box\phi}$, where $\varphi_{\{x:=\forall\theta\}}$ means x in φ is assigned to all possible value θ. According to the seventh point of trace semantics of DAP, we can conclude that $(\hat{v}, \hat{\omega}) \models \forall x\varphi$ iff $(\hat{v}_x, \hat{\omega}) \models \varphi$ for all states v_x that agree with v except for the value of x, i.e., $(\hat{v}, \hat{\omega}) \models \forall x\varphi$ iff $(\hat{v}, \hat{\omega}) \models \varphi_{\{x:=\forall\theta\}}$, i.e., $[\forall x\varphi]\Box\phi$ iff $[\varphi_{\{x:=\forall\theta\}}]\Box\phi$.

8) when $\mathscr{J}$ is the form $\exists x\varphi$, we want to prove $\frac{[\varphi_{\{x:=\exists\theta\}}]\Box\phi}{[\exists x\varphi]\Box\phi}$, where $\varphi_{\{x:=\exists\theta\}}$ means x in φ is assigned to some value θ. The proof is similar with 7).

Proof finished. #

(2) $([\mathscr{D}]\Box)\frac{[\mathscr{D}]\phi}{[\mathscr{D}]\Box\phi}$

Proof: assuming current state $v \not\models [\mathscr{D}]\Box\phi$, there exists a trace $\sigma = (\bar{\varphi}) \in \tau(\mathscr{D})$, first $\sigma = v$, $\sigma \not\models \Box\phi$, and $\forall\zeta \in [0, r]$, $\bar{\varphi}(\zeta) \models \mathscr{D}$. According to $\sigma \not\models \Box\phi$, there exists a *position* $(0, \eta), \eta \in [0, r]$, $\sigma_0(\eta) \not\models \phi$. For $\bar{\varphi}$ defined in $[0, \eta]$, $\bar{\varphi} \models \mathscr{D}$ is always true. Further, $\because\ \bar{\varphi}(\eta) \not\models \phi$, $\therefore\ (\bar{\varphi}|_{[0,\eta]}) \not\models \phi$, $\therefore\ v \not\models [\mathscr{D}]\phi$, $\therefore\ v \not\models [\mathscr{D}]\Box\phi \Rightarrow v \not\models [\mathscr{D}]\phi$, $\therefore\ v \models [\mathscr{D}]\phi \Rightarrow v \models [\mathscr{D}]\Box\phi$.

Proof finished. #

Therefore, we can conclude that DATL is sound, i.e., all formulas which can be proved with DATL are valid in all states of all interpretations. However, like other logics such as dL, DAL and dTL, DATL is incomplete, i.e., valid DATL formulas are not always provable.

B. Verification of Safety of Aircraft Collision Avoidance System

Here we use DATL sequent calculus to prove the formula ψ shown in Figure 2 , we write the formula ψ as the form $\vdash \phi \rightarrow$

$$
\to r\;\dfrac{[*]\Box\;\dfrac{[;]\;\dfrac{[\mathscr{D}]\Box\;\dfrac{[;]\;\dfrac{[]gen\;\dfrac{[DR']\;\dfrac{r\forall\;\dfrac{\dfrac{*}{\phi \vdash [trm^*](true)}}{\phi \vdash [trm^*](\forall\omega\forall\varpi\forall x\forall y\forall d\forall e(\phi \to \phi))}}{\phi \vdash [trm^*][free]\phi} \qquad []gen\;\dfrac{\dfrac{\cdots}{\phi \vdash [tang](\phi \wedge \mathscr{T})} \qquad \dfrac{\cdots}{\phi \wedge \mathscr{T} \vdash [\mathscr{F}(\omega,0,0) \wedge \mathscr{G}(\omega,0,0)]\phi}}{\phi \vdash [tang; \mathscr{F}(\omega,0,0) \wedge \mathscr{G}(\omega,0,0)]\phi}}{\phi \vdash [trm^*][free][tang; \mathscr{F}(\omega,0,0) \wedge \mathscr{G}(\omega,0,0)]\phi}}{\phi \vdash [trm^*][free][tang][\mathscr{F}(\omega,0,0) \wedge \mathscr{G}(\omega,0,0)]\phi}}{\phi \vdash [trm^*][free][tang][\mathscr{F}(\omega,0,0) \wedge \mathscr{G}(\omega,0,0)]\Box\phi}}{\phi \vdash [trm^*][trm]\Box\phi}}{\phi \vdash [trm^*]\Box\phi}}{\vdash \phi \to [trm^*]\Box\phi}
$$

Fig. 2. Calculus Process (Part 1)

$$
[;],[\exists],[\exists]\;\dfrac{r\forall,r\forall\;\dfrac{[:=]\;\dfrac{\wedge r\;\dfrac{ax\;\dfrac{*}{\phi \vdash \phi} \qquad \dfrac{*}{\phi \vdash \omega(x-c)^{\perp} - \omega(y-c)^{\perp} = \omega(x-y)^{\perp}}}{\phi \vdash \phi \wedge \omega(x-c)^{\perp} - \omega(y-c)^{\perp} = \omega(x-y)^{\perp}}}{\phi \vdash [d := \omega(x-c)^{\perp} \wedge e := \omega(y-c)^{\perp}](\phi \wedge \mathscr{T})}}{\phi \vdash \forall\omega\forall c[d := \omega(x-c)^{\perp} \wedge e := \omega(y-c)^{\perp}](\phi \wedge \mathscr{T})}}{\phi \vdash [tang](\phi \wedge \mathscr{T})}
$$

Fig. 3. Calculus Process (Part 2)

$$
\wedge l\;\dfrac{DS\;\dfrac{DI\;\dfrac{r\forall\;\dfrac{*}{\vdash \forall^{\alpha}(\mathscr{T}'_{\mathscr{F}(\omega,0,0)\wedge\mathscr{G}(\omega,0,0)})}}{\phi, \mathscr{T} \vdash [\mathscr{F}(\omega,0,0) \wedge \mathscr{G}(\omega,0,0)]\mathscr{T}} \qquad DI\;\dfrac{r\forall\;\dfrac{*}{\vdash \forall^{\alpha}(\mathscr{T} \to \phi'_{\mathscr{F}(\omega,0,0)\wedge\mathscr{G}(\omega,0,0)})}}{\phi \vdash [\mathscr{F}(\omega,0,0) \wedge \mathscr{G}(\omega,0,0) \wedge \mathscr{T}]\phi}}{\phi, \mathscr{T} \vdash [\mathscr{F}(\omega,0,0) \wedge \mathscr{G}(\omega,0,0)]\phi}}{\phi \wedge \mathscr{T} \vdash [\mathscr{F}(\omega,0,0) \wedge \mathscr{G}(\omega,0,0)]\phi}
$$

Fig. 4. Calculus Process (Part 3)

$[trm*]\Box\phi$ which is suit for sequent calculus, then use the rule $\to r$ to get $\phi \vdash [trm*]\Box\phi$, and use the rule $[*]\Box$ and $[;]$ to get $\phi \vdash [trm*][free][tang][\mathscr{F}(\omega,0,0) \wedge \mathscr{G}(\omega,0,0)]\Box\phi$. Because $\mathscr{F}(\omega,0,0) \wedge \mathscr{G}(\omega,0,0)$ is DA-Constraint, we use the rule $[\mathscr{D}]\Box$ to get $\phi \vdash [trm*][free][tang][\mathscr{F}(\omega,0,0) \wedge \mathscr{G}(\omega,0,0)]\phi$. Similarly, we can get $\phi \vdash [trm*](true)$ by using rules $[;]$, $[]gen$, $[DR']$, $r\forall$, and the formula is satisfied obviously, so the calculus of the left branch of Figure 2 ends, which marked with $*$. Meanwhile, the right branch of Figure 2 $\phi \vdash [tang; \mathscr{F}(\omega,0,0) \wedge \mathscr{G}(\omega,0,0)]\phi$ uses the rule $[]gen$ to get two branches: $\phi \vdash [tang](\phi \wedge \mathscr{T})$ and $\phi \wedge \mathscr{T} \vdash [\mathscr{F}(\omega,0,0) \wedge \mathscr{G}(\omega,0,0)]\phi$, where $\mathscr{T}$ is differential invariant that can be calculated through fix-point algorithms [4] and we get the result $\mathscr{T} \equiv d - e = \omega(x-y)^{\perp} \equiv d_1 - e_1 = -\omega(x_2 - y_2) \wedge d_2 - e_2 = \omega(x_1 - y_1)$.

Figure 3 and Figure 4 is a continuance of Figure 2. Similar with Figure 2, rules $[;]$, $[\exists]$, $r\forall$, $[:=]$, $\wedge r$, ax, $\wedge l$, DS and DI have been used in Figure 3 and Figure 4 and end with $*$.

In summary, the calculus process above prove the correctness of the formula $\psi \equiv \phi \to [trm*]\Box\phi$, i.e., it succeeds in verifying the safety of the aircraft collision avoidance system. But if some calculus process does not end with $*$, we consider that the corresponding property cannot be verified with DATL, which is determined by the incompleteness of DATL.

IV. Conclusion and Future Work

In this paper, we have extended dTL and DAL to achieve DATL. Accordingly, we defined trace semantics of DAP, modified four rules of dTL into new rules of DATL and proved soundness of new rules. In addition, we specified and verified an aircraft collision avoidance system, where we introduced temporality into the model of the system and the property specification with DATL compared with the work of Platzer [3]. The contributions of this paper are:

(1) compared to the DAL, DATL can specify and verify properties with temporality;

(2) compared to dTL, DATL can model more complex CPS;

(3) compared to the two-dimensional aircraft collision avoidance system in [3], this paper presents a three-dimensional avoidance system and considers the temporal behavior when specifying and verifying the safety of the system.

In the future, we will extend the verification tool KeYmaera which is developed by Platzer and Quesel [5], so that it can verify DATL formulas. We will also propose more rules to enhance the verification ability of DATL. In addition, we will verify more general aircraft collision avoidance system with DATL.

References

[1] E.A. Lee. Cyber physical systems: Design challenges. In *Object Oriented Real-Time Distributed Computing (ISORC), 2008 11th IEEE International Symposium on*, pages 363 – 369, 2008.

[2] T.A. Henzinger. The theory of hybrid automata. In *Logic in Computer Science, 1996. LICS '96.Proceedings., Eleventh Annual IEEE Symposium on*, pages 278–292, 1996.

[3] A. Platzer. *Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics*. Springer, Heidelberg, 2010.

[4] A. Platzer and E. Clarke. Computing differential invariants of hybrid systems as fixedpoints. *Form. Methods Syst. Des.*, 35(1):98–120, 2009.

[5] A. Platzer and J.D. Quesel. Keymaera: A hybrid theorem prover for hybrid systems (system description). In *IJCAR*, pages 171–178, 2008.

HybridUML Based Verification of CPS Using Differential Dynamic Logic

Min Zhu, Bixin Li, Jiakai Li, Qiaoqiao Chen,Xiaoxiang Zhai, Shunhui Ji
School of Computer Science and Engineering, Southeast University, Nanjing, China
{kong, bx.li}@seu.edu.cn

***Abstract*—CPS (Cyber-Physical Systems) which are characterized by the combination of computation, communication and control are applied in many safety-critical domains. For the successful application of CPS, it is very important to ensure the correctness of CPS. Many researchers are concerned about using formal verification to verify the correctness of CPS since it has played a key role in improving the security and reliability of systems. In this paper, we demonstrated the feasibility of our CPS modeling and verification framework through a case study. We first introduced HybridUML, an extension of UML, to model CPS, then we presented a model transformation method mapping HybridUML model to Hybrid Program, and finally verified the properties of the resulting model with KeYmaera.**

Keywords-CPS; differential dynamic logic; HybridUML; model transformation; verification

I. INTRODUCTION

Cyber-Physical Systems (CPS) integrate computation with physical processes. Embedded computers and networks monitor and control the physical processes, usually with feedback loops where physical processes affect computations and vice versa[1]. CPS have brought many opportunities and challenges to various industries, such as intelligent transportation, industrial automation, smart medical, agriculture and national defense. It is crucial that the designed CPS work as expected. A growing number of researchers are concerned about the property verification of CPS since verification technique has played a key role in improving the security and reliability of CPS. However, traditional model checking techniques which are designed for finite state systems do not work well in verifying CPS as there are an infinite number of states in CPS due to the discrete and continuous behaviors. Although model checking has been extended for infinite state systems in many studies [2-4], it still could not do well in verifying large-scale CPS since the restriction of reachability problem. Zhou [5] extended duration calculus [6] by introducing mathematical expression to the derivative of state variable. However, the reasoning method is not appropriate for automatic verification, especially for derivative and continuity. A theorem proving method based on differential dynamic logic (DDL) proposed by A.Platzer has been well applied in the verification of CPS [7] and the operating model of a DDL formula is named Hybrid Programs(HP). Transforming generic model into a formal model for verification is a hot research field of software engineering[8]. In addition, HybridUML[9], an extension of UML, is introduced since UML has no precise semantics[10] and cannot model continuous states. This paper introduces HybridUML as a generic model to model CPS, presents a method based on model transformation mapping HybridUML to Hybrid Programs, and verifies the resulting model finally.

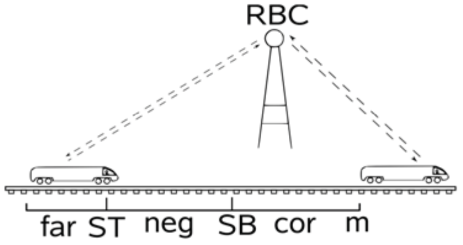


Figure 1. Dynamic movemnett authorities of ETCS

This paper demonstrates our DDL based framework for CPS modeling and verification by a case study. A simplified European Train Control System (ETCS) is shown in Figure 1[11]. Trains are guided by moving block principle and they are only allowed to move in the specified MA (Movement Authority). The RBCs (Radio Block Controller) update the MA dynamically based on the current track situation by wireless communication. The speed of a train can be regulated freely in the *far* mode, and the train can switch to the *neg* (negotiation) mode for MA extensions from *ST* (start taking). If there is no new MA updated after *SB* point, it starts braking in *cor* (correcting) mode.

Our verification architecture works as follows. First, the transformation from HybridUML to HP Model is accomplished through executing model transformation rules and rule template; second, KeYmaera input code generation from the resulting Model generated in the first step is fulfilled. KeYmaera is an automated and interactive theorem prover for a natural specification and verification logic for hybrid systems that combines deductive, real algebraic, and computer algebraic prover technologies[12]. The transformation rules in the paper are defined according to the relation between source meta-model and target meta-model on the basis of model

This work is supported partially by National Natural Science Foundation of China under Grant No. 60973149, partially by the Open Funds of State Key Laboratory of Computer Science of Chinese Academy of Sciences under Grant No.SYSKF1110, partially by Doctoral Fund of Ministry of Education of China under Grant No. 20100092110022, and partially by the College Industrialization Project of Jiangsu Province under Grant No.JHB2011-3.

transformation idea in MDA[13]. The DDL based framework for ETCS modeling and verification is shown in Figure 2.

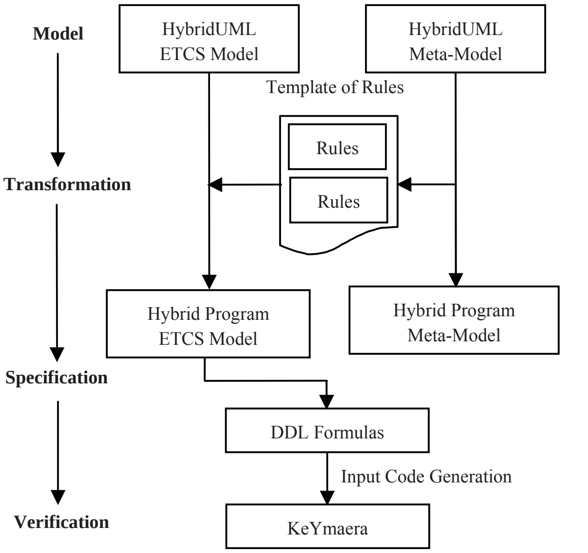


Figure 2. The framework of ETCS modeling and verification based on DDL

II. MODELING OF ETCS

In HybridUML, the system's static structure is described by class diagrams and composite structure diagrams whose basic modeling element are both Agent. Modes are hierarchical Hybrid Automata which contain discrete and continuous transitions. Transitions are entered or exited through control points which can be classified into entry and exit control points.

HybridUML is not restricted to be used with the HybridUML Expression Language[10] (DifferentialExpression, AlgebraicExpression,InvariantExpression). In order to facilitate the transformation, we model CPS using HybridUML with the corresponding expressions in first-order logic.

This paper transforms the HybridUML to Hybrid Programs based on the specification defined by Bisanz in [10]. We add some definitions based on HybridUMLModel as follows:

1) Classification of Modes and Agents.

According to the existence of submodes in Modes, Modes are divided into primitive Modes and composite Modes. Similarly, Agents are divided into primitive Agents and composite Agents by subagents. There are premises and conclusions before and after the symbol •.

$kind_M : M \to \{$ CompositeMode, PrimitiveMode $\}$

$\forall m \in M,\ submode_M(m) \neq \emptyset \bullet kind_M(m) :=$ CompositeMode

$\forall m \in M,\ submode_M(m) = \emptyset \bullet kind_M(m) :=$ PrimitiveMode

2) Top-level Mode. Only the primitive Agents contain top-level modes for behavior description.

$TM_A: A \to behavior_A(A),\ \forall a \in A \bullet kind_A(a) =$ PrimitiveAgent

3) srcT and tarT. We already know that srcT and tarT represent the source and target control point of transition and add the definition of source Mode and target Mode of transition.

$srcMode_T: T \to \{M \cup MI\} \quad tarMode_T: T \to \{M \cup MI\}$

Transitions are not only switch control between Modes and their submodes instances but also between submodes instances. So srcMode$_T$ and tarMode$_T$ must satisfy the following expressions:

$$\forall t \in T \bullet srcMode_T(t) \in M \wedge tarMode_T(t) \in MI$$
$$\vee\ srcMode_T(t) \in MI \wedge tarMode_T(t) \in MI$$
$$\vee\ srcMode_T(t) \in MI \wedge tarMode_T(t) \in M$$

4) classification of transitions. According to the classification of control points, Transitions are divided into three parts: EntryTransitions, InternalTransitions and ExitTransitions.Take EntryTransitions for example:

EntryTransitions represent the Transitions whose sources are Mode's entry control points and targets are Mode instance's entry control points.

$$EntryTransitions_T : M \to T$$
$$\forall t \in T \bullet src_T(\,t\,) \in CP \wedge kind_{CP}(\,src_T(\,t\,)\,) = entry$$
$$\wedge\ tar_T(\,t\,) \in CPI \wedge kind_{CP}(cp_{CPI}(\,tar_T(\,t\,)\,)\,) = entry$$

Firstly, we model ETCS using HybridUML, and get composite structure diagrams and statechart diagrams. Statechart diagram of train is shown in figure 3. Variable *A* and *b* represent maximum acceleration and maximum deceleration of train; Variable *m* represents the value of current *MA*; Variable *z* and *v* represent train's position and velocity of current *MA*; Variable *t* is the safety moving time determined by automatic train protection unit dynamically; Variable *message* means whether the train is in emergency situation; Variable *recommedspeed* means the recommended speed of current *MA*, that is, train must slow down when *v* beyond this value. Variable *message* and *recommedspeed* are shared between agent *Train* and *RBC* in composite structure diagram. When the position of train is beyond *SB*, we calculate the minimum value of *SB* using formula Λ through KeYmaera.

$$(\ \forall m\ \forall z\ (\ m - z\ ) \geq SB \wedge v^2 \leq 2b\ (\ m - z\ ) \wedge v \geq 0 \wedge A \geq 0\ )$$
$$\to [\ a := A;\ z' = v,\ v' = a,\ t' = 1\ \&\ v \geq 0 \wedge t \leq \varepsilon\ ]\ (\ z \leq m)$$
$$\equiv \mathrm{SB} \geq \frac{v^2}{2b} + (\frac{A}{b} + 1)(\frac{A}{2}\varepsilon^2 + \varepsilon * \mathrm{v}\ ) \qquad (\Lambda)$$

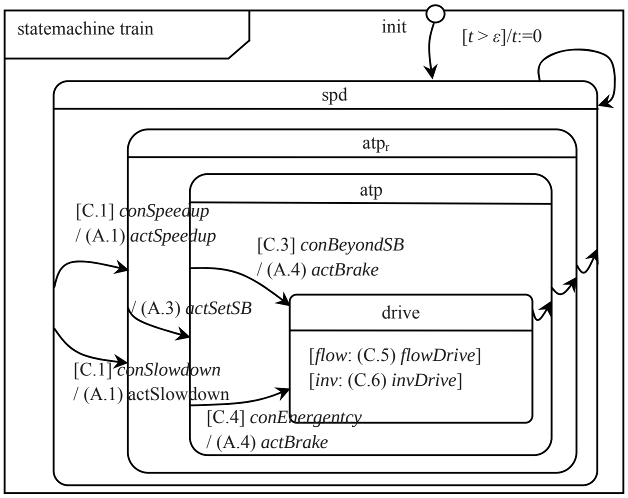


Figure 3. Statechart of agent train

The details of variables in figure 3 are as follows:

$(C.1)\ conSpeedup \equiv v \leq recommendSpeed$
$(A.1)\ actSpeedup \equiv a := *;\ ?\ (-b \leq a \leq A)$
$(C.2)\ conSlowdown \equiv v \geq recommendSpeed$
$(A.2)\ actSlowdown \equiv a := *;\ ?(-b \leq a \leq 0)$
$(A.3)\ actSetSB \equiv \mathrm{SB} := \dfrac{v^2}{2b} + (\dfrac{A}{b} + 1)(\dfrac{A}{2}\, \varepsilon^2 + \varepsilon * \mathrm{v})$
$(C.3)\ conBeyondSB \equiv m - z \leq SB$
$(C.4)\ conEnergentcy \equiv rbc.message := emergency$
$(A.4)\ actBrake \equiv a := -b$
$(C.5)\ flowDrive \equiv z' = v,\ v' = a,\ t' = 1$
$(C.6)\ invDrive \equiv \mathrm{v} \geq 0 \wedge \mathrm{t} \leq \varepsilon$

III. Mode transformation

In this section, we present the mode transformation rules and the rule template, and apply them to get Hybrid Programs. At last, the KeYmaera input code is generated. RuleType means the type of rule include MappingRule and ProcessingRule; the variables and data structure of rule are given in declaration part; the main body of rule is mapping/processing part; Return Result part return the rule result. Transforming HybridUML to Hybrid Programs mainly involves rules of static structure and rules of dynamic behavior.

A. Transformation Rules of static structure

Agents are the basic element of static structure in HybridUML model. This paper focuses on the situation in which only one Agent contains continuous behavior among Agents.

1) Shared variables table building

Variables have their own scope in Agents and share their value via variable connectors among different Agents. There are only global variables in Hybrid Programs which means that all variables' scope is the whole program. To avoid the shared variables are treated as different ones, we build a shared variables table named SharedVariableTable for the sake of signing each shared variable. Each line of the table contains the variable ports and corresponding variable names. The rule is named CreateSharedVariableTable.

2) Static structure to HPSkeleton transforming

Composite Agents don't have their own behavior as primitive Agents do. We focus on only one Agent contains continuous behavior among Agents. In Hybrid Programs, concurrency of Agents is handled as follows. Suppose that A_P is a set of primitive Agents, $A_P \in A \cup AI$, $\forall a \in A_P \bullet \mathrm{kind}_A\ (a) = \mathrm{PrimitiveAgent}$. If $a_0 \in A_P$ is a primitive Agent which has continuous behavior, the decisions of AP only depend on the point in time when other Agents only have discrete behavior, not on the communication latency. Thus, the nondeterministic interleaving in CPS where either the A_P or ($\cup$) other Agents chose to take action faithfully models every possible arrival time without the need for and explicit (delayed) channel model[14]. The * indicates that the interleaving of A_P and other Agents repeats arbitrary times. The rule of transform static structure to HPSkeleton named MappingStructureToHP.

B. Transformation rules of dynamic behavior

Composite modes must be flattened to primitive modes according to the semantic consistency requirement of model transformation. The rule that submodes inherit all constraints of their parent's mode recursively till the top-level mode is named MergeConstraints.

1) Hierarchical state chart flatting

In HybridUML, the modes containing source and target control points of transition may be primitive mode or composite mode, and the control points may be junction, according the source and target state of transition are all primitive state in HP. So the hierarchical state chart which is composed by modes must be flattened to primitive state charts. After the application of rule MergeConstraints, every mode of state charts in HybridUML has inherited its parent's mode's constraints. For each transition of state charts, if the source or target mode of transition is composite mode then search up or down to find the final primitive mode. In the process of searching, if the source control point of transition t_2 is the target control point of transition t_1, then add the transition whose source control point is the source control point of t_1 and target control point is t_2 to the transition set T. After this operation, delete t_2, t_2 from T. This rule is named CreateTransitionPath. For each transition in transition set T in HybridUML, if the target control point of the transition is junction, then apply rule CreateTransitionPath to every transition whose target control point is the junction till all the junctions have been eliminated. This rule is named EliminateJunction. The rule for flattening hierarchical state chart is named FlatHierarchyMode.

2) Transition transforming

There are only primitive modes and transitions between them in HybridUML after flattening. The entry transitions of top-level mode are mapped to InitBlock of *HP*, primitive modes and the transitions are mapped to *CE* and *DJ* of *HP* respectively. The rule is named MappingETtoInitBlock.

We defined TransitionGraph to signify the primitive modes and the transitions between them corresponding to HPContent of HP. TransitionGraph is defined as follows.

Definition 1. TransitionGraph (M_{TG}, T_{TG}) is directed graph which is made up of primitive modes and transitions between them. The set of vertex M_{TG} is the set of primitive modes of HybridUML, $\forall m \in M_{TG} \bullet kind_M\ (m) = PrimitiveMode$, and T_{TG} is the set of edges satisfying $\forall t \in T_{TG} \bullet kind_M\ (srcMode_T\ (t)) = PrimitiveMode \wedge kind_M\ (tarMode_T\ (t)) = PrimitiveMode$, the relationship between edges $t = \{<v,w>|\ t \in T_{TG} \bullet v = srcMode_M\ (t),\ w = tarMode_M\ (t)\ \}$ represents the transition whose source mode is v and target mode is w.

TransitionGraph contains two kinds of dynamic behavior of HybridUML: discrete transition and continuous evolution, which accord with discrete jump and continuous evolution in HP respectively. The discrete transitions which include trigger event, guard and action are mapped to corresponding elements in HP. In order to mark the current active state in HP, we add a variable *ActiveState* in InitBlock. During the transition transforming, we first compare the mode whose current transition belongs to *ActiveState*, if it is equal then guard of transition will be judged. And then *ActiveState* will be assigned the value of target mode of transition after transition action is finished. The rule of transition transforming is named MappingTGtoHP. Rule detail is illustrated in Figure 5.

```
MappingRule MappingTGtoHP {
  [Rule Input]
    TransitionGraph tg
  [Declaration]
    HPContent hpcontent
    $Trans_{TG}$: TG → $T_{TG}$
    $Mode_{TG}$: TG → $M_{TG}$
  [Mapping]
  // Transforming the discrete transition
  for each t in $Trans_{TG}$( tg )
    hpcontent ← ( ? hp.ActiveState = $srcMode_T(t)$ ;?$grd_T(t)$ ;
    $act_T(t)$ ; hp.ActiveState := tarMode(t)) ∪ hpcontent
  // Transforming the continous evolution
  for each m in $Mode_{TG}$( tg )
    hpcontent ← ( ?hp.ActiveState = m; $flow_M(m)$ &
    $inv_M(m)$) ∪ hpcontent
  [Return Result]
    return hpcontent
}
```

Figure 4. Rule MappingTGtoHP

C. Template of rules applied

In section IV we have built model transformation rules. To organize those rules, a template of rules is needed. Before building the template, we give two methods: method RenameSharedVariables sets the shared variables to the same name and method MergeHPModel merges the primitive mode in *HP*.After model transformation, we get the following *HP*:

$$ETCS \equiv (\ train \cup rbc)^*$$

$$train \equiv ctrl;\ drive$$

$$ctrl \equiv \sigma_1 \cup \sigma_2 \cup \sigma_3 \cup \sigma_4$$

$$\begin{aligned}\sigma_1 \equiv\ & (?ActiveState = drive;\ ?(v \le recommendSpeed);\\ & a := *;\ ?(-b \le a \le A);\ SB := \frac{v^2}{2b} + (\frac{A}{b}+1)\ (\frac{A}{2}\varepsilon^2 + \varepsilon * v);\\ & (?(\mathrm{m{-}z} \le SB \vee message = emergency);\ a := -b;\\ & ActiveState := drive))\end{aligned}$$

$$\begin{aligned}\sigma_2 \equiv\ & (?ActiveState = drive;?(v \le recommendSpeed);\\ & a := *;\ ?(-b \le a \le A);\ SB := \frac{v^2}{2b} + (\frac{A}{b}+1)\ (\frac{A}{2}\varepsilon^2 + \varepsilon * v);\\ & (?(\mathrm{m} - \mathrm{z} \le SB \vee message\ != emergency);a := A;\\ & ActiveState := drive))\end{aligned}$$

$$\begin{aligned}\sigma_3 \equiv\ & (?ActiveState = drive;?(v \ge recommendSpeed);\\ & a := *;?(-b \le a \le 0);\ SB := \frac{v^2}{2b} + (\frac{A}{b}+1)\ (\frac{A}{2}\varepsilon^2 + \varepsilon * v);\\ & (?(\mathrm{m} - \mathrm{z} \le SB \vee message = emergency\ );\ a := -b;\\ & ActiveState := drive\ )\ )\end{aligned}$$

$$\begin{aligned}\sigma_4 \equiv\ & (\ ?\ ActiveState = drive;\ ?\ (\ v \le recommendSpeed\ \ );\\ & a := *;\ ?\ (\ -b \le a \le 0\ );\ SB := \frac{v^2}{2b} + (\frac{A}{b}+1)\ (\frac{A}{2}\varepsilon^2 + \varepsilon * v);\\ & (?(\mathrm{m} - \mathrm{z} \le SB \vee message\ != emergency);\ a := A;\\ & ActiveState := drive))\end{aligned}$$

$$drive \equiv ?\ ActiveState = drive;\ t := 0;\ (\ z' = v,\ v' = a,\ t' = 1\ \&\ v \ge 0 \wedge \mathrm{t} \le \varepsilon)$$

$$rbc \equiv message := emergency \cup (\ m := *,\ recommendSpeed\ \ := *;\ ?\ (\ recommendSpeed\ \ > 0\ )\ )$$

IV. SPECIFICATION AND VERIFICATION OF ETCS

This case verifies the safety of the ETCS cooperation protocol, that is, whether train can always move within MA. The safety of ETCS is specified by DDL formula as follows:

$$\Psi \to [ETCS^*]z \le m \qquad (\zeta)$$

$$\psi \equiv ActiveState = drive \wedge v^2 \le 2b(m-z) \wedge b > 0 \wedge A \ge 0$$

$$ETCS \equiv (\ ctrl\ ;\ drive) \cup rbc,\ \ ctrl \equiv \sigma_1 \cup \sigma_2 \cup \sigma_3 \cup \sigma_4$$

In the formula ζ , Ψ is the initial condition. There are lots of branches while reasoning, and the process ends when all th e branches are reduced to an obvious expression. It costs 236 steps to reduce in KeYmaera and generates 10 branches in all. To sum up, we know that train will stay within its *MA* all the time when the initial condition Ψ is satisfied. There will not be any crash when all trains moves within their own *MA* and the train cooperation protocol is safe.

V. CONCLUSION

In this paper, we propose a DDL based framework for CPS modeling and verification. We model an ETCS using a unified modeling language HybridUML, transform HybridUML model to the operating model of DDL-Hybrid Program, and reason the resulting ETCS property formula by DDL proof rules. We propose a transformation method to translate a HybridUML model to its corresponding hybrid program. Through the framework, we not only get a unified modeling method that can be comprehended by most system designers but also can verify property through DDL calculus using KeYmaera.

[1] Lee EA. Cyber physical systems: Design challenges. UCB/EECS. 2008

[2] Lygeros J, Collins P. Computability of finite-time reachable sets for hybrid systems. Proceedings of the 44th IEEE Conference on Decision and Control, and the European Control Conference. 2005: 4688-4693.

[3] Chutinan A, Bruce H. Computational techniques for hybrid system verification. IEEE Transactions on Automatic Control(HSCC'03). 2003: 64-75.

[4] Tiwari A. Approximate reachability for linear systems. In Proceedings of Hybrid Systems: Computation and Control (HSCC'03). 2003: 514-525.

[5] Ravn AP, Zhou CC, Hansen MR. An extended duration calculus for hybrid real-time systems. Hybrid Systems. 1993. 7: 36–59:

[6] Hansen MR, Zhou CC. Duration Calculus: A Formal Approach to Real-Time Systems. Monographs in Theoretical Computer Science. 2004.

[7] Clarke EM, Platzer A. The image computation problem in hybrid systems model checking. In Proceedings of Hybrid Systems: Computation and Control (HSCC'07). 2007: 473–486.

[8] Caplat G, Sourrouille JL. Model Mapping Using Formalism Extensions. Software,IEEE. 2005. 22(2): 44-51.

[9] Bisanz S, Berkenkötter K, Hannemann U, Peleska J. The HybridUML profile for UML 2.0. International Journal on Software Tools for Technology Transfer (STTT). 2006. 8(2):

[10] Bisanz S. Executable HybridUML Semantics: A Transformation Definition. PhD thesis. University of Bremen. 2005.

[11] Platzer A, Quesel JD. European Train Control System: A case study in formal verification. ICFEM, LNCS. Springer: 2009: 246–265.

[12] Quesel JD, Platzer A. KeYmaera: A hybrid theorem prover for hybrid systems. Proc. of IJCAR 2008, LNCS 5195. Springer: Heidelberg. 2008: 171-178.

[13] Huang ZQ, Liu YP, Zhu Y. Research on Model Transformation Method of Real-time System Based on Metamodeling. Journal of Chinese Computer Systems. 2010. 31(11): 2146-2153.

[14] Platzer A. Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics. Springer. 2010.

A HybridUML and QdL Based Verification Method for CPS Self-Adaptability

Jiakai Li, Bixin Li, Qiaoqiao Chen, Min Zhu, Shunhui Ji, Xiaoxiang Zhai
School of Computer Science and Engineering, Southeast University, Nanjing, China
{jiakai_li, bx.li}@seu.edu.cn

Abstract—**CPS (Cyber-Physical Systems) are physical and engineered systems featuring a tight combination of computation and physical processes by communication networks. CPS are mainly applied in some critical domains, so it is very essential to ensure the correctness of CPS. Formal verification has been successfully applied in the correctness verification of CPS; however, the high theoretical level of formal modeling techniques of formal verification and the lack of visualization of formal models make it difficult to integrate formal verification with enterprise standard system development process. In this paper, we model CPS by HybridUML, an extension of UML, and then transform HybridUML model into the input language of theorem prover KeYmaera-QHP(Quantified Hybrid Program), and finally verify the QHP code with KeYmaera.**

Keywords-CPS; Formal Verification; HybridUML; QHP; MetaModel; Model Transformation

I. INTRODUCTION

CPS, in a broad sense, are controllable, trustable and extensible networked physical device systems that deeply integrate computation, communication and control capabilities based on environment perception [1]. CPS are mainly applied in areas having a high demand on performance, so to ensure the correctness of CPS is extremely important.

CPS evolve over time with interacting discrete and continuous dynamics, which accords with the definition of hybrid systems, so the verification theory of hybrid systems can be referenced in CPS verification. Although many present model checking[2,3,4,5] and theorem proving methods [6,7,8] have been proposed for Hybrid Systems verification, these methods have great deficiencies in supporting the verification of s ystems with distributed characteristics. To solve this, André Platzer proposed a verification technique based on QdL (Quantified Differential Dynamic Logic)[9,10,11]. In order to achieve automatic verification, André designed a theorem prover- KeYmaera. However, the high theoretical level of formal modeling method in QdL and the lack of visualization of QHP make it difficult to integrate QdL based formal verification with enterprise system development processes in which the Unified Modeling Language (UML) has been accepted as the de facto standard modeling language. However, UML lacks precise formal semantics, making it hard to directly verify UML models formally. The paper first models CPS by HybridUML, and then translates the model into the input model of KeYmaera[12], and finally performs formal verification with KeYmaera to ensure the correctness of CPS.

The interconnection topology and the number of active members of CPS network, especially those composed of mobile devices such as the Distributed Air Traffic Collision Avoidance System (DATC), tend to change dynamically, making how to ensure critical properties hold dynamically really challenging. The characteristic that CPS adjust their behaviors flexibly according to the network structural and dimensional dynamics is called self-adaptability. Verifying self-adaptability, Hierarchical Hybrid StateMachine-Mode is chosen as the CPS modeling view. First, the hierarchical feature of the top-level Mode model is eliminated according to execution semantics and a FlatMode model is gotten. Second, transform the FlatMode model to a QHP model by executing corresponding ATL transformation rules. Third, template rules in a customized template language will be called to transform the QHP model to QHP code, and then specify the CPS self-adaptability property with a QdL formula . Finally, verify property formula automatically with KeYmaera.

The paper is organized as follows. Section Ⅱ introduces HybridUML based modeling method. SectionⅢ describes the translation from HybridUML model to QHP code. Section Ⅳ illustrates the application of our verification architecture by a case study of a DATC. Section Ⅴ summarizes the paper and makes a discussion about the future work.

II. CPS MODELING BASED ON HYBRIDUML

A. Basic modeling elements of HybridUML

1) Agent

In HybridUML, Agent is a stereotype for Class in UML. As the basic building block for describing the static system architecture, an Agent can own an internal structure consisting of other Agent instances to support hierarchical models [13].

2) Hierarchical Hybrid StateMachine –Mode

In HybridUML, Mode is an extension of UML StateMachine and is used for describing Agents′ behavior [13]. As a Hierarchical Hybrid StateMachine, a Mo de can contain sub-modes and transitions inside its region. When a Mode is

This work is supported partially by National Natural Science Foundation of China under Grant No. 60973149, partially by the Open Funds of State Key Laboratory of Computer Science of Chinese Academy of Sciences under Grant No.SYSKF1110, partially by Doctoral Fund of Ministry of Education of China under Grant No. 20100092110022, and partially by the College Industrialization Project of Jiangsu Province under Grant No.JHB2011-3.

executing a continuous transition, it i s actually acting as a whole, which means the continuous updates of variables in the mode are described by the constraints contained in itself and all its active sub-modes.

B. The Extension of HybridUML

1) The Stereotype for Classes::Kernel::Expression

In order to deal with varying number of CPS network members and properly express the co-evolution of all members, first-order function symbols are used to parameterize primitive state variables, for example, $x(i)$ denotes the location of car i, and quantifiers and a type system are introduced to quantify over all objects of a given type, for example, $\forall i{:}C$, which represents all car members of type C. Correspondingly, an extension of Expression named QuantifiedExpression is introduced. Its Subclasses include QDifferentialExpression, such as $\forall i{:}C!x(i)''=a(i)$, QAlgebraicExpression, such as $\forall i{:}C!x(i){:=}a$, and QBooleanExpression, such as $\forall i,j{:}C\ far(i,j)$, in which $far(i,j)$ expresses a constraint.

2) The Stereotype for Classes::Kernel::Constraint

Corresponding to the extension of Expression, a Stereotype named QuantifiedConstraint for Constraint is given, endowing Constraints with quantified features. QuantifiedConstraint includes quantified differential constraints, which contain QDifferentialExpression instances expressing flow constraints, and quantified invariant constraints containing instances of type QBooleanExpression describing invariant constraints.

3) The Stereotype for CommonBehaviors::Comunications::Signal

To express the signal that new members emerge in CPS network, NewObjectSignal is introduced as an extension for Signal in UML. To express the type of the new member, a new attribute named agentType is added in each NewObjectSignal.

4) The Stereotype for CommonBehaviors::Comunications::SignalEvent and ChangeEvent

Corresponding to the extension of Signal, NewSignalEvent, a stereotype for SignalEvent, is introduced. A NewSignalEvent happens when a new member appears in the CPS network and it contains a NewObjectSignal instance inside. Considering the extension of Boolean expression, extending the ChangeEvent of UML becomes essential. As the profile of ChangeEvent, a QChangeEvent contains a QBooleanExpression representing the condition triggering events when it becomes true.

5) The Stereotype for CommonBehaviors::Comunications::Trigger

As only very limited event types are used in our paper, it becomes essential to restrict the event types that a Trigger contains. A profile of Trigger named ModeTrigger is proposed and the event types a ModeTrigger contains are confined to NewSignalEvent and QChangeEvent.

6) The Stereotype for ModeUpdateActivity

Due to the extension of algebraic expression, QUpdateActivity, a stereotype for ModeUpdateActivity, is given to d escribe discrete updates on variables of all CPS network members, and it c ontains a QAlgebraicExpression expressing discrete update behaviors.

III. TOP-LEVEL MODE MODEL TO QHP CODE TRANSLATION

A. Metamodel of Hierarchical Hybrid StateMachine-Mode

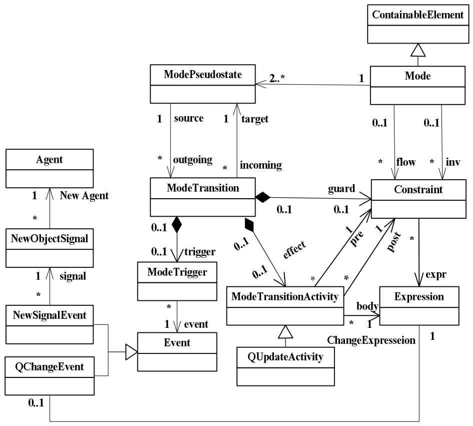


Figure 1. Metamodel of Hierarchical Hybrid StateMachine-Mode

Definition 1. (**Mode**) A Mode is a tuple: < *VS*:{DataType}, *SM*:{Mode}, *ES*:{ModePseudostate}, *XS*:{ModePseudostate}, *T*:{ModeTransition}, *Cons*:{Mode!Constraint}>[14], where *VS* is the variable set, *SM* represents all sub-modes, *ES* represents all entry pseudostates, *XS* denotes all exit pseudostates, *T* represents all ModeTransition instances contained in the Mode's region and *Cons* denotes all constraints contained in the Mode. A Mode without a parent Mode is called a top-level Mode and the one without sub-modes is called a leaf Mode.ModeTransition can be divided into t hree types: EntryTransition ($s\in ES$), InternalTransition ($s\in XS^{SM}$, $t\in ES^{SM}$), ExitTransition ($t\in XS$). As for other element, see Figure 1.

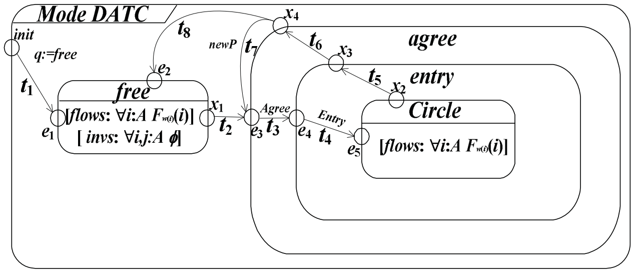


Figure 2. The Hierarchical Hybrid StateMachine-Mode model for DATS roundabout collision avoidance protocol

Being a Hierarchical Hybrid StateMachine, an active top-level Mode actually contains more than one active sub-modes inside. All such active Mode instances actually form a path starting from the top-level Mode down to the innermost active leaf Mode. Such a path is called a StateConfiguration. As shown in Figure 2, th e Mode path *DATC*:*agree*:*entry*:*Circle* forms *a* StateConfiguration when *Circle* is active.

A Mode acquires control through an EntryStep seen as a path that starts from an entry pseudostate of the Mode to the entry pseudostate of the to-be-active innermost leaf Mode. As shown in Figure 2, (*init*, t_1, *free*) is the EntryStep of Mode *DATC*. A Mode relinquishes control via an ExitStep, which is a path starting from the exit pseudostate of the innermost active leaf Mode to an exit pseudostate of the Mode, for example, (*Circle*, t_5, t_6, x_4,). When an InternalTransition is executed in a Mode, an ExitStep whose targe is the starting pseudostate of the InternalTransition and an EntryStep originating from the target pseudostate of the InternalTransition will take place meanwhile, forming an execution path called InternalStep, for example, (*free*, t_2, t_3, t_4, *Circle*). When a top-level Mode resides in a certain StateConfiguration, continuous variables will be updated according to the flow and invariant constraints contained in the StateConfiguration, and such an execution process is called a ContinuousStep, As shown in Figure 2, (*DATC*:*free*, *DATC.flows*∩*free.flows*, *DATC.invs*∩*free.invs*) is a ContinuousStep.

B. Metamodel of QHP model

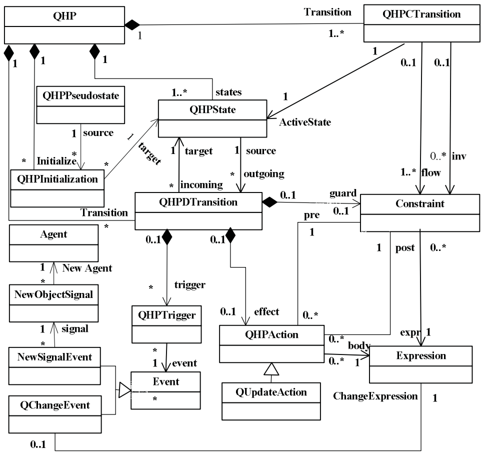


Figure 3. Metamodel of QHP model

Definition 2. (**QHP**) A QHP model can be formalized as a tuple: <*IBS*:{QHPInitialization},*DTS*:{QHPDTransition}, *CTS*:{QHPCTransition}, *Cycle*:Boolean, *SG*:{QHPState}>, where *IBS*, *DTS* and *CTS* denote the set of all QHPInitialization, QHPDTransition and QHPCTransition instances of the QHP model respectively, *Cycle* signifies whether the model can be executed repeatedly, and *SG* denotes all QHPState instances. As for other model element, see Figure 3.

C. Model transformation from toplevel Mode to QHP Code

1) Description of main transformation rule based on ATL

TABLE I. MAIN TRANSFORMATION RULE-TLMODE2QHPCODE

```
entrypoint rule TLMode2QHPCode{
from
  m:Mode(m.isTopLevelMode=true)
to
  p:QHP (
    IBS<- m.EntrySteps->collect(i|thisModule.ES2QHPI(i)),
    DTS<- m.InternalSteps->collect(d|thisModule.IS2QHPDT(d)),
   CTS<- m.ContinuousSteps->collect(i|thisModule.CS2QHPCT(i)),
   Cycle<- m.CycleSymbol)
do{
   QHPModel2QHPCode(p);}
}
```

As shown in TABLE Ⅰ, main rule TLMode2QHPCode expressed in ATL[15] accomplishes the translation from a top-level Mode model to its corresponding QHP code. For a given top-level Mode m, the execution process of the rule TLMode2QHPCode is as follows. First, get all EntryStep instances of *m* by calling ATL helper EntrySteps, and by rule ES2QHPI, each EntryStep is mapped to a QHPInitialization instance, making up *IBS*. Second, get all InternalStep instances of *m* by calling helper InternalSteps, and by rule IS2QHPDT, each InternalStep is mapped to a QHPDTransition instance composing *DTS*. Third, get all ContinusStep instances of *m* by calling helper ContinusSteps, and by rule CS2QHPCT, each ContinusStep is mapped to a QHPCTransition instance composing *CTS*. Fourth, get the mark indicating whether all InternalStep instances composes an execution cycle by calling helper CycleSymbol, and as a Boolean variable, CycleSymbol can be assigned directly to the variable *Cycle* of the QHP model *p*. Finally, template rule QHPModel2QHPCode will be invoked to generate final QHP code form QHP model *p*.

2) QHP code generation from QHP model

TABLE II. MAIN TEMPLATE RULE

```
template rule QHPModel2QHPCode{
from
  m:QHP
to
 IBS2QHPCode(m.IBS)+';'
  +'('+DTS2QHPCode(m.DTS)+ '∩'
  +'('+CTS2QHPCode(m.CTS)+ ')'
  +Cycle2Star(m.Cycle)
}
```

The template rule QHPModel2QHPCode is shown in TABLE Ⅱ.In order to transform QHP model to QHP code, a simple template-based model-to-code generation method is given in the paper. For a given QHP model *m*, the execution process of the rule QHPModel2QHPCode is as follows. First, get *IBS* of *m* by dot operation and call template rule IBS2QHPCode to output *IBS* in a specified format. Second, get *DTS* of *m* by dot operation and template rule DTS2QHPCode is called to fulfill the formatted output of *DTS*. Third, get *CTS* of *m* and call template rule CTS2QHPCode to output *CTS* in a specified format. Finally, rule Cycle2Star is invoked to determine whether to print the star symbol indicating whether QHP code can be executed cyclically. If *Cycle* is true, then output star symbol. All output formats above follow the format of QHP Code defined in the previous section.

IV. CASE STUDY

A. Case description and modeling

$$CAMP \equiv (free; Agree; Entry; Circle; (newP; Agree; Entry; Circle) \cup (?\, true))\, *$$

$$free = \forall i : A\ \mathscr{F}_{w(i)}(i)\ \&\ \phi$$

$$Agree \equiv c_1 := *; c_2 := *; w := *;$$

$$Entry \equiv \forall i : A\ (d_1(i) := -w(x_2(i) - c_2)); \forall i : A\,(d_2(i) := w(x_1(i) - c_1));$$

$$newP \equiv (n := new\, A; ?\, \forall i : A\, S(i, n))$$

$$Circle \equiv \forall i : A\ \mathscr{F}_{w}(i)$$

$$S(i, j) \equiv A((x_1(i) - x_1(j))^2 + (x_2(i) - x_2(j))^2 \ge P^2 \vee i = j)$$

$$\phi = \forall i, j : A\, S(i, j)$$

Figure 4. The QHP representation of DATS collision avoidance protocol

According to the verified property, the collision avoidance protocol of DATS we used is properly modified from [11]. The differential equation set representing the flight dynamics of an aircraft i of type A is shown in formula $F_{w(i)}(i)$ in which $x_1(i)$ and $x_2(i)$ denote the shift of i in the x-axis and y-axis respectively, $d_1(i)$ and $d_2(i)$ represent the velocity of aircraft i in the x-axis and y-axis respectively, and $w(i)$ represents the angular velocity of i.

$$x_1(i)'=d_1(i)\wedge x_2(i)'=d_2(i)\wedge d_1(i)'=-w(i)d_2(i)\wedge d_2(i)'=w(i)d(i)\quad (\boldsymbol{F_{w(i)}(i)})$$

Two aircraft i and j satisfy safety separation property if and only if the following formula holds:

$$S(i,j)\equiv(x_1(i)-x_1(j))^2+(x_2(i)-x_2(j))^2\geq P^2\vee i=j$$

At the beginning, all aircraft are far apart from one another, and they fly freely with angular velocity $w(i)$. However, when the distance between aircraft is no larger than ∂ that is positively proportional to P and the radius of the roundabout circle, all involved aircraft will come to an agreement on a common angular speed w and a roundabout circle center $c(c_1,c_2)$, then through phase *Entry*, each aircraft reaches a tangential location around center c. During *Circle* phase, each aircraft flies along the roundabout circle tangentially with the agreed common angular speed w. When a new aircraft dynamically appears in the collision avoidance zone, all aircraft again negotiate a new common angular speed w and roundabout circle center c, and then each aircraft goes through phase *Entry* and *Circle* again. Once the maneuver finishes, each aircraft will continue to fly in the original direction and enter phase *free* again. Adaptive collision avoidance protocol is illustrated in Figure 2 in the form of a Mode model. After the transformation of the model, final QHP code shown in Figure 4 is gotten.

B. Self-Adaptability specification and verification

In this case, we focus on the verification of adaptive collision avoidance property, i.e., when a new member enters the collision avoidance zone of existing aircrafts, the collision avoidance property still holds. Let $\phi\equiv\forall i,j{:}A\ S(i,j)$, and the property verified can be specified in the following formula: $\phi\rightarrow[CAMP]\phi$. The formula expresses that ϕ will always hold after evolving along *CAMP* for an arbitrary length of time if ϕ holds in the initial condition. In QdL, property verification is accomplished in the form of QdL proof calculus for which Sequent Calculus, which works by QdL proof rules, is adopted as the basic proof system [9]. During the proof, the proven property formula is placed at the bottom of the whole process, and the proof calculus is carried out from the bottom up. If the property finally holds, then the proof process ends up with a star symbol, otherwise the precondition that is required for the formula to hold is given. The abbreviated proof process that KeYmaera executes inside is shown in Figure 5. That all branches end successfully means the verified self-adaptability property holds.

$$\rightarrow r,ind\ \dfrac{[;][]gen\ \dfrac{[]gen\ \dfrac{\dfrac{true}{\phi\vdash[M1]\phi}\ DR'\,ax\ \dfrac{true}{\phi\vdash[free]\phi}}{\phi\vdash[\alpha]\phi}\qquad [\cup]\ \dfrac{[?]\ ax\ \dfrac{true}{\phi\vdash[?true]\phi}\ \dfrac{true}{\phi\vdash[(newP;M1)]\phi}}{\phi\vdash[\beta]\phi}}{\phi\vdash[\alpha;\beta]\phi}}{\vdash\phi\rightarrow[M^*]\phi}$$

Figure 5. Property proof process

V. Conclusion

The paper proposes a QdL based verification architecture for CPS self-adaptability property. First, CPS are modeled by HybridUML. Second, the model expressed by HybridUML is transformed into the input language of KeYmaera-QHP. Third, combined with the QHP code, the property to be proven is specified in the form of a QdL formula. Finally, the QdL formula is verified automatically using KeYmaera. In order to achieve the automatic translation from HybridUML Mode to QHP code and automatic verification by effective integration with KeYmaera, our future work will focus on the implementation of a model transformation tool.

Acknowledgment

The authors thank all the teachers and students who provide support for our work. Particularly, we thank Qi ShanShan and Liu CuiCui for their substantial work during paper correction.

References

[1] He JF. Cyber-physical Systems. Communication of the CCF, 2010,6(1):25-29.

[2] Clarke EM, Emerson EA, Sifakis J. Model checking: algorithmic verification and debugging. Communications of The ACM, 2009, 52(11):74–84 .

[3] Alur R, Courcoubetis C, Halbwachs N, et al. The algorithmic analysis of hybrid systems. Theoretical Computer Science, 1995,138(1): 3–34.

[4] Henzinger TA. The theory of hybrid automata. Proc. of the 11th Annual IEEE Symposium on Logic in Computer Science. 1996. 278-292.

[5] Clarke EM, Zuliani P. Statistical Model Checking for Cyber-Physical Systems. T. Bultan and P.-A. Hsiung (Eds.): ATVA 2011, LNCS 6996, 2011,1–12.

[6] Manna Z, S ipma H. Deductive verification of hybrid systems using STeP. Proc. of Hybrid Systems: Computation and Control, First International Workshop, HSCC 98, Berkeley, California, USA. California: Springer. 1998. 305–318

[7] Van Beek DA, Man KL, Reniers MA, et al. Syntax and consistent equation semantics of hybrid Chi. Journal of Logic and Algebraic Programming, 2006, 68(1-2):129–210.

[8] Zhou CC, Ravn AP, Hansen MR. An extended duration calculus for hybrid real-time systems. Hybrid Systems, 1993, 7: 36–59.

[9] Platzer A. Quantified Differential Dynamic Logic for Distributed Hybrid Systems. In: A.Dawar, H.Veith, eds. Proc. of CSL 2010, LNCS 6247. Heidelberg: Springer-Verlag, 2010. 469-483.

[10] Platzer A. Quantified differential dynamic logic for distributed hybrid systems. Technical Report, CMU-CS-10-126, SCS, Carnegie Mellon University, 2010.

[11] Platzer A. Quantified Differential Invariants. In: Emilio Frazzoli, Radu Grosu, eds. Proc. of the 14th ACM International Conference on Hybrid Systems: Computation and Control, HSCC 2011, Chicago, USA, April 12-14, 2011, Chicago: ACM, 2011. 63-72.

[12] Platzer A, Quesel JD. KeYmaera: A hybrid theorem prover for hybrid systems. In: Alessandro Armando, Peter Baumgartner, et.al, eds. Proc. of IJCAR 2008, LNCS 5195, Heidelberg: Springer, 2008.171-178.

[13] Berkenkötter K, Bisanz S, Hannemann U, et al. The HybridUML profile for UML 2.0. International Journal on Software Tools for Technology, 2006, 8(2):167–176.

[14] Alur R, Grosu R, Lee I, et al. Compositional modeling and refinement for hierarchical hybrid systems. In Proceedings of J. Log. Algebr. Program.. 2006, 105-128.

[15] Jouault F, Allilaire F. ATL: A model transformation tool. Science of Computer Programming, 2008, 72(1-2):31-39.

Disabling Subsumptions in a Logic-Based Component

Éric Grégoire Sébastien Ramon
CRIL CNRS UMR 8188 - Université d'Artois
Rue Jean Souvraz SP18
F-62307 Lens, France
{gregoire,ramon}@cril.fr

Abstract

In this paper, we address a problem that is often overlooked in the gradual construction of a logic-based knowledge component: how can a new piece of information g be added into a component KC so that g is not subsumed by KC? More precisely, the focus is on iterating this subsumption-freeness enforcement process when the component is increased step by step. Interestingly, it is shown that the approach and results initially related to standard Boolean logic can directly apply to some non-monotonic frameworks.

1 Introduction

The representation and handling of evolving knowledge has long been a major topic of Artificial Intelligence and knowledge engineering research, especially when knowledge is expressed in logic [1, 2, 3, 4, 5]. In this respect, updating and revising knowledge and beliefs remain very fertile domains of research [6]. This paper is concerned with a problem that has often been overlooked so far in such a context. When a new piece of knowledge g is not logically conflicting with a knowledge component KC, i.e. when g is consistent with KC, it is often assumed that g should simply be added into KC. This postulate is shared by most theoretical studies about the dynamics of knowledge, like knowledge and belief revision or update [6, 7, 8, 9, 10], and knowledge merging or fusion [11, 12]. All the aforementioned studies and the abundant subsequent literature concentrate on handing inconsistent situations [13]. Recently, we have shown that in real-life circumstances, inserting g can require further modifications of KC when g must not be subsumed by KC [14, 15] and when g is consistent with KC. For example, assume that a rule $\mathcal{R}_1$ "When the switch is on and the lamp bulb is ok then the light is on" is to be inserted within KC, which already contains $\mathcal{R}_2$ "When the switch is on then the light is on". Clearly $\mathcal{R}_1$ and $\mathcal{R}_2$ are not logically conflicting. Actually, $\mathcal{R}_1$ is *subsumed* by $\mathcal{R}_2$ in the sense that, from a logical point of view, $\mathcal{R}_1$ is a mere strict deductive consequence of $\mathcal{R}_2$. Accordingly, both rules cannot co-exist in a same knowledge component KC if we want $\mathcal{R}_1$ to prevail. Otherwise, nothing would prevent $\mathcal{R}_2$ to apply and conclude that the light is on, provided that the switch is on, even when the lamp bulb is not ok. Another example is as follows. Assume that an agent believes that Peter is in his office or at home. A new, more informative piece of knowledge comes up: "Peter is in his office or at home or in his car", leaving open the additional possibility that Peter is actually in his car. This last piece of information is however again a mere strict deductive consequence of the former knowledge, not conflicting with it. In some way, the former knowledge should be expelled from KC to really enforce the contents of the new -more informative-one. However, subsumption between formulas is not always that apparent; in the worst case, subsumption between two formulas can only be established by making use of all formulas of KC.

An original approach addressing this problem has been introduced in [14]: both its algorithmic facets and formal aspects have been investigated. This former study has been extended to a general non-monotonic setting in [15] and applied to the legal domain in [16]. In this paper, the focus is on the dynamics of such a process: how could the insertion of knowledge be iterated, taking into account the above treatment of information that must not be subsumed. Interestingly, this leads us to define a new form of *integrity constraints*: knowledge that cannot be logically strengthened. A second contribution of the paper is in showing that the involved standard logic apparatus offers a sufficient framework for expressing the most important aspects of this subsumption-related issue for an interesting case of non-monotonic logics.

The formal framework in this paper is standard Boolean logic. On the one hand, it is the simplest possible setting for presenting and discussing the above subsumption-related issues. On the other hand, recent dramatic progress in Boolean search and reasoning has now revived Boolean logic as a realistic framework for representing large knowledge sets and solving numerous complex reasoning artificial intelligence tasks [17]. Furthermore, as noted earlier, the results in this paper will be shown relevant to some more expressive non-monotonic logics.

The paper is organized as follows. In the next Section, basic notions about Boolean logic are recalled. The subsumption issue is presented in Section 3. In Section 4, the iteration of the subsumption-freeness enforcement process is investigated. Computational experimental results are provided and discussed in Section 5. Section 6 focuses on extending the results to a generic form of non-monotonic logic. The paper ends with perspectives and promising paths for further research.

2 Logic-Based Framework

To concentrate on the aforementioned conceptual problems, we consider the very simple framework of standard Boolean logic. Let $\mathcal{L}$ be a language of formulas over a finite alphabet $\mathcal{P}$ of Boolean variables, also called *atoms*. Atoms are noted $a, b, c, \ldots$ The $\wedge, \vee, \neg, \Rightarrow$ and $\Leftrightarrow$ symbols represent the standard conjunctive, disjunctive, negation, material implication and equivalence connectives, respectively. A *literal* is an atom or a negated atom. Formulas are built in the usual way from atoms, connectives and parentheses; they are noted $f, g, h, \ldots$ A formula is in conjunctive normal form (CNF) when expressed as a conjunction of clauses, where a clause is a disjunction of literals. The semantical concepts needed in the paper are as follows. Let Ω denote the set of all interpretations of $\mathcal{L}$, which are functions assigning either *true* or *false* to every atom. A *model* of a set of formulas KC is an interpretation of Ω that satisfies every formula of KC. KC is *consistent* when its set of models is not empty. $KC \vdash f$ expresses that the formula f can be deduced from KC, i.e. that it is *true* in all models of KC.

From a syntactical point of view, a *knowledge component* KC will thus be a set of formulas of $\mathcal{L}$. We opt for a semantical (vs. a purely syntactical) regard of KC. Under this point of view, KC is identified with the set of all its deductive consequences. Anyway, we will still distinguish between the formulas that are *explicitly* present in KC vs. formulas that are only *implicit* deductive consequences of the formulas that are explicitly present in KC. For convenience purpose, KC will be represented by its set of explicit formulas.

A word of caution might also be needed for readers who are familiar with rule-based systems but not with logic. We exploit the full (sound and complete) inferential capability of Boolean logic, i.e. we do not only simply allow for mere forward and backward chaining on $\Rightarrow$ as in traditional rule-based systems. For example, from the rule $a \Rightarrow b$ and $\neg b$, we infer $\neg a$ using contraposition. Also, the reader not familiar with logic should always keep in mind that a rule of the form $(a \wedge b \wedge \neg c) \Rightarrow (d \vee e)$ is logically equivalent to $\neg a \vee \neg b \vee c \vee d \vee e$, and will be treated as such. Actually, in the following, we consider clausal KCs and clauses, only.

3 Preempting Subsumption

In this Section, the approach in [14] for preempting subsumption in the Boolean framework is briefly summarized.

Two central concepts in this paper are the *strict implicant* and the *subsumption* ones, defined as follows.

Definition 1. *Let f and g be two formulas. f is a strict implicant of g iff $f \vdash g$ but $g \not\vdash f$. KC subsumes g iff $KC \vdash f$ for some strict implicant f of g.*

For example, $KC = \{office \vee home\}$ subsumes $office \vee home \vee car$. Indeed, $office \vee home$ is a strict implicant of $office \vee home \vee car$. Obviously, f (as mentioned in $KC \vdash f$) needs not be explicit in KC but can be a mere implicit formula in KC, i.e. a deductive consequence f of the explicit formulas of KC such that f is not itself explicit in KC.

In [14], it has been shown that if the choice is to modify KC before inserting a new formula g that must not be subsumed, then it is necessary to first delete from KC all possibilities to deduced $g \Rightarrow f_i$ for any strict implicant f_i of g. Alternatively, g can be introduced inside KC; then, all strict implicants of g must be expelled from the augmented KC. Interestingly, when the CNF format of the formulas is considered, it is sufficient to expel the longest strict implicants, in terms of the number of involved literals. In the above example, e.g. ensuring that the strict implicant $office \vee home$ of $office \vee home \vee car$ is expelled is sufficient to guarantee that e.g. the smaller implicant $office$ is also expelled (otherwise we would have $office \vdash office \vee home \vdash office \vee home \vee car$).

In order to check whether a clause is subsumed or not, a first straightforward preprocessing step could check whether any of its n strict longest sub-clauses is actually explicit in the component. This can be performed efficiently in $O(n, m)$, where m is the total number of clauses in the component. This check can actually easily look for the presence of smaller implicants as well. Obviously, this would not detect more complex subsumption paths. From a computational point of view, checking whether a formula subsumes another one is $coNP$-complete, and thus intractable in the worst case. However, recent dramatic progress in Boolean and search makes it often possible to get answers within seconds, especially thanks to powerful SAT solvers [17, 18], which check whether a set of clauses is consistent or not.

In [14], we have experimented an original method to address the subsumption issue. The goal is not only to answer whether the formula is subsumed or not, but also to deliver clauses that can be required to be expelled in order to eliminate the subsumption links. The technique considers clausal KCs and clauses, only. It is based on the SAT-solvers technology and on methods [17, 18] for delivering so-called MUSEs (Minimal Unsatisfiable Subsets). More precisely: assume we need to check whether KC (that contains g) subsumes g through a strict implicant f of g. The solver considers the new set of formulas $KC \cup \{\neg f\}$, which can only be inconsistent in case of subsumption. Then, the solver looks for the MUSes, namely the (cardinality-)minimal sets of clauses that are inconsistent. Making sure that at least one clause in each of the MUSes is dropped ensures that the subsumption link disappears. Expelling such clauses can be automatic, or the knowledge engineer can be asked whether she (he) really wants to drop them, or even be given the choice of selecting the clauses to be dropped in the MUSes. When he (she) prefers keeping these MUSes intact, she (he) is then conducted to revise his (her) former requirement about the subsumption-freeness status of g. This solver provides efficient results even for huge KCs, provided that the number of MUSes remains small [19]. As an alternative to computing all MUSes, the solver also allows to compute a cover of MUSes, which is formed of enough MUSes to explain all inconsistencies.

At this point, it is important to stress that the above subsumption-freeness enforcement process should often only concern the subpart of KC that concerns complex rules rather than mere facts. Indeed, for example, assume that the fact “Light is on” is in KC. Then this fact subsumes e.g. any rule having “Light is on” as a consequence, like “If the switch is on and the lamp bulb is ok then the light is on”. Accordingly, like in the first expert systems, we distinguish between a working memory (made of basic assertions translated as literals) and the so-called rule-based one, made of more complex formulas translating generic knowledge that is expected to be more permanent. In the following, KC will *always* denote this latter part of the set of formulas that must be taken into account to investigate and solve the subsumption issue.

Finally, the process can take into account a traditional concept of *integrity constraints*: these formulas cannot be expelled from KC to ensure the subsumption-freeness of a formula. The set of integrity constraints of KC is noted KC_{IC}.

4 Iterating the Process

When g is inserted inside KC and arrangements are made so that g is not subsumed by KC, two basic questions arise if we iterate the process. Assume that we now introduce another additional formula h in KC.

- If h must not be subsumed, what should be done if g is an obstacle for achieving this other subsumption-freeness insertion? In case h and g cannot be subsumption-free at the same time, what policy should be observed: do we need to give a higher priority to one of the formulas? In the positive case, which one?
- What should be done if the insertion of h conducts KC to subsume g?

Clearly, to address these questions, we do not only need to know, at each insertion step, which formulas in KC can be altered by the additional formulas to obey the subsumption-freeness requirements. We also need to know if all formulas that are to be subsumption-free must receive an equal treatment, or if some priorities should arbitrate between them.

First, we define two possible statuses for formulas in KC: a formula can be either *permissive* or *restrictive* with respect to subsumption. A restrictive formula in KC is an explicit formula in KC that is not only subsumption-free in KC but must remain subsumption-free if a new piece of information is then added into KC. We note KC_R the set of restrictive formulas in KC. No formula in KC_R is thus subsumed in KC. To some extent, elements of KC_R represent integrity constraints of a new type, made of formulas that cannot be logically strengthened. Also, like KC_{IC}, it is natural to expect KC_R to be small with respect to KC. Furthermore a small cardinality will ease the process from a computational point of view.

Let us stress that there is no *a priori* specific links between KC_R and KC_{IC}. It might happen e.g. that some or all elements of KC_R belong to KC_{IC} but those sets are independent in the general case.

Now, the set of permissive formulas in KC is given by $KC_P = KC \setminus KC_R$. Note that formulas in KC_P can be implicit and can (vs. must) be subsumed in KC.

As a case study, we investigate a form of preference for the more recently introduced information when two expectedly restrictive formulas cannot be subsumption-free at the same time. Accordingly, we suppose that KC_R is actually a LIFO-stack structure.

In Algorithm 1, the general skeleton of the algorithm for the iterated introduction of formulas involving the subsumption-freeness enforcement is provided, when all formulas are restricted to clauses.

Interestingly, the traversal of the stack of restrictive formulas is unique, since, because standard logic is monotonic, any deletion of formulas cannot alter the subsumption-freeness of remaining formulas. The subsumption-freeness enforcement policy considers the global set of MUSes for all longest strict implicants of g and requires each element to be broken by expelling one of its clauses, in interaction with the user or not. Alternative policies could require each MUS to be broken as soon as it is extracted, or compute and handle a cover of MUSes for each implicant. Also integrity constraints from KC_{IC} are protected from being expelled, although this is not detailed in these algorithms.

Obviously enough, the actual implementation of the algorithm should not be direct. Especially, redundant or useless computations should be avoided. Let us just stress on two features in that respect. First, to show that a formula is not subsumed by one of its longest strict implicants, models or counter-models are derived. They should be recorded so that they are checked when the same question arises at a next step of the process. Second, KC can be made of unrelated components. The newly introduced piece of information g does not need to be checked against parts of KC having no possible connection with g.

```
Algorithm 1: Main
  Data: –
  begin
      $KC \longleftarrow \emptyset$; $KC_{IC} \longleftarrow \emptyset$;
      $KC_R \longleftarrow empty\_stack()$;
      $finished \longleftarrow false$;
      while $finished$ is $false$ do
          write(”New clause: ”); read($g$);
          write(”Integrity constraint?: ”); read($is\_ic$);
          write(”Restrictive?: ”); read($is\_res$);
          construct($g, is\_ic, is\_res, KC, KC_{IC}, KC_R$);
          write(Finished?: ); read($finished$);
  end
```

```
Algorithm 2: construct
  Data: $g, is\_ic, is\_res, KC, KC_{IC}, KC_R$
  begin
      if $is\_ic$ is $true$ then
          $ic\_ok \longleftarrow$ check_IC($KC \cup \{g\}, KC_{IC} \cup \{g\}$);
          if $ic\_ok$ is $true$ then
              $KC_{IC} \longleftarrow KC_{IC} \cup \{g\}$;
          else
              interactWithUser($g, KC, KC_{IC}, KC_R$);
              exit;
      if $KC \cup \{g\}$ is inconsistent then
          revise($g, KC, KC_{IC}, KC_R$);
      $KC \longleftarrow KC \cup \{g\}$;
      if $is\_res$ is $true$ then
          push($g, KC_R$);
      enforce($KC, KC_{IC}, KC_R$);
  end
```

```
Algorithm 3: enforce
  Data: $KC, KC_{IC}, KC_R$
  begin
      while $g \longleftarrow$ nextElement($KC_R$) is not $null$ do
          $MUSes \longleftarrow \emptyset$;
          forall longest strict sub-clauses $f_i$ of $g$ do
              if $KC \cup \{\neg f_i\}$ is inconsistent then
                  $MUSes \longleftarrow MUSes \cup$
                  extractMUSes($KC \cup \{\neg f_i\}$);
          if $MUSes \neq \emptyset$ then
              breakMUSes($MUSes, KC, KC_{IC}, KC_R$);
  end
```

5 Experimental results

All experimentations have been conducted on a plain PC (IntelCore 2 Quad 2.66GHz - 4Gb Ram) under Linux Ubuntu 11.10 (3.0.0-16-generic). The solver is freely available from `http://www.cril.univ-artois.fr/~ramon/preempte`. In Table 1, a sample of typical experimental results is provided for the central and computationally-heavy part of the algorithm, namely the MUSes detection step. It presents the performance of the algorithm for that procedure, on usual SAT (i.e. set of Boolean clauses) benchmarks from earlier SAT competitions. The columns indicate the name of the benchmark, the number of involved clauses and of different variables, and the number of MUSes. Then, the performance of the extraction of one MUS and a cover of MUSes is then given, together with the size of the MUS and cover. The timeout was set to 250 seconds. Our belief is that in most real-life knowledge components, those MUSes will remain small (smaller than in those benchmarks made for challenging SAT solvers) and of a manageable number, since a new formula generally interacts with a small number of small subparts of the existing KC, only.

6 Pushing the Enveloppe

The present work has been developed within the standard Boolean logical setting. Interestingly, it also applies -with no additional computational cost- to a generic form of nonmonotonic logic that is suitable for representing rules with exceptions, where these exceptions can depend on consistency checks.

In the following, we resort to McCarthys Abnormality notation [20] to emphasize and encode possible exceptions to rules. Let $\mathcal{Ab}$ be a subset of $\mathcal{P}$. Its elements are noted $Ab_1, \ldots, Ab_m$ and called abnormality variables. They are intended to represent exceptions to rules that can be either

				Finding all MUSes		Finding a cover of MUSes	
instances	#cla	#var	#MUSes	#sec	#cla in MUSes	#sec	#cla in MUSes
battleship-5-8-unsat	105	40	1	0.23	105	0.18	105
battleship-6-9-unsat	171	54	1	1.88	171	0.48	171
battleship-10-10-unsat	550	100	-	*timeout*		19.58	417
battleship-11-11-unsat	726	121	-	*timeout*		82.21	561
battleship-12-12-unsat	936	144	-	*timeout*		172.93	711
5cnf_3500_3900_30f1.shuffled	420	30	-	*timeout*		17.01	194
5cnf_3900_3900_060.shuffled	936	60	-	*timeout*		33.99	777
marg3x3add8ch.shuffled-as.sat03-1448	272	41	1	177.15	272	7.10	272
marg3x3add8.shuffled-as.sat03-1449	224	41	1	8.79	224	3.48	224
php_010_008.shuffled-as.sat05-1171	370	80	-	*timeout*		1.14	370
rand_net60-30-1.shuffled	10 681	3 600	-	*timeout*		233.46	10 681
C208_FA_UT_3254	6 153	1 876	17 408	2.94	98	17.49	40
C208_FA_UT_3255	6 156	1 876	52 736	6.67	102	18.97	40

Table 1. A Sample of Typical Experimental Results for SAT Benchmarks (MUSes detecting step).

deduced through the standard logic deductive apparatus, or assumed inexistent by default.

For example, the rule "If the switch is on and the lamp bub is ok and if it can be consistently assumed that the switch is ok then the light is on" can be represented by the knowledge component: $KC = \{switch\_on \wedge \neg Ab_1 \wedge \neg Ab_2 \Rightarrow light\_on, \neg switch\_ok \Rightarrow Ab_1, lamp\_bulb\_ok \Leftrightarrow \neg Ab_2\}$.

Intuitively, KC is considered under a non-monotonic reasoning schema in the following way. Assume that we take into account in KC the additional contextual factual information $switch\_on$ and $lamp\_bulb\_ok$. Then, all Ab_i are considered. Roughly, when neither Ab_i nor $\neg Ab_i$ can be established then $\neg Ab_i$ is assumed by default. In the example, $\neg Ab_i$ is assumed by default and KC allows $light\_on$ to be inferred.

More formally, the non-monotonic inferential apparatus can be defined as follows:

Definition 2. *Let us index any model of KC by the set of Ab_i atoms that the model satisfies. A minimal model of KC is a model of KC such that no proper-subset of its index is the index of a model of KC.*

Definition 3. *A formula f can be non-monotonically inferred from KC (noted $KC \mid\!\sim f$) iff f is true in all minimal models of KC.*

A useful point here is that such a non-monotonic logic covers standard Boolean logic in the sense that whenever $KC \vdash f$ we also have that $KC \mid\!\sim f$. Now, a question is: in order to move up the subsumption issue to this non-monotonic framework, should we replace the standard entailment relation $\vdash$ by $\mid\!\sim$ in both the standard-logic definitions (see Definition 1) of subsumption and strict implicant?

Actually, we do not make such a step here for the following reason. First, remember that in the classical logic approach we do not consider the factual part of the knowledge to handle the subsumption issue. Now, in the absence of a proof of Ab_i from KC, we have that $KC \mid\!\sim \neg Ab_i$. Accordingly, we have that any formula of the form $\neg Ab_i \vee \ldots$ (equivalent to $\ldots \wedge Ab_i \Rightarrow$) is non-monotonically entailed from KC and would thus be subsumed by KC. Classifying all those formulas as subsumed ones would not meet the same objective that motivated our treatment of subsumption in classical logic. Indeed, these formulas would be subsumed because of additional factual assumptions made by default ($\neg Ab_1$ in the example). These assumptions do not necessarily represent the actual context of KC. In the same way that we have left apart from the subsumption issue the factual part of the knowledge translated by literals, we left here also apart the additional augmenting $\neg Ab_i$ assumptions. In this way, we keep our policy coherent with our motivation of separating the knowledge into two separate components: the factual information about actual circumstances on the one side, and more complex permanent rules on the other side. The factual information, either explicit in KC or by default, is not thus considered in this treatment of subsumption, be it classical or non-monotonic.

In this respect, by not taking into account these additional default assumptions, the treatment of subsumption in the non-monotonic setting collapses down to the treatment of subsumption within classical logic.

Obviously enough, this decision to consider subsumption according to the standard logic interpretation of a non-monotonic knowledge component cannot be justified for all non-monotonic logics and all circumstances. For example, in [15], we have refined the implicant and subsumption concepts to fit a finer-grained non-monotonic framework, where, among other things: 1. The treatment of exceptions is more local in the sense that not every possible default information is necessarily assumed. 2. It is decided to focus on the specific context of KC by including the involved factual information in KC. 3. The factual contextual information around the fired rules involving exceptions default is itself taken into account in the subsumption issue.

7 Conclusions and Perspectives

Enforcing subsumption-free formulas is a difficult task from both conceptual and computational perspectives. However, it is essential to avoid inadequate and unexpected inferences from a knowledge component. The contribution of this paper about this issue is twofold. On the one hand, we have shown how such an enforcement process can be iterated. On the second hand, we have shown that this procedure can be applied to a non-monotonic extension of classical logic that is suitable for representing rules with exceptions. This research opens many perspectives. First, it remains to extend the framework to more expressive logics. Especially, finite fragments of first-order logics are natural candidates in this respect. Also, adapting and grafting the approach to description logics and answer set programs is also a promising path for future research. We intend to pursue these lines of research in the future.

Acknowledgments

This work has been supported in part by the *Région Nord/Pas-de-Calais* and the EC through a FEDER grant.

References

[1] J. Doyle. A truth maintenance system. *Artificial Intelligence*, 12:231–272, 1979.

[2] M. Dalal. Investigations into a theory of knowledge base revision (preliminary report). In 7^{th} *National Conference on Artificial Intelligence (AAAI'88)*, volume 2, pages 475–479. Morgan Kaufmann, 1988.

[3] P. Z. Revesz. On the semantics of theory change: Arbitration between old and new information (preliminary report). In 12^{th} *ACM SIGACT-SIGMOD-SIGART Symposium on Principles Of Databases Systems (PODS'93)*, pages 71–82. ACM, 1993.

[4] V. S. Subrahmanian. Amalgamating knowledge bases. *ACM Transactions on Database Systems*, 19:291–331, 1994.

[5] R. Fagin, J. D. Ullman, and M. Y. Vardi. On the semantics of updates in databases. In 2^{nd} *ACM SIGACT-SIGMOD Symposium on Principles Of Database Systems (PODS'83)*, pages 352–365. ACM, 1983.

[6] E. Fermé and S. Hansson. AGM 25 years. twenty-five years of research in belief change. *Journal of Philosophical Logic*, 40:295–331, 2011.

[7] C. Alchourrón, P. Gärdenfors, and D. Makinson. On the logic of theory change: Partial meet contraction and revision functions. *Journal of Symbolic Logic*, 50(2):510–530, 1985.

[8] H. Katsuno and A. Mendelzon. On the difference between updating a knowledge base and revising it. In 2^{nd} *International Conference on Principles of Knowledge Representation and Reasoning (KR'91)*, pages 387–394. Cambridge University Press, 1991.

[9] P. Gärdenfors. *Knowledge in Flux: Modeling the Dynamics of Epistemic States*, volume 103. MIT Press, 1988.

[10] S. O. Hansson. *A Textbook of Belief Dynamics. Theory Change and Database Updating*. Kluwer Academic, 1999.

[11] S. Konieczny and R. Pino Pérez. On the logic of merging. In 6^{th} *International Conference on Principles of Knowledge Representation and Reasoning (KR'98)*, pages 488–498. Cambridge University Press, 1998.

[12] S. Konieczny and É. Grégoire. Logic-based information fusion in artificial intelligence. *Information Fusion*, 7(1):4–18, 2006.

[13] D. Zhang and É. Grégoire. The landscape of inconsistency: a perspective. *International Journal of Semantic Computing*, 5(3), 2011.

[14] Ph. Besnard, É. Grégoire, and S. Ramon. Enforcing logically weaker knowledge in classical logic. In 5^{th} *International Conference on Knowledge Science Engineering and Management (KSEM'11)*, pages 44–55. LNAI 7091, Springer, 2011.

[15] Ph. Besnard, É. Grégoire, and S. Ramon. Overriding subsuming rules. In 11^{th} *European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU'11)*, pages 532–544. LNAI 6717, Springer, 2011.

[16] Ph. Besnard, É. Grégoire, and S. Ramon. Logic-based fusion of legal knowledge. In *(submitted)*, 2012.

[17] 14^{th} *International Conference on Theory and Applications of Satisfiability Testing (SAT'11)*. LNCS 6695, Springer, 2011.

[18] `http://www.satlive.org`.

[19] É. Grégoire, B. Mazure, and C. Piette. Using local search to find msses and muses. *European Journal of Operational Research*, 199(3):640–646, 2009.

[20] J. McCarthy. Applications of circumscription to formalizing common sense knowledge. *Artificial Intelligence*, 28:89–116, 1986.

i^{2}Learning: Perpetual Learning through Bias Shifting

Du Zhang

Department of Computer Science
California State University
Sacramento, CA 95819-6021
zhangd@ecs.csus.edu

Abstract

How to develop an agent system that can engage in perpetual learning to incrementally improve its problem solving performance over time is a challenging research topic. In this paper, we describe a framework called i^{2}Learning for such perpetual learning agents. The i^{2}Learning framework has the following characteristics: (1) the learning episodes of the agent are triggered by inconsistencies it encounters during its problem-solving episodes; (2) the perpetual learning process is embodied in the continuous knowledge refinement and revision so as to overcome encountered inconsistencies; (3) each learning episode results in incremental improvement of the agent's performance; and (4) i^{2}Learning is an overarching structure that accommodates the growth and expansion of various inconsistency-specific learning strategies. Using mutually exclusive inconsistency as an example, we demonstrate how i^{2}Learning facilitates learning through bias shifting.

Keywords*: inconsistency, i^{2}Learning, inductive bias, bias shifting, perpetual learning agents.*

1. Introduction

One of the challenges in developing an agent system that can engage in perpetual learning to incrementally improve its problem solving performance over time is deciding on what will trigger its perpetual learning episodes. In this paper, we describe a framework called i^{2}Learning for such perpetual learning agents. i^{2}Learning, which stands for *inconsistency-induced learning*, allows for inconsistencies to be utilized as stimuli to learning episodes. The proposed framework has the following characteristics: (1) the learning episodes of the agent are triggered by inconsistencies it encounters during its problem-solving episodes; (2) the perpetual learning process is embodied in the continuous knowledge refinement and revision so as to overcome encountered inconsistencies; (3) each learning episode results in incremental improvement of the agent's performance; and (4) i^{2}Learning is an overarching structure that accommodates the growth and expansion of various inconsistency-specific learning strategies.

Inconsistencies are ubiquitous in the real world, manifesting themselves in a plethora of human behaviors and the computing systems we build [2,6,16-22]. Inconsistencies are phenomena reflecting various causes rooted in data, information, knowledge, meta-knowledge, to expertise [22]. As such, inconsistencies can be utilized as important heuristics in an agent's pursuit of perpetual learning capability. When encountering an inconsistency or a conflicting circumstance during its problem solving episode, an agent recognizes the nature of such inconsistency, and overcomes the inconsistency through refining or augmenting its knowledge such that its performance at tasks is improved. This continuous and alternating sequence of problem-solving episodes and i^{2}Learning episodes underpins the agent's incremental performance improvement process.

In this paper, our focus is on describing how the i^{2}Learning framework facilitates the development of perpetual learning agents that incrementally improve their performance over time. Because of its generality and flexibility, the i^{2}Learning framework accommodates different types of inconsistencies and allows various inconsistency-specific heuristics to be deployed in the learning process. Using mutually exclusive inconsistency as an example, we demonstrate how i^{2}Learning facilitates learning through *bias shifting*.

The rest of the paper is organized as follows. Section 2 offers a brief review on related work. Section 3 describes i^{2}Learning, the proposed framework for perpetual learning agents. In Section 4, we discuss the i^{2}Learning for a particular type of inconsistency, mutually exclusive inconsistency, and describe how an *iterative deepening* bias shifting process can be incorporated into the framework to accomplish the learning process. Finally, Section 5 concludes the paper with remarks on future work.

2. Related Work

The areas of related work to the results in this paper include: lifelong learning agent systems, learning through overcoming inconsistencies, and inconsistency-induced learning through bias shifting.

A never-ending language learner called NELL was described in [3]. NELL utilized semi-supervised learning methods and a collection of knowledge extraction methods to learn noun phrases from specified semantic categories and with specified semantic relations. NELL has four component learners: a pattern learner, a semi-structured extractor, a morphological classifier, and a rule learner. NELL also accommodates human interaction to approve or reject inference rules learned by the rule learner component.

The work in [1] reported an agent system called ALICE that conducted lifelong learning to build a set of concepts, facts and generalizations with regard to a particular topic directly from a large volume of Web text. Equipped with a domain-specific corpus of texts, some background knowledge, and a control strategy, ALICE learns to update and refine a theory of the domain.

The results in [10] defined continual learning to be a continual process where learning occurs over time, and time is monotonic. A continual learner possesses the following properties: the agent is autonomous; learning is embodied in problem solving, is incremental, and occurs at multiple time steps; and there is no fixed training set. Knowledge an agent acquires now can be built upon and modified later. A continual learning agent system called CHILD was described in [10].

YAGO2 is a large and extendable knowledge base capable of unifying facts automatically extracted from Wikipedia Web documents to concepts in WordNet and GeoNames [7]. YAGO2 exhibits its continuous learning capability by allowing new facts to be incrementally added to an existing knowledge base. Knowledge gleaned by YAGO2 is of high quality in terms of coverage and accuracy.

The results in [4] deal with clustering with inconsistent advice. Advice such as must-link and cannot-link can be incorporated into clustering algorithms so as to produce more sensible groups of related entities. Advice can become inconsistent due to different reasons. Clustering in the presence of inconsistent advice amounts to finding minimum normalized cuts [4].

Bias shifting can be a useful technique in the area of *transfer learning* [9]. However, there are a number of important differences between the aforementioned work and the focuses of research in this paper. (1) i^{2}Learning emphasizes on the stimulus for perpetual learning, i.e., the learning episodes of the agent are triggered by inconsistencies it encounters during its problem-solving episodes. This is not necessarily the focus in related work. (2) i^{2}Learning has a problem-solving slant, i.e., learning to incrementally improve performance for solving problems at hand, whereas the related work in [1,3,7,10] is primarily geared toward the general task of knowledge-acquisition, or building an ontology or a domain theory. (3) The learning episodes also differ: i^{2}Learning adopts discrete learning episodes (as triggered by conflicting phenomena), whereas the learning episodes in related work of [1,3,7,10] is largely continuous, not necessarily triggered by any events. (4) Inconsistencies are utilized as essential heuristics for the perpetual i^{2}Learning, whereas inconsistent advice in [4] is only used as constraint for clustering. (5) Most of the related work (with the exception of CHILD) is web-centric in the sense that learning is carried out with regard to web texts. i^{2}Learning, on the other hand, accommodates a broad range of heuristics in its learning process.

3. i^{2}Learning: A Framework for Perpetual Learning Agents

A perpetual learning agent is one that engages in a continuous and alternating sequence of problem-solving episodes and learning episodes. In such an alternating sequence, learning takes place in response to a whole host of stimuli, including inconsistencies encountered in the agent's the problem solving episodes. Learning episodes result in the agent's knowledge being refined or augmented, which in turn improves its performance at tasks incrementally. We use *learning burst* and *applying burst* to refer to reoccurring learning episodes and knowledge application (problem solving) episodes.

The proposed i^{2}Learning framework focuses on a particular scenario for the aforementioned perpetual learning agents: one in which the learning episodes of the agent are triggered by inconsistencies it encounters during its problem-solving episodes and the perpetual learning process is embodied in the continuous knowledge refinement and revision so as to overcome encountered inconsistencies.

A perpetual learning agent has the following components: (1) a knowledge base (KB) for persistent knowledge and beliefs, domain or ontological constraints, assumptions and defaults; (2) a meta knowledge base (mKB) for the agent's meta-knowledge (knowledge on how to apply domain knowledge in KB during problem solving); (3) a working memory (WM) that holds problem specific facts, and facts deduced with activated beliefs from KB; (4) a reasoning mechanism to facilitate problem solving process; (5) a component called CAL (Coordinator for Applying burst and Learning burst) that recognizes inconsistency and initiates learning bursts; (6) a bias space containing candidate biases for the learning process; and (7) a learning module i^{2}Learning that carries out inconsistency-induced learning to refine or augment KB, mKB, or WM (or any combination of the three) so as to overcome encountered inconsistencies. Figure 1 captures the structure of perpetual learning agents.

When a conflicting situation arises in WM during its problem solving process, the agent's CAL detects it, suspends the current problem solving session, initiates the next learning burst via passing the specific inconsistent circumstance to the learning module, and waits for the result from the learning module. The learning module i^{2}Learning in turn carries out the learning process by recognizing the type of inconsistency in the conflicting circumstance, and selecting the appropriate inconsistency-specific learning method or heuristics to explain, resolve, or accommodate the conflicting circumstance. Some human interaction may be required at this stage.

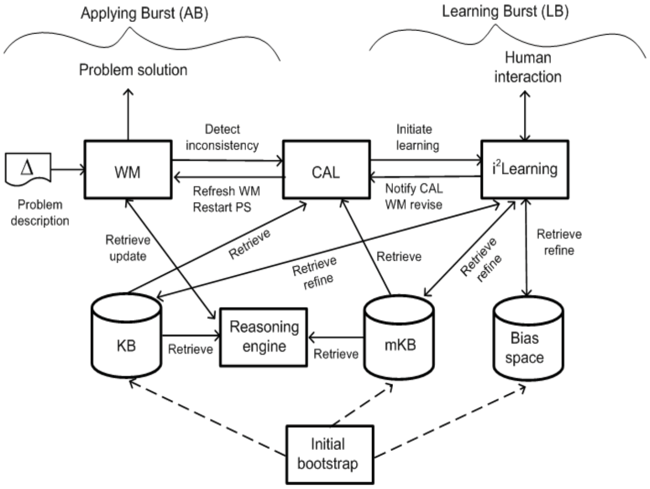


Figure 1. Framework of i^2Learning for Perpetual Learners.

The outcome of the learning process results in knowledge refinement or augmentation to KB, mKB, or WM, or all of them. Afterward, i^2Learning module notifies CAL of the learning result, and passes any WM revisions to CAL. CAL in turn refreshes WM with any changes from the i^2Learning module and restarts the problem solving session that was suspended earlier. This signifies the end of the current learning burst and the agent is ready to return to the problem solving episode to pick up where it left off. The agent will continue engaged in problem solving until it detects the next inconsistent scenario. Each such iteration results in an incremental performance improvement for the agent. Figure 2 illustrates the continuous nature of such perpetual learning agents. The logic of the i^2Learning module is given in Figure 3.

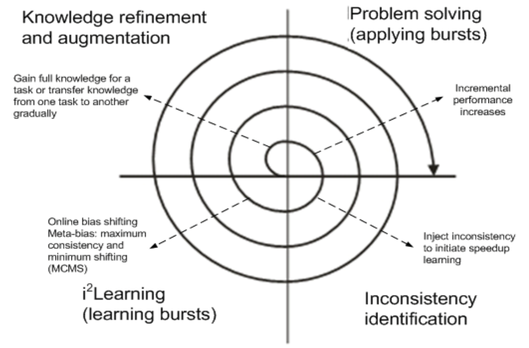

Knowledge refinement and augmentation
Problem solving (applying bursts)
Gain full knowledge for a task or transfer knowledge from one task to another gradually
Incremental performance increases
Online bias shifting Meta-bias: maximum consistency and minimum shifting (MCMS)
Inject inconsistency to initiate speedup learning
i^2Learning (learning bursts)
Inconsistency identification

Figure 2. Spiral Model of Perpetual Learning Agents.

The types of learning that help improve the agent's performance are essentially embodied in the types of inconsistency handling. There have been various classifications of inconsistent data, information, knowledge, and meta-knowledge in different domains [16-22].

```
1   Analyzing inconsistent scenario from WM;
2   Identifying category c and morphology m for
    the inconsistency;
3   Choosing appropriate learning method or
    heuristics;
4   Retrieving knowledge and bias;
5   Conducting $i^2$Learning(c, m)  {
6    case:
7     when $(c=c_i \wedge m=m_{ij})$: $i^2$Learning($c_i$,$m_{ij}$);break;
8     when $(c=c_k \wedge m=m_{kl})$: $i^2$Learning($c_k$,$m_{kl}$);break;
9     …………
10    else: default handling; break;
11  }
12  if  (human interaction is needed)  then  {
13       Query expert;
14       Receive human response;
15  }
16  if  (KB needs to be refined)  then  {
17       Refine KB;
18  }
19  if  (mKB needs to be refined)  then  {
20       Refine mKB;
21  }
22  if  (WM needs to be revised)  then  {
23       Revise WM and pass WM revisions to CAL;
24  }
25  Notify CAL to restart the current problem
26  solving session
```

Figure 3. The Learning Module.

The proposed framework is an overarching structure that accommodates growth and expansion in various inconsistency specific learning. Depending on which types of inconsistency the learning module can recognize and what corresponding algorithms or heuristics it comes equipped with in handling the inconsistency at hand, its inconsistency-induced learning capacities can change dynamically. The inconsistent scenarios an agent encounters at different points in time may be different and the learning strategies it adopts in the subsequent learning bursts can vary accordingly. The lines of 5–11 in Figure 3 can embody a rich set of inconsistency-specific learning algorithms.

There are two modes of i^2Learning: *problem-solving* mode, and *speedup* mode, each having different pace or rate for learning. When inconsistencies are encountered in WM during problem solving sessions, an agent works under the problem-solving mode and its pace of learning is driven by how frequent conflicting decisions or actions arise during knowledge application. On the other hand, inconsistencies can be intentionally injected into WM to induce learning bursts during the time when an agent is not engaged in problem solving. This latter case can be regarded as inconsistency-induced *speedup learning*. The primary objective of the agents is problem solving, and learning is just the means for agents to get progressively better at what they do.

4. i^{2}Learning(DEC, mex) via Bias Shifting

Inconsistencies or contradicting circumstances manifest themselves under different categories and in different forms of each category [22]. In this section, we describe how the learning process is carried out via a particular type of inconsistency, *mutually exclusive inconsistency*, in the category of declarative knowledge (DEC) expressed in terms of some declarative formalism (classic logic, defeasible logic, description logic, or default logic). The mutually exclusive inconsistency (mex) refers to the derivation of two literals containing mutually exclusive predicates that are syntactically different and semantically opposite of each other [15]. i^{2}Learning(DEC, mex) facilitates knowledge refinement through bias shifting.

4.1. Mutually Exclusive Inconsistency

Mutually exclusive inconsistency stems from knowledge in a KB that derives conclusions which are mutually exclusive and jointly exhaustive in a group of concepts. If P and Q are such predicates denoted as $P \not\asymp Q$ [16], then we have the following: $\forall x.[P(x) \supset \neg Q(x) \wedge Q(x) \supset \neg P(x)]$. For instance, if we have two firewall rules below expressed as first order formulas [19] in a firewall:

Allow(2, udp, 207.16.1.0, 24, 192.168.1.0, 24)
Deny(4, udp, 207.16.1.0, 24, 192.168.1.0, 24)

where the predicates "*Allow*" and "*Deny*" define the actions taken by the firewall regarding the UDP traffic (packets) from the source IP and the source port (IP of 207.16.1.0 and port 24) to the destination IP and the destination port (IP of 192.168.1.0 and port 24), then Rule 2 allows the traffic while Rule 4 denies it. The actions of "*Allow*" and "*Deny*" are mutually exclusive and joint exhaustive. Such an inconsistency in a firewall establishes a mutually exclusive inconsistency.

In order for CAL to detect the mutually exclusive inconsistency in WM, information must be available in mKB on sets of predicates that are considered mutually exclusive and jointly exhaustive:

$$Pred_{mex}=\{\{.., P, Q, ..\}_{\kappa} | \ (\kappa \text{ is a category of concepts}) \wedge (P \in \kappa) \wedge (Q \in \kappa) \wedge (P \not\asymp Q)\}$$

Definition 1. Let $ŧ$ and $\hbar$ be literals. We use $\hbar \ {}^{+}\!\!\leftarrow ŧ$ to denote a *derivation*, via some inference method, of length one or greater where $ŧ$ is the tail and $\hbar$ is the head of the derivation. For instance, if we have R: W(x) ← Q(x) and R': Q(x) ← P(x), then $W(x) \ {}^{+}\!\!\leftarrow P(x)$ where $ŧ = P(x)$ and $\hbar = W(x)$. Alternatively, we can also use the rule labels to denote the derivation $R \ {}^{+}\!\!\leftarrow R'$. □

Definition 2. Given an input data set Δ, we use the following notation to represent the fact that the derivations in $\{\hbar \ {}^{+}\!\!\leftarrow ŧ | \ (ŧ \in \Delta) \wedge (\hbar \text{ has properties X})\}$ are preferred over derivations in $\{\hbar' \ {}^{+}\!\!\leftarrow ŧ' | (ŧ' \in \Delta) \wedge (\hbar' \text{ has properties Y})\}$ with regard to Δ.

$$\frac{\Delta: \{\hbar \ {}^{+}\!\!\leftarrow ŧ | \ (ŧ \in \Delta) \wedge \hbar \ has\ properties\ X\}}{\{\hbar' \ {}^{+}\!\!\leftarrow ŧ' | (ŧ' \in \Delta) \wedge \hbar' \ has\ properties\ Y\}}$$

□

Definition 3. Let Ω denote KB and $\Im$ be a set of mutually exclusive literals which are represented as $\delta_p(\ddot{a})$ where δ_p indicates a literal with the predicate symbol p and $\ddot{a}$ is a vector of ground terms corresponding to domain elements. Let $\tilde{A}$ denote a set of *m features* $\{\alpha_1,\ldots, \alpha_m\}$ over which an inductive bias *b* is defined, and ϖ be a set of *weights* for features in $\tilde{A}$. Finally let $\acute{s}((\delta_p(\ddot{a}), \alpha_k(\ddot{a}))$ be a *support* function for a feature $\alpha_k(\ddot{a})$ in $\tilde{A}$ and a literal $\delta_p(\ddot{a})$ in $\Im$, and $S_A(\delta_p(\ddot{a}))$ be the *total support* of all the features in $A \subseteq \tilde{A}$ for a literal $\delta_p(\ddot{a})$ in $\Im$.

- $\Im = \{\delta_p(\ddot{a}), \delta_q(\ddot{a}) | \ (\Omega \vdash \delta_p(\ddot{a})) \wedge (\Omega \vdash \delta_q(\ddot{a})) \wedge (p \neq q) \wedge (\delta_p(\ddot{a}) \not\asymp \delta_q(\ddot{a}))\}$.
- $\tilde{A} = \{\alpha_1,\ldots, \alpha_m\}$ where $\tilde{A} \subseteq \Omega$ and α_i is a predicate symbol denoting a feature, $i \in [1,.., m]$.
- $b \subseteq \tilde{A}$ denoting an inductive bias of the learning process which consists of a subset of features.
- $\varpi(\tilde{A}) = \{ \ \omega(\alpha_k) \ |(\alpha_k \in \tilde{A}) \wedge (\omega(\alpha_k) \in [0, 1]) \wedge (\sum_{k=1}^{m} \omega(\alpha_k) = 1)\}$. In general, we can assume that features in $\tilde{A}$ have equal weights.
- $\acute{s}(\Im, \tilde{A}) = \{\acute{s}(\delta_p(\ddot{a}), \alpha_k(\ddot{a})) = \nu | \ (\delta_p(\ddot{a}) \in \Im) \wedge (\alpha_k(\ddot{a}) \in \tilde{A}) \wedge (\nu \in \{+1, -1, 0\})\}$
  - $\acute{s}(\delta_p(\ddot{a}), \alpha_k(\ddot{a})) = +1, \quad \text{if } \Omega \vdash (\delta_p(\ddot{a}) \ {}^{+}\!\!\leftarrow \alpha_k(\ddot{a}))$
  - $\acute{s}(\delta_p(\ddot{a}), \alpha_k(\ddot{a})) = -1, \text{ if } \Omega \vdash (\neg\delta_p(\ddot{a}) \ {}^{+}\!\!\leftarrow \alpha_k(\ddot{a}))$
  - $\acute{s}(\delta_p(\ddot{a}), \alpha_k(\ddot{a})) = 0, \text{ if } \Omega \nvdash [(\delta_p(\ddot{a}) \ {}^{+}\!\!\leftarrow \alpha_k(\ddot{a})) \wedge (\neg\delta_p(\ddot{a}) \ {}^{+}\!\!\leftarrow \alpha_k(\ddot{a}))]$
- $S_A(\delta_p(\ddot{a})) = \sum_{i=1}^{k} \omega_i \ \acute{s}\left(\delta_p(\ddot{a}), \alpha_i(\ddot{a})\right)$,
  where $A = \{\alpha_1,\ldots, \alpha_k\} \subseteq \tilde{A}$. □

4.2. Machine Learning Bias and Bias Shifting

A bias in a machine learning algorithm refers to any considerations the algorithm uses in selecting some hypotheses over other hypotheses [8]. Any learning algorithm must have a bias, which can be *static* (bias established at the outset of learning and remained unchanged in a learning algorithm), or *dynamic* (bias shifting from one to another during learning) [5]. *Bias shifting* is a multi-dimensional process that: (1) can be triggered by different events such as the availability of fresh training cases or the occurrence of inconsistency; (2) can result in the next bias to be selected either from a pre-established bias sequence or through evaluating potential alternative biases [5]; (3) can be conducted as either an *offline* or an *online* process [5]; (4) can rely on different shifting approaches such as bias strengthening or weakening methods; and (5) should itself have a bias (meta-bias on preference in selecting the next bias) [5].

Several properties exist about biases, *correctness* and *strength* [14], and *conservation* law [12]. A correct bias is one that defines a hypothesis space that contains the target concept to be learned. A *strong bias* is one that results in a small hypothesis space whereas a *weak bias* yields a large hypothesis space. There are pros and cons for strong and weak biases [13,14]. The conservation theorems indicate that though no single bias works well under all circumstances, learning algorithms with dynamic bias through bias shifting perform better than algorithms with static bias under certain conditions [12].

The performance criteria for biases include: *predictive accuracy* of learned hypothesis, *efficiency* in a bias guiding through the learning process, and *readability* of biases [5]. Finally, it should be noted that there is a discrepancy between the aforementioned machine learning bias and the statistical bias [13]. Our focus in this paper is on the machine learning bias. The bias shifting process in our work is triggered by the occurrence of inconsistency and carried out as an online evaluation of potential candidate biases. Given a bias as consisting of a feature set, the shifting methods are based on strengthening or weakening a bias through adding or removing features with regard to the initial feature set. The performance criterion for the bias shifting is to improve the predictive accuracy of the existing hypothesis so as to overcome inconsistencies.

Definition 4. A bias b is *consistent* with regard to $\Im$ if the hypothesis h_b obtained via b has the following property:

$h_b \vdash$ (one of the elements in $\Im$, not both) □

The bias shifting process itself must have a bias which is in meta-bias space [5]. In the proposed framework, the meta-bias that underpins the i^{2}Learning process is referred to as *Maximum Consistency-Minimum Shifting*, or MCMS for short. The existing bias b in the previous learning episode contains a set of features that produced a model that does not support generalization in a consistent manner, as evidenced by the set of inconsistent literals $\Im$ derived during problem solving. Perpetual learning is embodied in the continued process of overcoming inconsistencies, which in turn relies on bias shifting so as to refine the previously learned model. Bias shifting amounts to refining the feature set used in defining the bias. The current bias b can be either strengthened or weakened through adding or removing feature(s) from its defining set. The minimum shifting is embodied in removing or adding one feature at a time to revise b. The objective is to have a refined bias b' that is consistent.

4.3. i^{2}Learning(DEC, mex)

We use the *iterative deepening* search [11] to carry out the bias shifting process. The feature set for the current bias b is the starting node in the search space. Removing a feature from, or adding a feature to, b generates a successor node (child node) b' of b. If b' results in a hypothesis $h_{b'}$ that produces a consistent generalization to circumvent the inconsistency manifested in $\Im$, then b' is returned as the refined bias. $h_{b'}$ will guide the refinement process to KB and mKB. On the other hand, if all the possible successor nodes obtained through removing/adding one feature from b cannot lead to a hypothesis that produces a consistent generalization to resolve the inconsistency in $\Im$, then the iterative deepening search will repeat this process by generating the successor nodes to b'. The depth of the search for a consistent bias is iteratively deepened, hence capturing the essence of minimum shifting. Each time when a new $\Im$ is encountered during the agent's problem solving episode, the learning module is engaged in overcoming the inconsistency continuously, which enables the continuous knowledge refinement that incrementally improves the agent's performance at its tasks, the essence of perpetual learning.

We define the following algorithms to be used by i^{2}Learning(DEC, mex):

- `IDBR(bias)`: for iterative deepening bias refinement, which takes a bias (feature set) as its input and returns either a consistent bias or failure. Here we only consider the feature-removal case in bias refinement.
- `RFR(node, limit)`: for recursive feature removal, which takes a feature set and a search depth as its input and returns either a consistent bias or failure/leaf.
- `biasTest(node)`: for checking if a given feature set can be a consistent bias, which takes a feature set as its input and returns the truth value of true or false.
- `sucSet(node)`: for creating a successor set for a given feature set, which takes a feature set as its input and returns a successor set.

Algorithms for i^{2}Learning(DEC, mex)

Input: $\Im$, $\tilde{A}$, KB, mKB, Δ, ϖ, *bias* (feature set);
Output: *bias'*, a consistent bias; $h_{b'}$, refined model;

```
IDBR(bias) {
  for depth = 0 to |bias|-1 do {
    result = RFR(bias, depth);
    if (result $\neq$ leaf) then {return result}
}
```

```
RFR(node, limit) {
if (biasTest(node)) then
      {return node} //consistent bias
else if (limit=0) then {return leaf}
else {
  leaf_Reached=false;
  for each successor in sucSet(node) do {
   result=RFR(successor,limit-1);
  if (result=leaf) then {leaf_Reached=true}
  else if (result$\neq$failure) then
                   {return result}
  }
  if (leaf_Reached) then {return leaf}
  else {return failure}
 }
}
```

```
biasTest(node) {
 obtain $h_{node}$;
 if ($h_{node}$ yields consistent generalizations)
 then {return true}
 else {return false}
}
```

```
sucSet(node) {
 successorSet = $\emptyset$;
 for each feature in node do {
  successorSet=successorSet $\cup${node–feature};
 }
 return successorSet;
}
```

4.4. An Illustrative Example

Now let us look at an illustrative example. Assuming we have the following KB for classifying animals and plants that is obtained based on a bias *b* of three features in Ã = {Shape, Cell_Walls, Cell_Functions} where Shape is about an object's morphology, Cell_Walls pertains to cell walls being non-rigid or rigid, and Cell_Functions defines if cells break *sugar* down to *carbon dioxide* or turn *carbon dioxide* into *sugar*. We rewrite *b* = { S, CW, CF} where S stands for Shape, CW for Cell_Walls, and CF for Cell_Functions.

R_1: CucumberShape(x)← SeaCucumber(x)
R_2: Cucumber(x)← CucumberShape(x)
R_3: Plant(x)← Cucumber(x)
R_4: NonRigidCellWalls(x)← SeaCucumber(x)
R_5: SugarToCarbonDioxide(x)← SeaCucumber(x)
R_6: Animal(x)← NonRigidCellWalls(x)
R_7: Animal(x)← SugarToCarbonDioxide(x)
R_8: ¬Animal(x)← RigidCellWalls(x)
R_9: ¬Animal(x)← CarbonDioxideToSugar(x)
R_{10}: Plant(x)← RigidCellWalls(x)
R_{11}: Plant(x)← CarbonDioxideToSugar(x)
R_{12}: ¬Plant(x)← NonRigidCellWalls(x)
R_{13}: ¬Plant(x)← SugarToCarbonDioxide(x)
R_{14}: ¬RigidCellWalls(x)← NonRigidCellWalls(x)
R_{15}: ¬CarbonDioxideToSugar(x)← SugarToCarbonDioxide(x)
R_{16}: ¬NonRigidCellWalls(x)← RigidCellWalls(x)
R_{17}: ¬SugarToCarbonDioxide(x)← CarbonDioxideToSugar(x)

Given a seacucumber sc as an input Δ={SeaCucumber(sc)}, we obtain Plant(sc) from {R_1, R_2, R_3}, and Animal(sc) from {R_4, R_5, R_6, R_7}. The given instance of sc cannot be a plant and an animal at the same time, hence ℑ = {Animal(sc), Plant(sc)} and Animal(sc) ≭ Plant(sc). We know that there is one piece of supportive evidence for Plant(sc) (via R_3← R_2← R_1) and two pieces of supportive evidence for Animal(sc) (via R_6← R_4 and R_7← R_5). In addition, we can also obtain two pieces of negative evidence about Plant(sc) (or two pieces of positive evidence about ¬Plant(sc)) (via R_{12}← R_4 and R_{13}← R_5). The issue becomes how to refine *b* through minimum shifting to generate a consistent bias *b*′. By calling **IDBR** with *b*={S, CW, CF}, the iterative deepening bias refining process takes place with regard to the search space in Figure 4.

The minimum shifting process will result in *b*′ = {CW, CF} being returned as a consistent bias. Table 1 shows the supports for elements in the successor set of *b*. In addition, the mKB Φ will be augmented with the following control information such that next time when an input Δ_i includes a ground atom for SeaCucumber(x), derivations of {R_6← R_4, R_7← R_5} are preferred and the derivation of {R_3←R_2← R_1} will be circumvented.

$$\frac{SeaCucumber(x):\quad \{R6 \leftarrow R4, R7 \leftarrow R5\}}{\{R3 \leftarrow R2 \leftarrow R1\}}$$

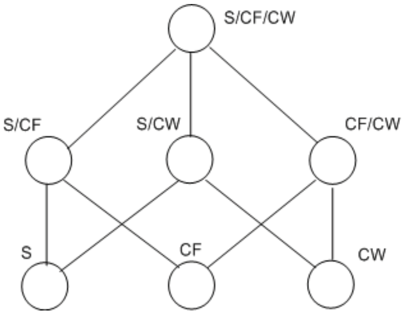


Figure 4. Search space.

Table 2. Support of {CW, CF} for Plant(sc) and Animal(sc).

	CellWalls	CellFunction		
Plant(SC)	0	0	0	**Total Support**
¬Plant(SC)	–0.5	–0.5	–1.0	
Animal(SC)	+0.5	+0.5	+1.0	
¬Animal(SC)	0	0	0	

5. Conclusion

In this paper, we describe i^{2}Learning, a framework for perpetual learning agents. i^{2}Learning allows the learning episodes of the agent to be initiated by inconsistencies the agent encounters during its problem-solving episodes. Learning in the framework amounts to the continuous knowledge refinement and/or augmentation in order to overcome encountered inconsistencies. An agent's performance at tasks can be incrementally improved with each learning episode. i^{2}Learning offers an overarching structure that accommodates the growth and expansion of various inconsistency-specific learning strategies. Through the mutually exclusive inconsistency, we demonstrate algorithmically how i^{2}Learning facilitates learning in terms of bias shifting.

The main contributions of this research work include the following. We demonstrate that learning through overcoming inconsistency is a viable and useful paradigm. The i^{2}Learning framework accommodates various inconsistency-specific heuristics to be deployed in the continuous learning process. Through iterative deepening bias shifting, an agent's performance can be incrementally improved by overcoming instances of mutually exclusive inconsistency.

Future work can be carried out in the following directions. Experimental work is needed on the iterative deepening bias shifting process for the mutually exclusive inconsistency case. Details of other frequently encountered inconsistencies and their respective learning heuristics still need to be fleshed out.

Acknowledgements. We express our sincere appreciation to the anonymous reviewers for their comments that help improve the content and the presentation of this paper.

References

[1] M. Banko and O. Etzioni, Strategies for Lifelong Knowledge Extraction from the Web. *Proc. of the Fourth International Conference on Knowledge Capture* (K-CAP), Whistler, BC, October 2007, pp.95-102.

[2] R.J. Brachman, and H.J. Levesque, *Knowledge Representation and Reasoning*. Morgan Kaufmann Publishers, 2004.

[3] A. Carlson, et al. T oward an Architecture for Never-Ending Language Learning. *Proc. of AAAI*, Atlanta, Georgia, July, 2010.

[4] T. Coleman, J. S aunderson, and A. Wirth, Spectral Clustering with Inconsistent Advice. *Proc. of 25th International Conference on Machine Learning*, Helsinki, Finland, 2008, pp.152-159.

[5] D. Gordon and M. desJardins, Evaluation and Selection of Biases in Machine Learning. *Machine Learning Journal*, Vo.20, 1995, pp.1-17.

[6] R. Gotesky, The Uses of Inconsistency. *Philosophy and Phenomenological Research*. Vol. 28, N o. 4, 1968, pp.471-500.

[7] J. Hoffart et al, YAGO2: Exporing and Querying World Knowledge in Time, Space, Context, and Many Languages. *Proc. of the 20th International World Wide Web Conference*, Hyderabad, India 2011.

[8] T. Mitchell, The Need for B iases in Learning Generalization. Technical Report CBM-TR-117, Rutgers University, 1980.

[9] S.J. Pan and Q. Yang, A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering*, Vol.22, No.10, 2010, pp.1345-1359.

[10] Ring, M.B. CHILD: A First Step towards Continual Learning. *Machine Learning*, Vol.28, 1997, pp.77–105.

[11] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2010.

[12] C. Shaffer, A Conservation Law for Generalization Performance. *Proc. of 11th International Conference on Machine Learning*, 1994, pp.259-265.

[13] P. Turney, How to Shift Bias: Lessons from the Baldwin Effect. *Journal of Evolutionary Computation*, Vol.4, Issue 3, 1996, pp.271-295.

[14] P. Utgoff, Shift of Bias for Inductive Concept Learning. In R. Michalski, J. Carbonell, and T. Mitchell (eds.) *Machine Learning: An AI Approach, Vol.II*, Morgan Kaufmann, 1986, pp.107-148.

[15] D. Zhang, Fixpoint Semantics for Rule Base Anomalies, *the Proceedings of the Fourth IEEE International Conference on Cognitive Informatics,* Irvine, CA, August 2005, pp.10-17.

[16] D. Zhang, Taming Inconsistency in Value-Based Software Development. *Proc. of the Twenty First International Conference on Software Engineering and Knowledge Engineering*, Boston, July 2009, p p.450-455.

[17] D. Zhang, On Temporal Properties of Knowledge Base Inconsistency. *Springer Transactions on Computational Science* V, LNCS 5540, 20 09, pp.20-37.

[18] D. Zhang, Toward A Classification of Antagonistic Manifestations of Knowledge. *Proc. of Twenty Second International Conference on Tools with Artificial Intelligence*, Arras, France, 2010, pp.375-382.

[19] D. Zhang, The Utility of Inconsistencies in Information Security and Digital F orensics. In T. Özyer et al (ed.) *Recent Trends in Information Reuse and Integration*, Springer-Verlag, 2011, pp.381-397.

[20] D. Zhang, 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] D. Zhang, On Localities of Knowledge Inconsistency. *International Journal of Software Science and Computational Intelligence*, Vol.3, No.1, 2011, pp.61-77.

[22] D. Zhang and É. Grégoire, "The Landscape of Inconsistency: A Perspective," *International Journal of Semantic Computing*, Vol. 5, No. 3, S eptember 2011, pp.235-256.

Evolutionary Learning and Fuzzy Logic applied to a Load Balancer [*]

Francisco Calaça Xavier
Instituto de Informática
Universidade Federal de Goiás
chicocx@gmail.com

Max Gontijo de Oliveira
Instituto de Informática
Universidade Federal de Goiás
maxopala@gmail.com

Cedric L. de Carvalho
Instituto de Informática
Universidade Federal de Goiás
cedric@inf.ufg.br

Abstract

Load balancing is a fundamental requirement in the development of parallel and distributed applications. The emergence of computational grids increased the demand for more efficient load balancers. Fuzzy Logic and Evolutionary Learning shown to be effective in order to increase the performance of a load balancer. This paper presents a Fuzzy Inference System capable to assist a load balancing in a computational grid. Evolutionary Learning was used to generate rules for an Inference System, which evaluated the behavior of nodes when they are subjected to a workload. The obtained results proved that it is possible to improve the performance of a load balancer using a Fuzzy Inference System when the rules are previously generated according to a particular behavior of the system.

Keywords: Evolutionary Learning, Fuzzy Inference System, Load Balance

1. Introduction

The scalability problem appears with the increasing use of information systems. One way to scale these systems is by using load balancers.

This paper proposes the use of a dynamic load balancer. Software agents [3] are installed in the nodes that monitor the state of load and transmit this data to the load balancer. In load balancer, this state of load is used by a Fuzzy Inference System to change dynamically the rate of requests sent to each of these nodes.

In this proposal, initially we used Evolutionary Learning [2] to learn fuzzy rules according to the behavior of nodes. Subsequently, these rules were used in a Fuzzy Inference System that dynamically changes the behavior of the load balancer.

The rest of this paper is organized as follows. Initially, in Section 2, some related works are described. Section 3 introduces the concept of load balancing. Section 4 describes the Artificial Intelligence techniques used to build the load balancer developed in this work. In Section 5, a case study and the results obtained with our experiments is presented. Finally, we conclude the work in Section 6.

2. Related Work

Salehi et al [13] proposed the application of the ant colony technique in a load balancer. In this case, intelligent ants react proportionally to the conditions of the balancer's nodes. The load balancing is the result of the interaction between these ants over the nodes.

In the approach proposed by Salehi et al, if a node gets overloaded, it will create a new ant that will balance the load by sending the requests to less loaded nodes. This step will lead to balance the load between the nodes.

According to Wang et al [16], hosting databases on virtual machines (VMs) has great potential to improve efficiency of resource use and eases deployment of databases.

The work of Wang et al [16] considers the problem of allocating resources on demand to a virtual machine running a database that serves data to a large amount of dynamic and complex queries according to some Quality of Service(QoS) [1] requirements.

To solve this problem, it was proposed an autonomous resource management approach. This approach uses a fuzzy modeling to capture and evaluate the performance of a VM that hosts a database. Its goal is to take decisions regarding resource requirements for each virtual machine, anticipating their needs and changing dynamically work loads. The proposed solution results demonstrated that CPU and I/O disk can be efficiently allocated to the VMs running the database in order to meet QoS requirements.

Steady-State Genetic Algorithm for Extracting Fuzzy Classification Rules From Data (SGERD) [10] is a steady

[*] Sponsored by Fundação de Amparo à Pesquisa do Estado de Goiás - FAPEG and Centrais Elétricas de Goiás - CELG

[1] Quality of service (QoS) refers to several aspects that allow evaluating the quality of information transport in computer networks and communication systems.

state GA [11] [12] that aims the generation of a number Q of rules per class, following the Cooperative-Competitive Genetic Learning approach [8]. The generations (children) are finite and limited to the extent of the problem. The algorithm aims to keep the same amount Q of rules for each class. The fuzzy rules *if-then* that classify a particular problem with n attributes are represented in the following form:

$$R_j: \text{ If } x_1 \text{ is } A_{j1} \ldots \text{and } x_n \; is \; A_{jn}, \text{ then class } C_j \quad (1)$$

$$\text{for j = 1, 2, 3, \ldots N}$$

where $X = [x_1, x_2, \ldots x_n]$ is the vector of input patterns, $A_{ji}(i = 1, 2, \ldots, n)$ are the linguistic terms used and represent the antecedents of the rule R_j. C_j is the consequent class R_j and N is the fixed number of fuzzy rules necessary to classify the input patterns. This fuzzy classification rule R_j can also be expressed as $A_j \Rightarrow C_j$, where A_j are all the antecedents linguistic terms of the rule and C_j is the class that this rule classifies.

3. Load Balancer

Load balancing [5] is a technique used to evenly distribute the workload between two or more resources, such as network, CPU, hard drives, etc.. Its objective is to optimize the use of these resources, maximize performance, minimize response time and avoid overloading. As illustrated in Figure 1, the balancer is able to split requests among all nodes.

Two versions of load balancing problem have been investigated in the literature: static and dynamic [4]. The static version divides the workload equally among nodes, regardless of their individually load. The dynamic version changes the workload of a node as it increases. Therefore, this last type of balancer performs real time measurements of node workload in order to keep the whole system workload balanced.

Generally, load balancers use a “Round-Robin” task scheduler [14] [15]. In “Round-Robin” task schedulers, the tasks are stored in a circular queue. The scheduler runs this queue and allocates the resource for each of these tasks. Thus, each task has a probability of being executed equal to $\frac{1}{n}$, where n is the total number of tasks to be allocated.

Therefore, there may be an overload on the nodes due to several factors. One of these factors, which often occurs, is when a particular request causes an excessive consumption of resources, for example, when a very large file is transmitted. The node that receives this file will suffer a significant increase in its load which, inevitably will cause an imbalance. Another factor that may also occur, is when nodes are not identical, i.e., when there are different hardware receiving files. In this case, if the same amount of requests is sent to each of nodes, those who have small amounts of resources such as processing capacity and memory will work in their operating limits, causing a significant slowdown or even may fail.

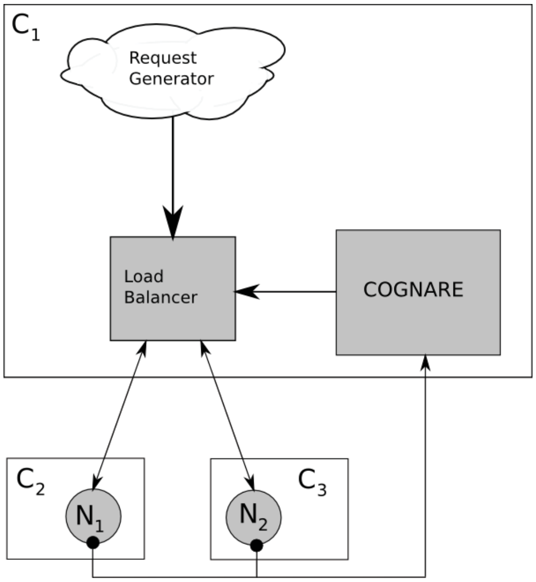


Figure 1: *Scheme for use and connection of computers used in our tests.*

4. Evolutionary Learning and Fuzzy Logic employed in a Load Balancer

Evolutionary Learning (EA) [2] is a machine learning technique, which uses elements of Evolutionary Computation, to extract knowledge from a database. Generally, this knowledge is represented by *if-then* type rules. These rules are easily understood by humans and easily processed by machines. Several branches of Evolutionary Computation such as Genetic Algorithms, Evolution Strategy, Evolutionary Programming, Particle Swarm, among others have been used in order to extract this knowledge in the form of rules.

A Fuzzy Rule Based System [8] [9] (FRBS) is composed of a Knowledge Base (KB), which contains information in the form of *if-then* fuzzy rules, the input layer, which contains the fuzzification interface, the fuzzy inference system, which together with the knowledge base performs inference from the input and output layer, which contains the defuzzification interface.

The fuzzy rules contained in the Knowledge Base have the following structure: “*if* a set of conditions are met (antecedent of the rule) *then* a set of consequences can be inferred (consequence of the rule).”

In addition to the rules, the knowledge base also has membership functions used in fuzzification and defuzzification processes of the values of input and output respectively.

This paper proposes the use of Evolutionary Learning and Fuzzy Logic to improve the performance of a load balancer. This assistance is done through the dynamic modification of the percentage of requests sent to each of nodes served by the balancer.

With this objective, a module called COGNARE was built. Its purpose is to perform a dynamic change in the rate of requests sent to each of the nodes served by a balancer. The inputs of this module are the current states of the hardware such as memory, CPU, disk, network, etc.. The output of this module is the allocation rate for each of the resource managed by the load balancer.

A sensor agent software [3] was installed in each of the nodes. Their function is to obtain the values of CPU load, memory, disk, etc. and send them to COGNARE, as illustrated in Figure 1. In this Figure, these agents are represented by black circles.

Initially, the rules needed by fuzzy inference system were learned. To that end, we sent requests to the balancer, without using COGNARE. The workload percentages of each node were changed in order to measure the processing speed variation of the whole system. This change was made randomly. After this, CPU and memory use of each node was measured. Table 1 shows this.

CPU N_1 (%)	CPU N_2 (%)	Memory N_1 (%)	Memory N_2 (%)	Rate (%) $N_1 \vert N_2$
64	21	17	19	79|21
53	47	13	22	63|37
7	89	11	52	11|89
87	13	18	13	81|19

Table 1: *Table used in learning the rules. Just some of the values obtained are showed.*

The membership functions, illustrated in Figure 2, were used to fuzzify values of the Table 1. In this Figure, VL, L, M, H and VH mean very low, low, medium, high and very high respectively. After this step, we generated Tables 2 and 3.

CPU N_1	CPU N_2	Memory N_1	Memory N_2	Rate N_1
H	L	L	L	L
M	M	L	L	H
L	H	L	L	VH

Table 2: *Values in Table 1 fuzzified. Output refers to the behavior of node 1.*

CPU N_1	CPU N_2	Memory N_1	Memory N_2	Rate N_2
H	L	L	L	H
M	M	L	L	L
L	H	L	L	VL

Table 3: *Values in Table 1 fuzzified. Output refers to the behavior of node 2.*

The initial population was obtained from all possible fuzzy rules which contain only one linguistic antecedent value. In this case, as shown in Figure 2, the total quantity of linguistic terms is 5 (VL, L, M, H and VH). The amount of linguistic variables is 2 (CPU and memory). Thus, the initial number of rules is $N = 5 * 2 \implies N = 10$. For each initial rule, it was calculated the consequent of their respective class. The consequent class is also one of the linguistic terms shown in Figure 2. As an example, if in Figure 1, the load node C1 is 23%, then the fuzzy linguistic term will be Low (L). If the load of the node C2 is 52%, then fuzzy linguistic term will be fuzzy Medium (M).

After applying the technique described by Mansoori [10] the rules are generated. Table 4 presents some of these rules.

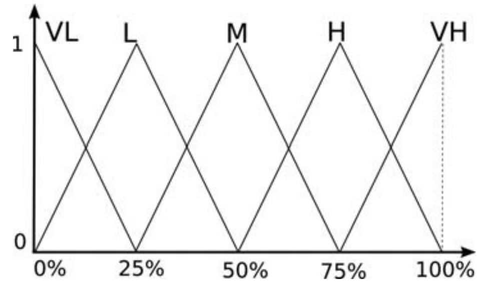


Figure 2: *Membership functions used for fuzzify the values in Table 1.*

No.	Rule
1	if CPU_1 is LOW and $MEMORY_1$ is MEDIUM then $RATE_1$ is HIGH
2	if CPU_1 is LOW and CPU_2 is MEDIUM then $RATE_1$ is HIGH

Table 4: *Some rules obtained after the learning process.*

Starting from the generated rules, we created a Fuzzy Inference System. The input variables are the values of the system CPU and memory of each node. The membership functions and fuzzy sets are described in Figure 2.

When the load balancer is working, COGNARE constantly run fuzzy inference system. This step recalculates the working rate of each node served by the load balancer.

In this way, COGNARE could change dynamically working rates for each node of the system.

5. Case study and evaluation of results

We used three computers in the same network in order to test the technique proposed in this work. The three computers have distinct setups. The Tables 5 and 6 show these setups. The resources of the computer named C_3 are lower than the other two. This fact helped to check the load balancer's efficiency in situations where exist differences among hardwares.

Name	Cores	CPU	RAM
C_1	4	800 MHz	4 GB
C_2	4	800 MHz	6 GB
C_3	2	1200 MHz	2 GB

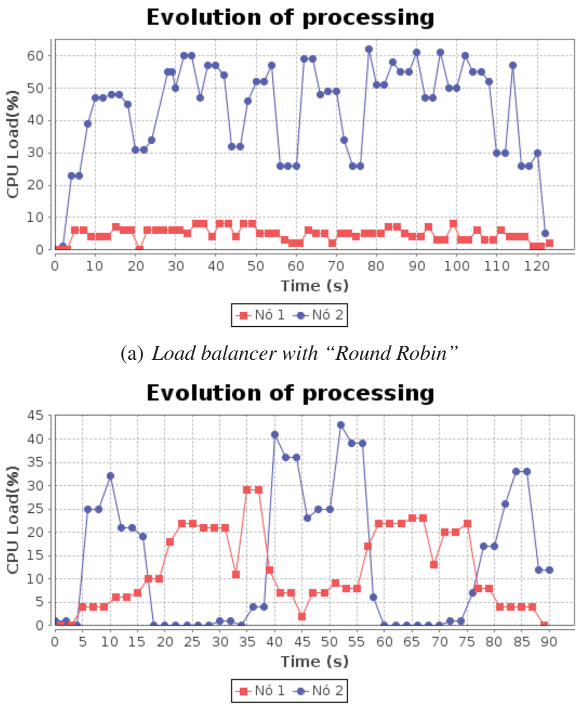
Table 5: *Configuration of the computers used in our tests.*

As illustrated in Figure 1, the computer C_1 generates requests to the load balancer and COGNARE. The computers C_2 and C_3 act like balancer nodes. The Table 6 describes each computer used in our tests.

Name	Setup
C_1	AMD Phenom(tm) II X4 955 Processor - 4 GB RAM
C_2	Intel(R) Core(TM) i7-2620M CPU @ 2.70GHz - 6 GB RAM
C_3	AMD Turion(tm) 64 X2 Mobile Technology TL-50 1.2GHZ - 2GB RAM

Table 6: *Computers used in our tests.*

In the performed tests, 2000 requests were sent to the load balancer. So, the total time to process all requests was measured. The average speed to process requests per minute was calculated. Therefore, a high speed means a most efficient system. The test was repeated 20 times. Among this 20 times, 10 times were executed with COGNARE changing dynamically the load of the nodes. The another 10 times were executed using a *Round-Robin* algorithm, where the allocation was made disregarding the differences between computers nodes.



(a) *Load balancer with "Round Robin"*

(b) *Load balancer with COGNARE*

Figure 3: *Behavior of nodes in the tests performed.*

After the tests, the average speeds were calculated. The Table 7 shows these values.

Test	Speed (requests per minute)
COGNARE	1484
Round Robin	804

Table 7: *Averages of the tests performed.*

As showed in the Table 7, the load balancer with COGNARE improved the system, increasing its processing capacity. This happened because the weaker hardware received fewer requests than the other hardware.

The Figure 3 shows the behavior of the nodes in the our tests. Using *Round-Robin (a)*, the computer 1 works very less than the computer 2 and the maximum CPU's usage was 62.5%. In the case *(b)*, using COGNARE, the system's performance was improved. The maximum CPU's usage was 42%. Moreover, despite the differences between the hardwares, using COGNARE, the computers showed a similar behavior.

6. Conclusion

This paper presented the use of Evolutionary Learning and Fuzzy Logic for dynamic allocation of resources in a load balancer.

As described in the previous sections, agents have been installed in each of the nodes. The aim of these agents was to measure the CPU and memory utilization and to send this information to COGNARE.

From the information regarding the status of the nodes, we extracted the fuzzy rules. These rules are used in a Fuzzy Inference System to change the rate of load of each node.

We believe that the main contribution of this proposal is the ability the system got with COGNARE to learn in advance the rules about its behavior. This previous learning have significantly reduced the workload during execution of the balancer, which caused a significant increase in performance of the system as a whole.

In future works, we plan to use other algorithms related to the Evolutionary Learning, such HIDER [1] [2], NSGA-II [6], TARGET [7], among others to improve COGNARE's learning process. We are also planning to do other experiments. This time we intend to use a larger number of nodes in order to better capture the efficiency of COGNARE.

The results of our experiments have revealed that COGNARE is capable of increasing the performance of a load balancer. Because of these results, COGNARE, is already in use, balancing the load of a high-demand system, used by Government of Goiás[2].

7. Acknowledgment

The authors wish to thank Fundação de Amparo à Pesquisa do Estado de Goiás - FAPEG and the Centrais Elétricas de Goiás - CELG by the grant which funded this project.

References

[1] J. S. Aguilar-Ruiz, R. Giráldez, and J. C. Riquelme. Natural encoding for evolutionary supervised learning. *IEEE Transactions on Evolutionary Computation, 11(4)*, pages 466–479, 2007.

[2] J. S. Aguilar-Ruiz, J. C. Riquelme, and M. Toro. Evolutionary learning of hierarchical decision rules. *IEEE Transactions on systems, man, and cybernetics-part B: Cybernetics, VOl. 33, No. 2, April*, pages 324–331, 2003.

[3] F. L. Bellifemine, G. Caire, and D. Greenwood. *Developing Multi-Agent Systems with JADE.* Wiley, 2007.

[4] M. Bramson, Y. Lu, and B. Prabhakar. Randomized load balancing with general service time distributions. *SIGMETRICS Perform. Eval. Rev.*, 38(1):275–286, June 2010.

[5] V. Cardellini, R. I, M. Colajanni, and P. S. Yu. Dynamic load balancing on web-server systems. *IEEE Internet Computing*, 3:28–39, 1999.

[6] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation, 6(2)*, pages 182–197, 2002.

[7] J. B. Gray and G. Fan. Classification tree analysis using target. *Computational Statistics Data Analysis, 52(3)*, pages 1362–1372, 2008.

[8] F. Herrera. Genetic fuzzy systems: taxonomy, current research trends and prospects. *Evolutionary Intelligence*, 1:27–46, 2008. 10.1007/s12065-007-0001-5.

[9] L.A. and Zadeh. Fuzzy sets. *Information and Control*, 8(3):338 – 353, 1965.

[10] E. Mansoori, M. Zolghadri, , and S. Katebi. Sgerd: A steady-state genetic algorithm for extracting fuzzy classification rules from data. *IEEE Transactions on Fuzzy Systems, 16(4)*, pages 1061–1071, 2008.

[11] V. Marques and F. Gomide. Fuzzy coordination of genetic algorithms for vehicle routing problems with time windows. In *Genetic and Evolutionary Fuzzy Systems (GEFS), 2010 4th International Workshop on*, pages 39 –44, march 2010.

[12] T. M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.

[13] M. A. Salehi, H. Deldari, and B. M. Dorri. Balancing load in a computational grid applying adaptive, intelligent colonies of ants, 2007.

[14] M. Shreedhar and G. Varghese. Efficient fair queuing using deficit round robin. *IEEE Trans. Net*, 1996.

[15] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster. Virtual Infrastructure Management in Private and Hybrid Clouds. *Internet Computing, IEEE*, 13(5):14–22, Sept. 2009.

[16] L. Wang, J. Xu, M. Zhao, Y. Tu, and J. Fortes. Fuzzy modeling based resource management for virtualized database systems. In *Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS), 2011 IEEE 19th International Symposium on*, pages 32 –42, july 2011.

[2] Goiás is one of 27 Federal Units of Brazil.

Using Social Networks for Learning New Concepts in Multi-Agent Systems

Shimaa M. El-Sherif[1], Behrouz Far[2]
Department of Electrical and Computer Engineering
University of Calgary
Calgary, Canada
e-mail: [1]smmelshe@ucalgary.ca
[2]far@ucalgary.ca

Armin Eberlein
Department of Computer Science & Engineering
American University of Sharjah
Sharjah, UAE
e-mail: eberlein@ucalgary.ca

***Abstract*—Traditionally, communication between agents in multi-agent systems is possible by committing to a common ontology. Unfortunately, this commitment is unrealistic and difficult to achieve in all cases. It is preferred in communication between agents to enable each agent to use its own conceptualization of its knowledge domain (i.e. each agent needs to use its own ontology). But that makes communication between agents more difficult and complex. In order to overcome this obstacle, agents need to negotiate the meaning of concepts and use their learning capability. Agents can learn new concepts they do not know but need in order to communicate with other agents in the system. This paper addresses the formation of new concepts in a multi-agent system where individual autonomous agents try to learn new concepts by consulting other agents. In this paper, individual agents create their distinct conceptualization and rather than commit to a common ontology, they use different ontologies. This paper uses positive and negative examples to help agents learn new concepts. It also investigates the selection of those examples and their numbers from teacher agents based on the strength of the ties between the learner agent and each teacher agent. A contribution of this paper is that the concept learning is realized by a multi-agent system in the form of a social network. We investigate using the concept of social networks in defining relationships between agents and show that it will improve the overall learning accuracy.**

Keywords- distributed knowledge management; concept learning; multi-agent system; social network; ontology

I. Introduction

Interest in Distributed Knowledge Management (DKM) systems has been growing due to their ability to solve real world complex problems that cannot be solved by centralized Knowledge Management systems. The challenges that DKM faces are:

- Representation of knowledge
- Distribution of knowledge
- Sharing of distributed knowledge

Ontologies can help represent knowledge and are therefore a good solution for the first challenge faced by DKM (representation of knowledge). Regarding the second challenge (distribution of knowledge), multi-agent systems (MAS) can handle distributed and heterogeneous environments. MAS is an environment in which different agents can interact with each other to solve complex problems. The third challenge is the most difficult issue for DKM: how to share and communicate knowledge and how to overcome the semantic heterogeneity. This is the main target of this paper.

In this paper we propose a concept learning system based on blending the heterogeneity of MAS and sharing capability of social networks. In our system, several MASs interact with each other to learn a new concept. Each MAS controls a repository of knowledgebase (i.e. concepts and their relations) that consists of an ontology with concept definitions and instances illustrating each concept. These MASs can interact with each other through a social network with varying strengths of ties between each two agents. The strengths of ties can be updated according to the frequency and type of interaction between agents. In the case study system, there is one learner agent that tries to learn a new concept from multiple teacher agents using this setup. Each teacher agent has its own ontology representation with its distinct understanding of the new concept. Teacher agents try to teach the learner agent this new concept the way they understand it by sending it positive and negative examples.

II. Background and Literature Review

In this section we will cover the main areas that are essential to understanding our proposed system. We will briefly describe multi-agent systems (MAS), ontologies and social networks. Then, we outline the current state of the art in ontological concept learning.

A. Multi-Agent System (MAS)

MAS can be defined as: “a loosely coupled network of problem solvers (agents) that interact to solve problems which are beyond the individual capabilities or knowledge of each problem solver” [1]. MAS is therefore a collection of heterogeneous agents, each of which with its own problem solving capability, able to locate, communicate and coordinate with each other.

In our system, agents perform major functionalities on behalf of each repository. The main roles of agents in our system are: handling query statements; managing concepts in the ontology hierarchy; learning a new concept (in the learner agent); selecting positive and negative examples for a certain concept (in the teacher agents); searching the ontology hierarchy for the best matched concept to teach the learner agent; managing tie strengths between agents; and finding peers.

B. Ontology

There are several definitions of ontology. We will use Daconta's definition [2] as it is relevant to our work: "Ontology defines the common words and concepts (meanings) and their relationships used to describe and represent an area of knowledge, and so standardize the meanings."

If two agents use the same ontology or ar e able to understand each other's ontology, communication between them is p otentially possible. Normally, diverse agents use different ontologies. In this case they need a mechanism to understand each other. In this paper, we illustrate how a single learner agent can learn new concepts from different teacher agents. Those teacher agents do not need to agree on the same definition of the new concept; their understanding of this new concept may be close but slightly different from each other.

C. Social networks

By social networks we do not mean Facebook, Twitter or other related web services. In this work, a social network is a set of actors (e.g. human, process, agent, document repository) and relationships between them. It can be represented as a set of nodes that have one or more types of relationships (ties) between them [3]. Using social networks gives us flexibility in dealing with the concepts in heterogeneous ontologies. It allows agents to understand the meaning of the same concept even though its definition might be slightly different in each agent's ontology.

The strength of a tie is affected by several factors. Granovetter [4] prop osed four dimensions that may affect tie strength: the duration of the relationship; the intimacy between the two actors participating in the relationship; the intensity of their communication with each other; and the reciprocal services they provide to each other. In social networks of humans, other factors, such as socioeconomic status, educational level, political affiliation, race and gender are also considered to affect the strength of ties [5]. Structural factors, such as network topology and information about social circles, may also affect the tie strength [6]. Gilbert et al [7] suggest quantitative measures (variables) for tie strength including intensity variable, days passed since the last communication and duration [7]. Another variable that may affect the strength of the tie is the neighborhood overlap variable [8] which refers to the number of common friends the two actors have. Petróczi et al [9] introduced mutual confidence between the actors of social networks. We proposed in [10] a method to calculate the strength of ties between agents in a social network using Hidden Markov Models (HMM) [11]. We showed that tie strength depends on several factors: Closeness factor: by measuring how close two agents are to each other (i.e. the degree of similarity between the two ontologies used by the two agents participating in the relationship); Time-related factor: combines all time factors that affect the strength of the relationship (e.g. duration of the relationship, frequency of communication between the two agents, time passed since the last communication); Mutual confidence factor: clarifying the nature of the relationship under measure, if it is a one-sided relationship or a mutual relationship. Then we built an HMM model to measure the strengths of ties between agents in a social network using those factors.

D. Literature review

In this section, we describe some of the previous work done in the concept learning area.

In [12], Steels uses a distributed multi-agent system where no central control is required. He describes a "language game" in which each agent has to create its own ontology based on its experience of its environment. The agents communicate with each other to stabilize to a common ontology or a shared set of lexicons to be used afterwards in their interactions.

Palmisano [13] uses a m achine-learning approach for creating a shared ontology among different agents that use different ontologies. The main purpose of Palmisano's work is to make each agent maintain its own ontology and at the same time keep track of the concepts' meanings in other agents' ontologies it communicates with.

Afsharchi tries in his work [14] to enable an agent to learn new concepts from a group of agents then represent this new concept in its own terminology without the need of a shared ontology [15]. He deals with a multi-agent system as a group of agents with different ontologies and any of them can learn a new concept by asking the other agent about this concept and then representing it using its own taxonomies.

III. CONCEPT LEARNING FRAMEWORK

In our framework, we assume that in a society of *n* agents Ag_1, Ag_2… Ag_n, each agent Ag_i controls a repository R_i (Figure 1). Each repository uses an ontology (O_i) that consists of a set of concepts, relationships. Each concept C in each repository possesses some supporting documents to represent instances of the concept.

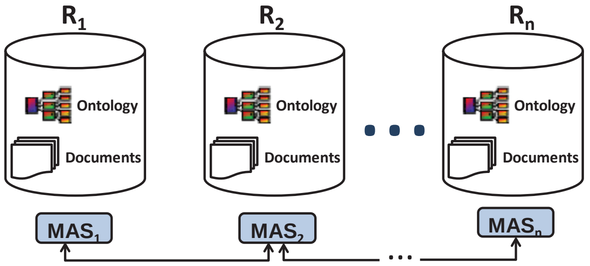


Figure 1. System Architecture

The goal of this paper is to advance the concept learning mechanism based on semantic interoperation between concept learning and semantic search modules based on MAS proposed in [16][17]. In this prototype system, each MAS controls a knowledgebase with a certain ontology. The system is expected to have two main modules: Concept Learning module [18]; and Semantic Search module [19]. In this paper we focus only on advancing the concept learning module by applying the social networks concept to the relationships between agents in order to improve the overall learning accuracy.

Figure 2 shows a flow diagram of tasks performed in the concept learning system. The learner agent (Ag$_L$) initiates the learning process by sending a learning request to all peer agents (teacher agents in this case). This request contains a query with all available information about the required concept C$_{goal}$ to be learned (e.g. concept name, keywords list, annotation information, feature/contextual information).

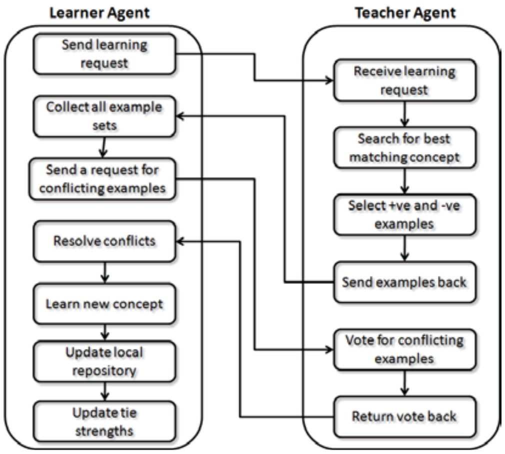


Figure 2. Flow diagram of tasks in the concept learning system

After receiving the initial query, each teacher agent (Ag$_T$) finds a concept C$_{best}$ that best matches the information in the initial query it receives. After finding C$_{best}$, Ag$_T$ chooses positive and negative examples representing C$_{best}$ based on the similarity between C$_{best}$ features and information sent in the initial query (see section IV below). The number of positive and negative examples chosen from each teacher agent depends on how socially close this teacher agent is to the learner agent (i.e. the strength of tie between Ag$_L$ and each Ag$_T$), because tie strength reflects how much Ag$_L$ trusts and depends on Ag$_T$. After selecting positive and negative example sets, each Ag$_T$ sends its own set to Ag$_L$.

Ag$_L$ collects all sets from all teachers to learn the new concept C$_{goal}$. Ag$_L$ checks for any conflict that may occur. As each teacher agent uses a different ontology and different representation of concepts in its ontology, the positive and negative examples chosen by teacher agents are expected to be different and some conflicts may occur as shown in Figure 3.

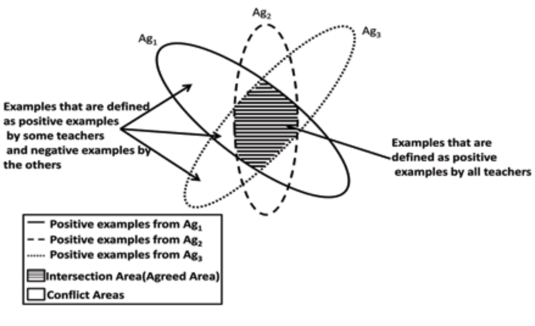

Ag$_1$
Ag$_2$
Ag$_3$
Examples that are defined as positive examples by some teachers and negative examples by the others
Examples that are defined as positive examples by all teachers
Positive examples from Ag$_1$
Positive examples from Ag$_2$
Positive examples from Ag$_3$
Intersection Area(Agreed Area)
Conflict Areas

Figure 3. Representation of conflict

Some examples are agreed on by all teachers to be positive examples. Other examples may be mentioned by some teachers and not mentioned by others, i.e., they are considered as a conflict. The worst case of conflict occurs when one or more examples are considered as positive examples by some teachers and as negative examples by others. These conflicts can be resolved by the learner agent sending back a conflict resolution request to all teacher agents asking them to vote on the conflicting examples [20]. Sending votes back to Ag$_L$ helps resolve the conflict based on votes of teacher agents and strength of ties between Ag$_L$ and each teacher agent. After resolving all conflicts, Ag$_L$ starts the learning process to learn the new concept C$_{new}$ considering the opinions of all teacher agents. Then Ag$_L$ updates its repository by representing the new concept C$_{new}$ in its ontology O$_L$.

Finally, Ag$_L$ measures the closeness between its updated ontology and the ontologies of all teacher agents. Based on this closeness and the interactions that occurred during the whole learning process, the strength of ties between Ag$_L$ and all teacher agents is updated [10].

IV. DOCUMENT CLASSIFICATION

In our system, we need to classify the documents used in all repositories. Document classification means assigning documents to one or more concepts using data mining methods.

A. Document Preprocessing

In this stage, we extract all features of textual documents, these documents are considered positive examples of concepts in the ontology used in our experiment, in order to process these documents with data mining algorithms. The main purpose of document preprocessing is feature extraction. It is the selection of a list of words (features) that best describe the document. At the beginning, documents are tokenized to get the entire list of words in the documents. Second, using a feature reduction technique to delete undesired words (words that are irrelevant to the document content), such as common English words like: a, an, in, for, the etc. Also we can create our own list of words that do not affect the document categorization. Next, word stemming is applied by removing suffix and prefix of words. In this case, we can treat words with the same stem as a single word. For example: mathematics and mathematical can be considered as the same feature in the document. Finally, a statistical metric is used to produce a feature vector that best represents the group of documents by calculating the relevance of each feature to the category it belongs to. In our experiment, we use Term Frequency plus Inverse Document Frequency (TF×IDF).

B. Document Representation

We select the top *n* words created for each category with the highest scores accompanied with their weight as a feature vector for each category. In our experiment, we set *n* to 20 as we noticed that in almost all documents, all relevant document features can be expressed with less than or equal to 20 words.

C. Document Categorization

This is the learning process. In this stage, we assign each group of documents to a certain category and calculate the accuracy of the learning technique used. In this paper, we use three different learning techniques: K-Nearest Neighbor (K-NN); Naive Bayes; and Support Vector Machine (SVM). In our experiment, we adopt the RapidMiner (http://rapid-i.com/content/view/181/190) tool to perform document classification.

V. Data set

Our data set consists of structured hierarchy ontologies of course syllabi of three universities; Cornell University, University of Michigan and University of Washington. In order to test our approach, we use three MASs as teacher agents: Ag_C, Ag_M and Ag_W. Each MAS controls a repository that contains one of the three ontologies for the course syllabi of the three universities. We also set up a learner agent Ag_L to learn some new concepts from the three teacher agents at the same time. We choose to learn the concept "computer science" because it has different representations among the three ontologies. The University of Michigan organizes "computer science" as an engineering discipline and as a joint program with Electrical Engineering. The University of Washington organizes "computer science" also as an engineering discipline but independent from Electrical Engineering and as a joint program with Computer Engineering. Cornell University considers "computer science" as an engineering discipline but independent from both Electrical and Computer Engineering.

VI. Experiment

In this experiment, we want to show how using social networks can improve the accuracy of the learning process. We follow the same strategy proposed in [17] in choosing positive and negative examples. This strategy depends on the value of sim(q_{spec}, C_{best}), where, sim(q_{spec}, C_{best}) is the ratio between the number of examples that meet the entered query and the total number of examples of chosen concepts C_{best}. If sim(q_{spec}, C_{best}) is greater than a specific threshold value (we set the value of the threshold to 0.6), this concept completely represents the concept to be learned. In this case, we can use any of its examples as positive examples. The negative examples can be chosen from its siblings (external negative examples). In this case, the siblings' examples are considered a good source of negative examples because the two concepts have the same parent which means they have some common features. At the same time, they are two different concepts, each with its own set of examples. So the examples of the siblings are considered discriminating examples. If sim(q_{spec}, C_{best}) is smaller than the chosen threshold value (i.e. 0.6), that means C_{new} overlaps with the concept C_{best}, which means that some of the examples of the concept C_{best} reflect the meaning of C_{goal} but the others do not. In this case, the returned documents contain only positive examples of C_{new}. The negative examples can be chosen from the rest of the examples (internal negative examples).

The scenario of our experiment is as follows:

1. No "computer science" concept is defined in the learner agent Ag_L. The learner agent Ag_L therefore uses only keywords to search for the best matching concept C_{best} in the teacher agents' ontologies. The keywords used are "(computer science) or (program language)". At the beginning, the strengths of ties between Ag_L and all teacher agents (Ag_C, Ag_M, Ag_W) are the same.
2. Each teacher agent searches its ontology for the best matching concept C_{best} that has the highest value of sim(q_{spec}, C_{best}). Depending on this value, each teacher agent picks sets of positive and negative examples.
3. Each teacher agent sends its own sets of positive and negative examples to the learner agent Ag_L.
4. Ag_L develops a new concept "computer science" in its ontology based on these examples using machine learning methods, and then calculates the accuracy of the learning process in each case.
5. Calculate the feature vector of the newly learnt concept "computer science" in Ag_L.
6. Calculate the closeness between the feature vector created for the new learnt concept and feature vectors of each C_{best} chosen by each teacher agent.
7. Depending on the closeness values calculated in step 6, set the initial tie strengths between the learner agent Ag_L and teacher agents Ag_C, Ag_M, Ag_W.
8. Repeat the learning process by using the same keywords used before as well as the feature vector created in step 5 based on the tie strengths calculated in 7 (repeat steps 2 to 5).

In order to measure the accuracy of the learned concept, we use the confusion matrix to measure the proportion of true results (i.e. true positive and true negative as opposed to false positive and false negative). The overall accuracy is calculated as follow:

$$Overall\ accuracy = \frac{True\ positive + True\ negative}{True\ positive + False\ positive + False\ negative + True\ negative} \quad (1)$$

VII. Results

The first step in our experiment in learning the new concept "computer science" is to find the best known concept C_{best} in teacher agents' ontologies.

At the beginning, the learner agent Ag_L does not have the concept "computer science" in its ontology. We use only keywords to learn this concept. Our keywords that describe the "computer science" concept are ("computer science" or "program language"). According to the ratio between the number of documents returned by the search and the total number of documents describing each concept, we can calculate sim(q_{spec}, C_{best}) for all concepts in each teacher agent's ontology. The concept with higher value of sim(q_{spec}, C_{best}) is chosen as the best matching concept C_{best} for each teacher. Table I shows the value of sim(q_{spec}, C_{best}) of the chosen concepts C_{best} from each university.

We choose the following concepts:

- "Computer Science E" from Cornell University
- "Electrical Engineering and Computer Science" from University of Michigan.
- "Computer Science and Engineering" from University of Washington.

For all universities, sim(q_{spec}, C_{best}) < 0.6 is chosen to be the threshold, so we select the negative examples internally from the documents of the chosen concepts. In this case, the strength of ties between Ag_L and all teacher agents Ag_C, Ag_M and Ag_W

are the same, i.e. social closeness is not considered, therefore the number of positive and negative examples from each teacher agent are the same. We got 21 positive examples and 21 negative examples from each teacher agent for those concepts. Those positive and negative examples are given to Ag_L to learn the new concept "computer science". We use three machine learning techniques: K-NN, Naive Bayes, SVM.

TABLE I. THE SIMILARITY VALUES OF THE SEARCH RESULTS. THE SEARCH KEYWORDS ARE ("COMPUTER SCIENCE" OR "PROGRAM LANGUAGE")

University/department	sim (q_{spec}, C_{best})
Cornell	
Computer science e	0.36
Michigan	
Electrical engineering and computer science	0.12
Washington	
Computer science and engineering	0.25

TABLE II. USING K-NN FOR LEARNING

	true CS	**false CS**
positive CS	52	23
negative CS	11	40

Overall accuracy = 73.02%

TABLE III. USING NAIVE BAYES FOR LEARNING

	true CS	**false CS**
positive CS	42	21
negative CS	21	42

Overall accuracy = 66.71%

TABLE IV. USING SVM FOR LEARNING

	true CS	**false CS**
positive CS	44	17
negative CS	19	46

Overall accuracy = 70.48%

Where, *CS* is the concept "Computer Science". In Tables II, III and IV, *true CS* represents the number of examples that are classified as positive examples of the CS concept; *false CS* represents the number of examples classified as negative examples of the CS concept; *positive CS* are the real positive examples of the CS concept; *negative CS* are the real negative examples of the CS concept.

After learning the new concept "computer science", we extract the feature vector of this concept. TF×IDF is used to extract the feature vector of each concept in all ontologies.

Now the learner agent Ag_L has the new concept "computer science" in its ontology. Ag_L has also a feature vector of this concept (See Table V). We need to update the strength of ties between the learner agent and each teacher agent. We measure the closeness between the feature vector of the new learnt concept "computer science" in Ag_L and all concepts used by teacher agents (i.e. "Computer Science e" from Ag_C, "Electrical Engineering and Computer Science" from Ag_M and "Computer Science and Engineering" from Ag_W).

The closeness values are shown in Table VI. From these values we can notice that, the learnt concept is closest in its definition to the concept of Cornell University, the next closest to University of Washington but far from the definition of the concept used by University of Michigan. We consider these closeness values as the initial values of tie strengths of our social network.

TABLE V. FEATURE VECTORS OF THE NEWLY LEARNT CONCEPT AND CHOSEN CONCEPTS FROM TEACHER AGENTS' ONOTOLOGYIES

Feature vector of the learnt concept (computer science): { program , languag , comput , system , scienc , cover , function , type , algorithm , design , learn , logic , object , analysi , compil , machin , unix , java , includ , model }

computer science e (Cornell University):
{ program , comput , system , languag , algorithm , logic , design , scienc , cover , analysi , equat , learn , model , discuss , function , network , optim , parallel , type , applic }

Electrical engineering and computer science (University of Michigan): { system , design , comput , circuit , model , analysi , program , optic , control , signal , algorithm , digit , applic , devic , network , languag , linear , perform , logic , communic }
computer science and engineering (University of Washington) { comput , system , design , program , algorithm , languag , softwar , analysi , parallel , model , imag , network , logic , altern , architectur , machin , simul , databas , memori , applic }

TABLE VI. THE CLOSENESS VALUES BETWEEN THE LEARNER AGENT AG_L AND TEACHER AGENTS AG_C, AG_M, AG_W

Teacher agent	closeness value
Cornell University (Ag_C)	0.490
University of Michigan (Ag_M)	0.087
University of Washington (Ag_W)	0.180

In order to refine the definition of the learnt concept "computer science", we use both keywords ("computer science" or "program language") and conceptual knowledge (the feature vector extracted) in searching for the best matched concept in teacher agents' ontologies. The selected best concepts are the same as in the first case. The number of positive and negative examples selected are proportional to the tie strengths, so we use 31 positive examples and 31 negative examples for the concept "Computer Science e" from Ag_C, 12 positive examples and 12 negative examples for the concept "Computer Science and Engineering" form Ag_W and 5 positive examples and 5 negative examples for the concept "Electrical Engineering and Computer Science" from Ag_M. We use those sets of positives and negative examples to teach Ag_L the concept "computer science" using K-NN, Naive Bayes and SVM.

TABLE VII. USING K-NN FOR LEARNING

	true CS	**false CS**
positive	39	9
negative	9	39

Overall accuracy = 81.21%

TABLE VIII. USING NAIVE BAYES FOR LEARNING

	true CS	**false CS**
positive	33	12
negative	15	36

Overall accuracy = 71.79%

TABLE IX. USING SVM FOR LEARNING

	true CS	false CS
positive	34	13
negative	15	34

Overall accuracy = 70.83%

Using K-NN as a learning technique, in the first phase of the experiment, the accuracy of the learning was 73.02%. In the second phase of the experiment it is increased to 81.21%. Using Naïve Bayes as a learning technique, in the first phase of the experiment, the accuracy of the learning was 66.71%. In the second phase of the experiment it increased to 71.79%. Using SVM as a learning technique, in the first phase of the experiment the accuracy of the learning was 70.48%. In the second phase of the experiment it increased to 70.83%. We can notice that the accuracy of the learning process improved with all three algorithms. As all parameters are the same in both phases of the experiments except for the strengths of ties between the learner agent Ag$_L$ and all teacher agents Ag$_C$, Ag$_M$ and Ag$_W$, we can conclude that this improvement is due to using social networks, i.e. using the tie strength, in our system. Using social networks helps the learner agent to better resolve conflicts that may occur during the learning process. According to the above tie strengths used, the learner agent Ag$_L$ is closer to Ag$_C$, which means that Ag$_L$ depends more on Ag$_C$ in learning new concepts (i.e. the learner agent trusts more the teacher agent that control the Cornell University knowledge base). That is why it uses more examples from this teacher in the learning process. The opposite occurs with Ag$_M$. The strength of the tie between Ag$_L$ and Ag$_M$ is the weakest, so the learner agent knows that the concept definitions in this ontology are far from its own ontology, so Ag$_L$ cannot trust Ag$_M$ much, i.e. the lowest number of examples are borrowed from this agent. That helps Ag$_L$ to improve the accuracy of the learning process. That is also reflected in the updated values of the tie strengths shown in table X. Ag$_L$ gets closer to Ag$_C$ and farer from Ag$_M$.

TABLE X. THE UPDATED TIE STRENGTH BETWEEN THE LEARNER AGENT AG$_L$ AND TEACHER AGENTS AG$_C$, AG$_M$, AG$_W$

Teacher agent	Tie strength
Cornell University (Ag$_C$)	0.51
University of Michigan (Ag$_M$)	0.04
University of Washington (Ag$_W$)	0.14

VIII. CONCLUSION

In this paper, we propose a new mechanism to be used in learning a new concept from multiple teacher agents each having the concept represented in slightly different way in their ontologies. This concept learning technique depends on sending positive and negative examples to identify the asked concept from teacher agents to learner agent. However, the number of positive and negative examples received from each teaching agent is decided by the strength of tie between the learner and teacher. Through a detailed experiment we showed that this improves the overall accuracy of the learning process. Although in the experiment we focused on a network of four nodes to illustrate the results, the method is general enough and can be applied to networks of large number of nodes.

ACKNOWLEDGEMENT

Armin Eberlein would like to acknowledge the financial contribution of the American University of Sharjah.

REFERENCES

[1] E. H. Durfee, V. Lesser, "Negotiating task decomposition and allocation using partial global planning," Distributed Artificial Intelligent, Issue 2, Vol 2, pp 229-244, 1989.

[2] Daconta, M.C., Obrst, L.J., Smith, K.T, "The semantic web. a guide to the future of XML," Web Services and Knowledge Management (Indianapolis (IN), USA 2003).

[3] Robert A. Hanneman, Mark Riddle, "Introduction to social networks methods," 2005.

[4] Granovetter, M., "The strength of weak ties: A network theory revisited, sociological theory," pp. 201 – 233, 1983.

[5] Lin, N., Ensel, "Social resources and strength of ties: Structural factors in occupational status attainment," American Sociological Review, vo;. 46, pp. 393 – 405, 1981.

[6] Burt, R., "Structural holes: The social structure of competition," Harvard University Press, 1995.

[7] Eric Gilbert and Karrie Karahalios, "Predicting tie strength with social network," Proceedings of the 27th international conference on Human factors in computing systems, Boston, MA, USA, pp. 211-220, 2009.

[8] J. P. Onnela, J. Saramaki, J. Hyvonen, G. Szabo, D. Lazer, K. Kaski, J. Kertesz, A. L. Barabasi, "Structure and tie strengths in a mobile communication network," Proceedings of the National Academy of Science of the United States of America, vol. 104, no. 18, 2007.

[9] Andrea Petróczi, Tamás Nepusz and Fülöp Bazsó, "Measuring tie-strength in virtual social networks," vol. 27, no. 2, INSNA, 2007.

[10] Shimaa M. El-Sherif, Behrouz Far, Armin Eberlein, "Calculating the strength of ties of a social network in a semantic search system using Hidden Markov Models," International Conference on Systems, Man and Cybernetics SMC 2011.

[11] Olivier Cappe, Eric Moulines and Tobias Ryden, "Inference in Hidden Markov Models", Springer, 2007.

[12] Luc Steels, "The origins of ontologies and communication conventions in Multi-Agent Systems," Autonomous Agents and Multi-Agent Systems, I(2), pp 169–194, 1998.

[13] Ignazio Palmisano, Luigi Iannone, Domenico Redavid, Gionanni, Semeraro, "Ontology alignment throug instnce negotiation: A machine learning approach," Lecture Notes in Computer Science, Vol. 4275, pp 1058–1074, 2006.

[14] Mohsen Afsharchi, Behroz H. Far, "Conceptual hierarchy learning in a Multi-Agent System," Proceedings of the Computer Society of Iran Computer Conference (CSICC), 2006.

[15] Mohsen Afsharchi, Behroz H. Far, Jorg Denzinger, "Learning non-unanimous ontology concepts to communicate with groups of agents," IAT, Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology, pp 211–217, 2006.

[16] B. H. Far, C. Zhong, Z. Yang and M. Afsharchi, "Realization of semantic search using concept learning and document annotation agents," The 21th Int. Conf. on Software Engineering and Knowledge Engineering SEKE 2009, 2009.

[17] Zilan (Nancy) Yang, "A practical ontology-based concept learning in MAS," master thesis, University of Calgary,Department of Electrical and Computer Engineering, 2010.

[18] Shimaa El-Sherif, Behrouz Far and Armin Eberlein, "Using social networking in resolving conflicts of concept learning process," Proceedings of IEEE CCECE, 2010.

[19] Shimaa El-Sherif, Behrouz Far and Armin Eberlein,"Semantic search based on multi-agent system and social networking," Proceedings of the sixth IASTED International Conference, ACSE, pp. 103-110, 2010.

[20] Mohsen Afsharchi, Behrouz H. Far, Jörg Denzinger, "Learning non-unanimous ontology concepts to communicate with groups of agents," IAT 2006: 211-217.

Identifying Coincidental Correctness for Fault Localization by Clustering Test Cases

Yi Miao[1,2], Zhenyu Chen[1,2], Sihan Li[1,2], Zhihong Zhao[1,2], Yuming Zhou[1]
[1] State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China
[2] Software Institute, Nanjing University, Nanjing, China
zhaozh@software.nju.edu.cn

***Abstract*—Coverage-based fault localization techniques leverage coverage information to identify the faulty elements of a program. However, these techniques can be adversely affected by coincidental correctness, which occurs when faulty elements are executed but no failure is triggered. This paper proposes a clustering-based strategy to identify coincidental correctness. The key rationale behind this strategy is that test cases in the same cluster have similar behaviors. Therefore, a passed test case in a cluster, which contains failed test cases, is highly possible to be coincidental correctness. Our experimental results show that, by cleaning or relabeling these possibly coincidentally correct test cases, the effectiveness of coverage-based fault localization techniques can be effectively improved.**

Keywords- coincidental correctness; cluster analysis; fault localization

1 INTRODUCTION

Coverage-Based Fault Localization (CBFL) leverages the execution information of both the failed test cases and passed test cases to assist the developer in identifying the program elements that induce a given failure. The intuition behind these techniques is that entities in a program that are primarily executed by failed test cases are more likely to be faulty than those that are primarily executed by passed test cases [1]. Although CBFL has shown promising results in previous studies, it is still necessary to further improve its effectiveness. One of the main challenges is the coincidental correctness problem.

Coincidental correctness occurs when a test case executes the faulty elements but no failure is triggered. The PIE model presented in J.M. Voas et al., [2] emphasizes that for a failure to be observed, the following three conditions must be satisfied: "*Execution*", "*Infection*", and "*Propagation*". The case is termed weak coincidental correctness, if the program produces the correct output when only the condition "*Execution*" is satisfied. The case is termed strong coincidental correctness, if the program produces the correct output when only the conditions "*Execution*" and "*Infection*" are satisfied. As the second condition of the PIE model has nothing to do with CBFL, strong coincidental correctness is not within the discussion of this paper. Hence, hereafter, coincidental correctness will refer to the former definition as it is more relevant to the execution profiles, and thus has more impact on CBFL.

Previous studies have demonstrated that coincidental correctness is prevalent in both two forms: strong and weak [3], [4]. W. Masri et al. have proven that coincidental correctness is responsible for reducing the safety of CBFL. More specifically, when coincidentally correct test cases are present, the faulty elements will likely be ranked as less suspicious than when they are not present. As shown in the previous studies [4], [5], the efficiency and accuracy of CBFL can be improved by cleaning the coincidentally correct test cases. However, it is difficult to identify coincidental correctness because we do not know the locations of faulty elements in advance.

In this paper, we propose a clustering-based strategy to identify the subset of test suite that is possible to be coincidentally correct. We apply cluster analysis to group test cases into different clusters. According to [6] and [7], test cases in the same clusters have similar behaviors. As such, a passed test case in a cluster, which contains failed test cases, is highly possible to be coincidental correctness because it has the potential to cover the faulty elements as those failed test cases do. We also present two strategies to deal with the coincidental correct test cases (see Section 3 for detail). By cleaning or relabeling these test cases, adverse effects of coincidental correctness can be reduced, which may lead to an increase of the efficiency and accuracy of coverage-based fault localization techniques. The experimental results show that our strategy is effective to alleviate the coincidental correctness problem and to improve the effectiveness of fault localization.

The remainder of this paper is organized as follows. Section 2 details the motivation of our work. Section 3 describes our approach to identify coincidentally correct test cases in detail. Section 4 presents the experimental work and results for the proposed strategy. Section 5 introduces some related work on coincidental correctness and cluster analysis on software testing. Finally, Section 6 presents our conclusions and future work.

2 MOTIVATION

2.1 Prevalence of Coincidental Correctness

In [3], Masri et al. demonstrated that coincidental correctness is prevalent in both its forms (strong and weak). Furthermore, the exhibited levels of weak coincidental correctness are much more significant than those of the strong one. To show the prevalence of the scenario under study, we conducted an experiment on the Siemens programs. The

The work described in this article was partially supported by the National Natural Science Foundation of China (90818027, 61003024, 61170067).

Siemens set contains seven C programs, and all of them can be downloaded from the SIR repository [17]. Each of the programs has a correct version, a number of faulty versions seeded with a single fault, and a corresponding test suite. We compare the output results of the correct versions with that of the corresponding seeded versions to determine the failures. Failures determined in this manner are called output-based failures. According to the execution profile, if a faulty element is executed during a test case, but no output-based failure is detected, we categorize the test case as coincidentally correct.

Our study only takes into account 115 seeded versions, and excludes the other versions because they contain code-missing errors or the faulty statements are not executable. Figure 1 summarizes the result. It illustrates the exhibited level of coincidental correctness is significant. The horizontal axis represents the percentages of coincidentally correct tests (each bar corresponds to a range of size 10%). The vertical axis represents the percentage of seeded versions that exhibited a given range. As can be seen, coincidental correctness is common in software testing.

2.2 Safety Reducing Effect

Denmat et al. [15] pointed out the limitation of CBFL and argued that the effectiveness of this technique largely depended on the hypothesis that executing the faulty statements leads most of the time to a failure.

In the following, we use Ochiai as an example to show that coincidental correctness is a potential safety-reducing factor. As shown in L. Naish et al., [16], the suspiciousness metric of Ochiai is defined as:

$$\mathrm{M(e)} = \frac{a_{ef}}{\sqrt{(a_{ef} + a_{nf})(a_{ef} + a_{ep})}}$$

e = faulty program element

a_{ef} = number of failed runs that execute e

a_{nf} = number of failed runs that do not execute e

a_{ep} = number of passed runs that execute e

Assume that there are k tests which execute e but do not raise a failure. Two strategies can be applied on these tests to improve the accuracy of the CBFL technique. The first strategy is to remove these tests from the test suite, that is, to subtract k from a_{ep}. Consequently, the suspiciousness metric will be:

$$\mathrm{M'(e)} = \frac{a_{ef}}{\sqrt{(a_{ef} + a_{nf})(a_{ef} + a_{ep} - k)}}$$

It is easy to know that $M(e) \leqslant M'(e)$. To verify:

$M'(e) \geqslant 0$, $M(e) \geqslant 0$ and $M'(e)/M(e) \geqslant 1 \Rightarrow M(e) \leqslant M'(e)$.

The second strategy is to relabel those tests from “passed” to “failed”, i.e., to subtract k from a_{ep} and add it to a_{ef}, the suspiciousness metric will be:

$$\mathrm{M''(e)} = \frac{a_{ef} + k}{\sqrt{(a_{ef} + a_{nf} + k)(a_{ef} + a_{ep})}}$$

It is easy to know that $M(e) \leqslant M''(e)$. To verify:

$M''^{2}(e) - M^{2}(e) \geqslant 0 \Rightarrow M(e) \leqslant M''(e)$. It can be seen that ignoring coincidentally correct test cases will leads to an underestimating of the suspiciousness of the faulty element.

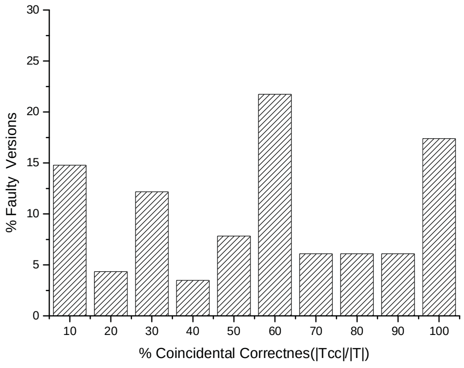


Figure 1. Frequency of Coincidental Correctness

3 METHODOLOGY

3.1 General Process

Some symbols we use throughout the rest of the paper are explained as follows:

T: the test suite used for a given program.

Tp: the set of passed test cases.

Tf: the set of failed test cases.

Tcc: the set of coincidentally correct test cases.

Ticc: the set of identified coincidentally correct tests.

Given a test suite *T*, which is comprised of *Tp* and *Tf*, the goal is to identify *Tcc* from *Tp*. The result is *Ticc*, and each element of *Ticc* is a potential candidate of the members of *Tcc*.

In this paper, we propose a clustering-based strategy to obtain *Ticc*. The goal of cluster analysis is to partition objects into clusters such that objects with similar attributes are placed in the same cluster, while objects with dissimilar attributes are placed in different clusters [7]. So, execution profiles are used as the features fed to a clustering algorithm. Specifically, test cases which execute the faulty elements and have similar execution profiles with the failed test cases are likely to be clustered together. Therefore, if a cluster consists of both failed test cases and passed test cases, the passed test cases within this cluster are very likely to be coincidentally correct. Note that our approach is based on the single-fault assumption. Multi-fault programs are not within the discussion of this paper, but will be explored in the near future.

For a developer to find the fault with the help of automatic fault-localization techniques, he/she can use the following procedure to take advantage of our strategy to improve the effectiveness of the diagnosis: First, a set of test cases is executed on the given program. As a result, each test case is labeled "passed" or "failed" according to its output result. Execution profiles which reveal the coverage information are collected at the same time. Then, clustering is conducted on the execution profiles. The next step is to identify coincidentally correct test cases using the method mentioned above, and the identified test cases are added to *Ticc*. Two strategies (cleaning or relabeling) can be employed to handle with the test cases that belong to *Ticc*, see Section 3.2.3 for details. Finally, a CBFL technique is applied to the refined test suite.

3.2 Detailed Technologies

3.2.1 Execution Profile Collection

We use gcov (GNU call-coverage profiler) [18] to obtain statement coverage information. For a test case, its statement coverage profile p_i = <e_1, e_2, ..., e_n>, where n represents the number of lines of the given program, and e_i = 1 if the *i*th line of code is executed, otherwise e_i = 0.

3.2.2 Cluster Analysis

The execution profiles of the test suite *T* are collected to form the input to the cluster analysis. The execution profile of each test case is regarded as an object to be clustered. The number of objects is equal to the number of test cases of *T*. In our context, n-dimensional Euclidean distance [19] is used as the distance function because it is easily calculated and widely used.

The simple K-means is employed as the clustering algorithm in our approach. We chose this algorithm because it is simple and fast. Besides, it performs reasonably well in our previous studies [6], [8]. It takes the number of clusters as a parameter. In our context, this number is set according to the size of *T*. Let CN denote the number of clusters, CN = |*T*|*p, where |*T*| is the number of test cases in *T* and 0 < p < 1.

Note that it is hard to decide a fixed ratio of the number of clusters. It mainly depends on the size of the test suite, and how much risk the developers are willing to take in order to identify the coincidental correct test cases. If the size of test suite is large, a relatively low value of *p* can be chosen to keep the level of the false negatives. If the developers have a high demand for accuracy of the recognition of coincidental correctness, a relatively high value of *p* can be chosen to keep the level of false positives.

3.2.3 Handling with the coincidental correctness

Passed test cases which are grouped into the same cluster with the failed ones are very likely to be coincidentally correct, and are added to *Ticc*. The reasons are two-fold:

1) A test case which executes the faulty statement does not necessarily induce a failure, but not vice versa. It is a sufficient condition for a failed test case to execute the faulty statements.

2) It is assumed that test cases with similar execution profiles will be clustered together. Therefore, the identified passed test cases will have similar execution profiles with the failed ones.

As such, the identified test cases have a great chance to execute the faulty elements, but still produce the correct output. In other words, they conform to the definition of coincidental correctness.

To deal with *Ticc*, we propose two strategies:

- The Cleaning Strategy: Test cases in *Ticc* are removed from the original test suite *T*. According to the suspiciousness metric, it will improve the suspiciousness values of the faulty statements by subtracting the number of coincidental correctness from the number of passed test cases.
- The Relabeling Strategy: The labels of test cases in *Ticc* are changed from "*passed*" to "*failed*". It will also improve the suspiciousness values of the faulty statements by subtracting the number of coincidental correctness from the passed test cases and adding it to the number of failed ones. The improvement may be more significant than the cleaning strategy to a certain extent, but it may have risks.

3.2.4 Fault Localization

In this study, we select Ochiai (rather than Tarantula) as the CBFL technique. The main reason is that Ochiai is a recently proposed technique and the metrics is more effective than Tarantula in locating faults. Although these two techniques share the basic principle of CBFL, and they operate on exactly the same input data, as demonstrated in R. Abreu et al., [9], the Ochiai similarity coefficient can improve diagnostic accuracy over other coefficients, including those used by the Pinpoint and Tarantula tools [16]. As a result, it will be more convincing that if our approach can improve a well-performed CBFL technique.

Note that the objective of this study is not to compare various fault localizers, but rather to develop a strategy that will improve the CBFL across multiple fault localizers. Although existing CBFL techniques use different metrics for the coverage entities, most of them share the same basic principle to locate the faulty elements. In other words, they have the same hypothesis and share similar input data. Therefore, we believe that if our approach works on Ochiai, it will perform reasonably well on other fault localizers. In the future work, we will conduct experiments to investigate this conjecture.

4 EXPERIMENT AND EVALUATION

4.1 Subject Programs

The Siemens set is used as the subject programs in our study because it is a particularly widely used benchmark for evaluating software testing and fault localization techniques. The detailed information on these programs is listed in Table 1. The second column of the table shows for each program the number of faulty versions used in our experiment. The third column shows for each program the lines of code (the number of executable statements in the parentheses) that it contains.

TABLE 1. DETAILED INFORMATION OF THE SUBJECT PROGRAMS

Program	Versions	LOC (Executable)	Test Suite Size
replace	29	563(243)	5542
printtokens	3	563(190)	4130
printtokens2	6	510(200)	4115
schedule	8	412(150)	2650
schedule2	4	307(127)	2710
totinfo	16	406(123)	1052

The program *tcas* is not included because it is too small for cluster analysis. It has only 173 lines of code, of which 54 are executable statements. Therefore, many test cases may have same execution profiles. Consequently, the number of the clusters generated is limited, and it is difficult to effectively distinguish test cases in this case.

Additionally, we also exclude some of the remaining versions for the following reasons: these versions have no failures detected by any test case. Besides, similar to the experimental setup of Jones and Harrold [1], executable statements are used instead of LOC. Thus we ignore the versions with modifications in the header files, mutants in variable declaration statements, or modifications in a macro statement started with "#define". Furthermore, versions contain code-missing errors are also excluded. Because CBFL hinges on the assumption that a statement which has been executed by most failed test cases is a good candidate for being faulty, and in other words, if a faulty statement causes a test case to fail, then the test case must have executed that statement. However, this will not hold in certain cases such as code-missing fault. In this situation, there is no so-called faulty statement and CBFL cannot localize the fault exactly [14]. Finally, some versions are omitted because they do not contain any coincidental correctness. In summary, we have excluded 25 faulty versions in total, and use 66 versions for our experiment.

4.2 Evaluation Metrics

Similar to [1], to evaluate the ability of our approach to identify coincidental correctness, we compute metrics to quantify the generated false negatives and false positives. Also, to assess the impact of our approach on the effectiveness of CBFL, we use the T-score reduction as the evaluation metric.

1) Measure of generated false negatives:

$$\frac{|Tcc - Ticc|}{Tcc} \qquad (1)$$

This measure assesses whether we have successfully identified all the coincidentally correct test cases. The lower the measure value is, the better the recognition accuracy is.

2) Measure of generated false positives:

$$\frac{|(Tp - Tcc) \cap Ticc|}{Tp - Tcc} \qquad (2)$$

This measure assesses whether we have mistakenly categorized test cases as coincidentally correct. Similarly, the lower the measure value is, the better the recognition accuracy is.

3) Measure of effectiveness improvement:

T-score reduction $\triangle TS = TS - TS'$, where TS and TS' represents for the T-score before and after applying our approach respectively.

T-score is widely used in evaluating fault localization techniques [1], [10], [11]. It measures the percentage of code that has been examined in order to find the defect, and is defined as follow:

$$\text{T-score} = \frac{|V_{examined}|}{|V|} * 100\% \qquad (3)$$

$|V|$ refers to the size of the program (lines of the executable statements), and $|V_{examined}|$ refers to the number of statements investigated by the programmer in order to find the defect. The lower the T-score value is, the more effective the method will be. Therefore, a larger $\triangle TS$ implies a greater improvement.

4.3 Experimental Results

4.3.1 Recognition Accuracy

Table 2 shows the ability of our approach to recognize the coincidental correctness. It takes p (the ratio of the number of clusters) as a parameter, and p = 6%. This value is selected according to previous studies [6], [8]. The column named "Range" represents the percentages of false negatives and false positives, and the column of "Versions" represents the percentage of versions that exhibit a given range. "FN" and "FP" are short for "False Negative" and "False Positive" respectively.

From Table 2, the following observations can be made about the recognition accuracy of our approach:

1) 2% versions can recognize more than 90% coincidentally correct test cases

2) 21% versions generate 10%-50% false negatives

3) 33% versions generate 50%-90% false negatives

4) 44% versions fail to recognize most of the coincidentally correct tests

5) Most of the versions, 92%, specifically, generate a small number of false positives, in the range [0%, 10%]

It can be speculated that larger number of clusters yields a higher rate of false negatives but a lower rate of false positives. It is reasonable because the purity of a cluster increases as the number of clusters increases. As a result, some of the coincidentally correct test cases, once put into a cluster with some failed ones, are spread to another cluster full of passed test cases. Therefore, these coincidentally correct test cases will be missed. Similarly, some non-coincidentally correct test cases will be spread to another cluster full of passed test cases so that these test cases will not be mistaken for coincidental correctness.

TABLE 2. RECOGNITION ACCURACY

Range	Versions%(FN)	Versions%(FP)
0%~10%	1.51	92.42
10%~20%	1.51	3.03
20%~30%	7.57	4.54
30%~40%	4.54	0
40%~50%	7.57	0
50%~60%	3.03	0
60%~70%	6.06	0
70%~80%	9.09	0
80%~90%	15.15	0
90%~100%	43.93	0

4.3.2 *Impact on the Effectiveness of Fault Localization*

We use box plots to depict the overall experimental results. It takes *p* (the ratio of the number of clusters) as a parameter, and *p* = 6%. Each box plot represents the statistics of the T-score reduction for each subject program. The bottom and top of the box are 25th and 75th percentile, respectively. The line within the box denotes the median of the values in the box and the point denotes the mean. The ends of the whiskers represent the minimum and maximum of all the data.

Figure 2 and 3 illustrates the impact of our approach on the effectiveness of CBFL. Figure 2 depicts the results applying the cleaning strategy, and Figure 3 depicts the results applying the relabeling strategy. The ideal situation, where all the coincidentally correct test cases are picked out, with 0% false negatives and 0% false positives, is also shown on the figures as a comparison. The bold dot presents the average T-score reduction for each program under the ideal situation.

19.57% versions have been improved by the cleaning strategy, and the T-score reduction can reach up to 8.55%. 33.33% versions have been improved by the relabeling strategy, and the T-score reduction can reach up to 22.0%.

In summary, as shown in Figure 2 and 3, the cleaning strategy is relatively a safe method, because if p is set to a reasonable value, 6% in our case, more than 87% versions will be improved or stay the same, with an increase rate of 0.67%~8.55%, while the rest 13% may be deteriorated, with the decrease rate of -2.0% ~ -0.41%.

The effectiveness of fault localization of some versions remains the same. We observe that the faulty statements in most of the versions have already been ranked at the very front position, such that dealing with the coincidental correctness will have little effect on the improvement. Moreover, as presented in W. Masri et al., [5], although cleaning coincidental correctness will lead to an increment of the suspiciousness metric of the faulty statement, the rank of the statement will not necessarily increase correspondingly. Therefore, the effectiveness of CBFL after applying our methodology remains the same or even gets worse for some versions. The key factors that influence the improvement of CBFL are the rate of false negatives and false positives, which are heavily depended on the clustering results.

Furthermore, we conducted a paired t-test on the differences between the T-scores before and after applying our strategy. The paired t-test is a statistical technique that is used to compare two population means in the case of two samples that are correlated, especially in a “before- after” study. Suppose the T-score before using our strategy is A, and the T- score after using our strategy is B. H0: A <= B, Ha: A > B. Note that, during this test, only T-scores less than 20% are taken into account. The reason is that it is not common to ask programmers to examine more than 20% of the code in practice [11]. It can observed that, by using cleaning strategy and relabeling strategy, the p-value is 0.01 and 0.02, respectively, both of which implies that the improvement is significant at the 0.05 level.

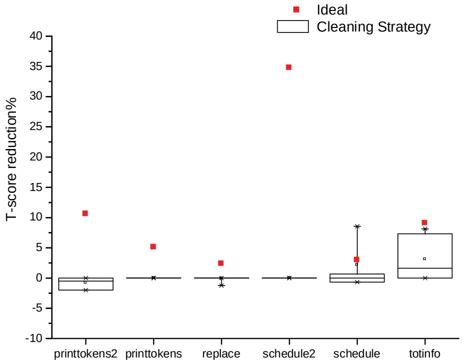


Figure 2. Impact of cleaning strategy on the effectiveness of CBFL

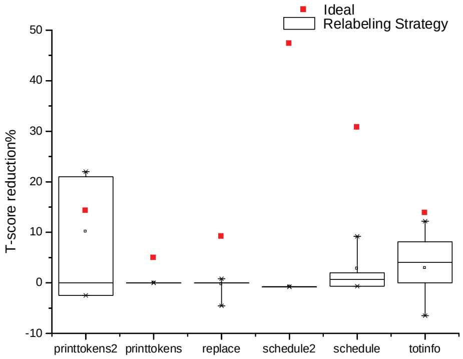


Figure 3. Impact of relabeling strategy on the effectiveness of CBFL

Statistical test is also conducted on the results of the ideal situation. Under the ideal situation, using cleaning and relabeling strategy, the p-value is $3.43*10^{-4}$ and $9.71*10^{-17}$ respectively, and both of them indicate a very significant improvement.

The above result shows that it is promising to improve the effectiveness of CBFL by dealing with coincidentally correct test cases. And using our approach, the preliminary experimental result is encouraging and convincing.

4.4 *Threads to Validity*

The threats to external validity include the use of Siemens set as our subject programs. As a matter of fact, these programs are all small C programs and the faults are manually injected. To reduce this threat, we plan to apply our technique to larger programs in the future.

The threats to internal validity include the tools we use to generate execution profiles and conduct the cluster analysis. In our context, we use gcov to record coverage information and rely on the data mining tool Weka for cluster analysis. Both of them are mature and widely used. Another issue related to internal validity is the clustering algorithm we choose. As in our study, we use simple K-means because it is simple and effective. However, having the failed test cases clustered either too centralized or too scattered will have adverse effect on the results. To be more specific, they will lead to high false negative and false positive rates, respectively.

5 RELATED WORK

Voas [2] introduced the PIE model, which emphasizes that for a failure to be observed, the following three conditions must be satisfied: "*Execution*", "*Infection*", and "*Propagation*". W. Masri et al. [3] have proved that coincidental correctness is responsible for reducing the safety of CBFL.

As shown in the previous studies [4, 5], the efficiency and accuracy of CBFL can be improved by cleaning the coincidentally correct test cases. However, it is challenging to identify coincidental correctness because we do not know the location of fault beforehand. X. Wang et al. [4] have proposed the concept of context pattern to help coverage refinement so that the correlation between program failures and the coverage of faulty statements can be strengthened. W. Masri et al. [5] have presented variations of a technique that identify the subset of passed test cases that are likely to be coincidentally correct. One of these techniques first identifies program elements (cc_e) that are likely to be correlated with coincidentally correct test cases. Then it categorizes test cases that induce some $cc_e s$ as coincidental correctness. The set of coincidentally correct test cases would be partitioned into two clusters further. A more suspicious subset will be cleaned to improve the effectiveness of fault localization. The experimental result is promising, however, although it used the same subject program (the Siemens test suite) as ours in their experiment, it is applicable to only 18 versions of the 132 versions, which has a smaller application scope than our approach (applicable to 66 out of 132 versions).

Previous empirical observations have shown that, by cluster analysis, test cases with similar behaviors could be grouped into the same clusters. Therefore, cluster analysis has been introduced for test case selection. Vangala et al. [12] used program profiles and static execution to compare test cases and applied cluster analysis on them, identifying redundant test cases with high accuracy. Dickinson et al. introduced cluster filtering technique [7, 13]. It groups similar execution profiles into the same clusters and then selects a subset of test cases from each cluster based on a certain sampling strategy. Since test cases in the same cluster have similar behaviors, the subsets are representative for the test suite so that it is able to find most faults by using the selected subsets instead of the whole test suite.

6 CONCLUSIONS AND FUTURE WORK

In this paper, we proposed a clustering-based strategy to identify coincidental correctness from the set of passed test cases. To alleviate the adverse effect of coincidental correctness on the effectiveness of CBFL, two strategies, either removing or relabeling, were introduced to deal with the identified coincidentally correct test cases. We conducted an experiment to evaluate the proposed approach. The experimental results suggested that it achieved approximate results as the ideal situation did.

We intend to conduct more comprehensive empirical studies and explore the following issues in our future work:

1) Search for better clustering algorithms to fit in with this scenario. As denoted in section 4.4, the failed test cases clustered either too centralized or too scattered would lead to poor results. We use K-means in our experiment for its simplicity, and there are many other clustering algorithms need to be explored.

2) Conduct empirical studies on how multiple-faults affect the result of our approach and explore how to deal with this situation to minimize the adverse effects.

REFERENCES

[1] J.A. Jones and M. J. Harrold, "Empirical evaluation of the tarantula automatic fault-localization technique", ASE, 2005, pp.273-282.

[2] J.M. Voas, "PIE: A dynamic failure-based technique", IEEE Trans. Softw. Eng., 1992, pp.717-727.

[3] W. Masri, R. Abou-Assi, M. El-Ghali and N. Fatairi. Nour, "An empirical study of the factors that reduce the effectiveness of coverage-based fault localization", ISSTA, 2009, pp.1-5.

[4] X. Wang, S. Cheung, W. Chan and Z. Zhang, "Taming coincidental correctness: Coverage refinement with context patterns to improve fault localization", ICSE, 2009, pp.45-55.

[5] W. Masri and R. Abou-Assi, "Cleansing test suites from coincidental correctness to enhance fault-localization", ICST, 2010, pp.165-174.

[6] S. Yan, Z. Chen, Z. Zhao, C. Zhang and Y. Zhou, "A dynamic test cluster sampling strategy by leveraging execution spectra information", ICST, 2010, pp.147-154.

[7] W. Dickinson, D. Leon and A. Podgurski, "Finding failures by cluster analysis of execution profiles", ICSE, 2001, pp.339-348.

[8] C. Zhang, Z. Chen, Z. Zhao, S. Yan, J. Zhang and B. Xu, "An improved regression test selection technique by clustering execution profiles", QSIC, 2010, pp.171-179.

[9] R. Abreu, P. Zoeteweij, R. Golsteijn, A. J.C. van Gemund, "A practical evaluation of spectrum-based fault localization", Journal of Systems and Software, Volume 82, Issue 11, 2009, pp. 1780-1792.

[10] B. Liblit, M. Naik, A. Zheng, A. Aiken and M. Jordan, "Scalable statistical bug isolation", PLDI, 2005, pp.15-26.

[11] C. Liu, X. Yan, L. Fei, J. Han and S. Midkiff, "SOBER: statistical model-based bug localization", ESEC/FSE, 2005, pp.286-295.

[12] V. Vangala, J. Czerwonka and P. Talluri, "Test case comparison and clustering using program profiles and static execution", ESEC/FSE, 2009, pp. 293-294.

[13] W. Dickinson, D. Leon and A. Podgurski, "Pursuing failure: the distribution of program failures in a profile space", ESEC/FSE, 2001, pp. 246-255.

[14] V. Debroy, W. Eric Wong, X. Xu, B. Choi, "A grouping-based strategy to improve the effectiveness of fault localization techniques", QSIC,2010, pp.13-22.

[15] T. Denmat, M. Ducassé and O. Ridoux, "Data mining and cross-checking of execution traces: a re-interpretation of Jones, Harrold and Stasko test information", ASE, 2005, pp. 396-399.

[16] L. Naish, H. J. Lee, and K. Ramamohanarao, A model for spectra-based software diagnosis, TOSEM, in press.

[17] Software-artifact Infrastructure Repository. http://sir.unl.edu/, University of Nebraska.

[18] http://gcc.gnu.org/onlinedocs/gcc/Gcov.html

[19] http://en.wikipedia.org/wiki/Euclidean_distance

Regression Testing Prioritization Based on Fuzzy Inference Systems

Pedro Santos Neto - UFPI - Teresina - PI - Brazil
pasn@ufpi.edu.br

Ricardo Britto - UFPI - Teresina - PI - Brazil
pasn@ufpi.edu.br

Thiago Soares - UFPI - Teresina - PI - Brazil
thiagoacs2@gmail.com

Werney Ayala - UFPI - Teresina - PI - Brazil
werney@gmail.com

Jonathas Cruz - UFPI - Teresina - PI - Brazil
jonathas jivago@hotmail.com

Ricardo Rabelo - USP - S˜ao Paulo - SP - Brazil
ricardor@sc.usp.br

***Abstract*—The software testing is a fundamental activity related to product quality. However, it is not performed in suitable way by many organizations. It is necessary to execute testing in a systematic and planned way. This work presents a *fuzzy* inference system for test case prioritization, based on the use of inputs related to volatility, complexity and relevance of requirements. The developed inference system allows the specification of a prioritization strategy by the tester based on linguistic rules in a simple and easy way.**

I. Introduction

Testing is the final verification of a software product [1]. Despite of the relevance of this phase on a project, testing is not systematic fulfilled for the enterprises. That situation happens mainly because of the costs and the short schedules related to the software projects.

The software engineering community must find ways to facilitate the inclusion of systematic testing into software processes. In some enterprise environments, which use software testing to assure the software quality, might be necessary prioritize existing test cases because often there is not enough time or resources to execute all planned test cases. In this case, it is desirable to prioritize the test cases, in order to maintain the most important ones in the first positions, assuring their execution. To do that, it is necessary create a execution order that optimizes the most important features for the testing in a certain time, such as code covering.

Since creating the test execution order is a np-hard problem [2], it is not possible to solve that problem in an efficient way through standard techniques [3]. Thus, many researchers have applied computational intelligence techniques to solve the test case prioritization problem ([4], [5], [6]).

This work presents an approach based on Mamdani fuzzy inference system [7] that uses requirements to solve the test case prioritization problem ([8], [5], [9]). The approach allows that a test engineer can insert his knowledge about the prioritization process into the fuzzy inference system. Therefore, the fuzzy inference system mimics the behavior of the expert.

The main contribution of this work is the use of a fuzzy inference system to qualify each existing test case of a software project. This feature allows the prioritization through a simple sort algorithm, driven by the value inferred for each test case, based on common input data as relevance of the requirement for the project, its complexity and volatility. The complexity and volatility can be easily obtained from common software metric tools, like Sonar[1].

This work is organized as following: in Section 2 is presented a software testing primer; in Section 3 a fuzzy inference system is described; in Section 4 the proposed approach is detailed; in Section 5 an empirical evaluation is presented; in Section 6 some related works are discussed; finally, in Section 7 the conclusions and future works are presented.

II. Software Testing Primer

Software Testing is one of the main activities of quality assurance in software development. Software testing consists of the dynamic verification of the behavior of a program on a finite set of test cases, suitably selected from the usually infinite executions domain, against the expected behavior.

There are different classifications related to software testing. Tests can be classified by the target: one module (unit), several modules grouped (integration) and all the modules together (system).

Testing can be aimed at verifying different properties. This represents the test classification by objectives. Test cases can be designed to check that the functional specifications are correctly implemented (functional testing). Other important objectives for testing include (but are not limited to) reliability measurement, usability evaluation, performance, stress and acceptance, for which different approaches would be taken. Note that the test objective varies with the test target; in general, different purposes being addressed at a different level of testing.

[1] http://www.sonarsource.org/

According to IEEE [10], regression testing is the selective retesting of a system or component to verify that modifications have not caused unintended effects. In practice, the idea is to show that software did not regress, that is the software which previously passed the tests still does. The repetition of tests is intended to show that the softwares behavior is unchanged, except insofar as required. Obviously a trade-off must be made between the assurance given by regression testing every time a change is made and the resources required to do that. A important limiting factor applied to regression testing is related to time-constraints. Sometimes, the total execution of testing can take many hours. This makes unfeasible the test execution constantly. Due to this factor it is required to prioritize the test cases, in order to create the best sequence that meets the time constraints.

It is relevant to emphasize that the selection must be found quickly and the process to find this must not require the tests execution, since the objective is right to reduce the test execution.

Unfortunately, several approaches developed so far, mainly related to the use of search based techniques applied to this problem, are only theoretical proposals, that can not be executed in a real software development environment.

It is mandatory the use of information that can be easily obtained by automatic tools. Therefore, it is mandatory the use of algorithms that do not require the tests execution to find the best order. The sequence must be inferred from test properties.

III. Fuzzy Inference Systems

Fuzzy inference systems are based on linguistic production rules like ”if ... then”. The fuzzy sets theory [11] and fuzzy logic [12] provide the math base required to deal with complex processes, based on inaccurate, uncertain and qualitative information.

Fuzzy inference systems have their operation based on three steps: i) fuzzification, ii) inference procedures, and iii) defuzzification. The fuzzification is a mapping of the numerical inputs to fuzzy sets. Those sets are represented by linguistic terms, such as ”very”, ”little”, ”medium”, etc. The fuzzy inference procedure is responsible to infer output fuzzified value based on the input values. Defuzzification is applied to associate a numerical value to a fuzzy output, obtained from the fuzzy inference procedure.

In this work, we applied a Mamdani fuzzy inference system model. That system model was applied because it computationally simulate the human ability to take rational decisions in an environment with inaccuracies, uncertainties and noise [7]. Production rules in a Mamdani inference model have fuzzy sets in their antecedents and consequents. Therefore, a rule base in a Mamdani model can be defined exclusively in a linguistic way, without the need of numerical data.

A fuzzy inference systems can express and handle qualitative information. This allows the mapping of the experience of domain specialists, facilitating the decision making process. Thus, using a Mamdani fuzzy inference system, to solve the test case prioritization problem, allows to get an action strategy / control that can be monitored and interpreted from the linguistic point of view. Thus, the action / control strategy of the Mamdani fuzzy inference system can be considered as reasoned and consistent as the strategy of domain experts.

IV. Proposed Approach

Our approach evaluates the requirements to infer the test case criticality for regression testing purpose. However, test cases from real systems can eventually cover artifacts that may be related to more than one requirement. In the current stage of our work we do not take into account this behavior.

Each requirement is evaluated based on 3 variables that represent the volatility, complexity and relevance. By using these variables a value that represents the criticality of a test case is generated. We projected a Mamdani fuzzy inference system to infer the test case criticality, as shown in Figure 1.

The crisp input variables of the fuzzy inference system are Volatility, Complexity and Relevance. The meaning of each variable is described as following:

- Volatility (V): value representing the amount of versions of the artifacts related to a requirement. The amount of versions can be easily obtained from a control version systems like GIT[2] and SVN[3].
- Complexity (C): value representing the ciclomatic complexity of the artifacts related to a requirement. This can be easily obtained from a common software metrics tools, like Sonar.
- Relevance (R): value representing the relevance of the requirement for the customer.

The values related to volatility and complexity are calculated from the data obtained for each artifact associated to a requirement. Thus, the volatility for a specific requirement is obtained from the volatility of the whole classes linked to the requirement implementation. To do this, we use the traceability matrix [1] to obtain the forward traceability information.

The 3 input variables can receive real values between 0 and 5. The output variable of the fuzzy inference system is **C**riticality of the **T**est **C**ase (CTC). The bigger the CTC, the bigger the test case execution priority. Thus, the test case with the biggest inferred CTC is the first to be executed.

The CTC output variable can receive real values between 0 and 10, representing the real values of that variable.

To map the fuzzy sets, we used a uniform distribution of the sets. This is the normal process to do this task. After done the uniform distribution of the fuzzy sets, it is possible to improve the mapping empirically or using computational intelligence techniques such genetic algorithms [13].

We used triangular and trapezoidal membership functions to empirically map fuzzy sets to the input variables. To each input variable 3 fuzzy sets were mapped: Low (L), Medium (M) and High (H). We also used triangular membership functions to empirically map fuzzy sets to the output variable. We mapped

[2] http://git-scm.com/

[3] http://subversion.tigris.org/

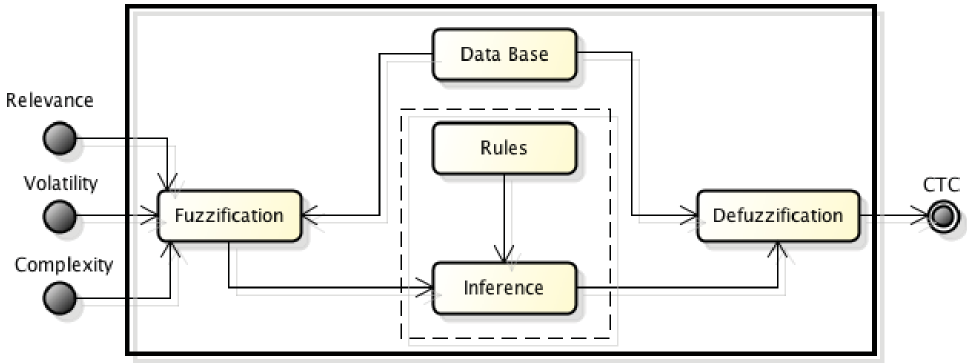


Fig. 1. Proposed Fuzzy Inference System

5 fuzzy sets to CTC: Very Low (VL), Low (L), Medium (M), High (H) and Very High (VH). We can see in details the distribution of the fuzzy sets in Figure 2.

To generate CTC of each test case through the 3 specified input variables, we must set the rule base in the proposed fuzzy inference system. The implemented rule base, which is the kernel of the CTC estimation strategy, has 27 rules. The rule base is presented in a matrix way, as seen in Table I. In this Table, R means Relevance, C means Complexity, V means Volatility, VL means very low, L means low, M means medium, H means high and VH means very high.

TABLE I
THE RULE BASE OF THE PROPOSED APPROACH

	R/C								
V	H/H	H/M	H/L	M/H	M/M	M/L	L/H	L/M	L/L
H	VH	H	M	H	M	M	M	M	L
M	VH	H	M	M	M	M	M	L	VL
L	H	M	M	M	M	L	M	L	VL

The rule base and the fuzzy sets associated to the input and output variables were defined using the authors' knowledge related to software testing, but the rule base was improved with the expert knowledge as discussed in the following section. The proposed approach allows that a test engineer easily changes the test case prioritization strategy. To do that, the expert must only change the rule base of the fuzzy system.

In this work, we implemented the centroid defuzzification technique [14], formalized by Equation 1. In this equation, $x*$ is the defuzzified output representing the CTC value, $\mu_i(x)$ is the aggregated membership function and x is the output variable. It is important to notice that the implemented inference system allows user to change the defuzzification method in a simplified way.

$$x* = \frac{\int \mu_i(x)\,x\,dx}{\int \mu_i(x)\,dx} \qquad (1)$$

V. Empirical Evaluation

In order to evaluate the proposed approach, we have carried out an empiric evaluation. We had 2 objectives to be reached with the empiric evaluation:

1) Show the ability to mimic the expert knowledge. It means that our fuzzy inference system should prioritize the test cases of an under development system in a similar way that the expert who adjusted the rule base and the membership functions of the fuzzy inference system;
2) Show the ability to generalize the knowledge of the expert. It means that the system should prioritize different systems in a similar way.

In order to reach the above mentioned objectives, we applied our approach to prioritize test cases of 2 real information systems. The evaluated information systems are described as follows:

- **JBook**: a book loan system to control the library of a real software house.
- **ReqG**: a requirement management tool created to control the requirements and use cases of a project.

The systems were analyzed with the help of a test expert that have participated in the development of both systems. The tester is member of a software development company at Brazil. He has 2 years software development experience.

The empirical evaluation was conducted in the following order: i) the tester indicated the values for Relevance (R), Complexity (C) and Volatility (V) for each requirement related to ReqG system; ii) the tester indicated the prioritization order for ReqG tests in his point of view, using only his feeling about the features; iii) we calibrated a rule base of our fuzzy inference system with the tester information; iv) the tester indicated the values (R, C, V) for JBook system; v) the tester indicated the prioritization order for JBook tests in his point of view, using only his feeling about the features; vi) we generated the prioritization order for JBook using the proposed approach; vii) we compared the results.

In order to calibrate the fuzzy rule base we have used linear regression [15]. This is an approach to model the relationship between a scalar variable and one or more explanatory variables. The rule base were calibrated from a estimation based on the coefficients R, C and V, suggested from the analysis of the prioritization indicated by the tester. Running a linear regression we can easily obtain the coefficients of the equation that models the behavior. The bigger the coefficient of a variable, the bigger its influence in the result. Thus, we have to reflect this in the rule base. The variable with bigger influence must be more important in the result determination of CTC.

In order to facilitate the presentation of the results, we grouped the results in two tables. Table II shows the test case prioritization inferred by our approach for ReqG system. Table III shows the test case prioritization inferred by our approach for JBook system.

The tables have the same columns. Column R shows the relevance of each requirement of the system. The column C shows the complexity of each requirement. The column V shows the volatility of each requirement. The column CTC shows the criticality of the test case which cover the respectively requirement. Finally, the column Tester shows the prioritization indicated by our expert.

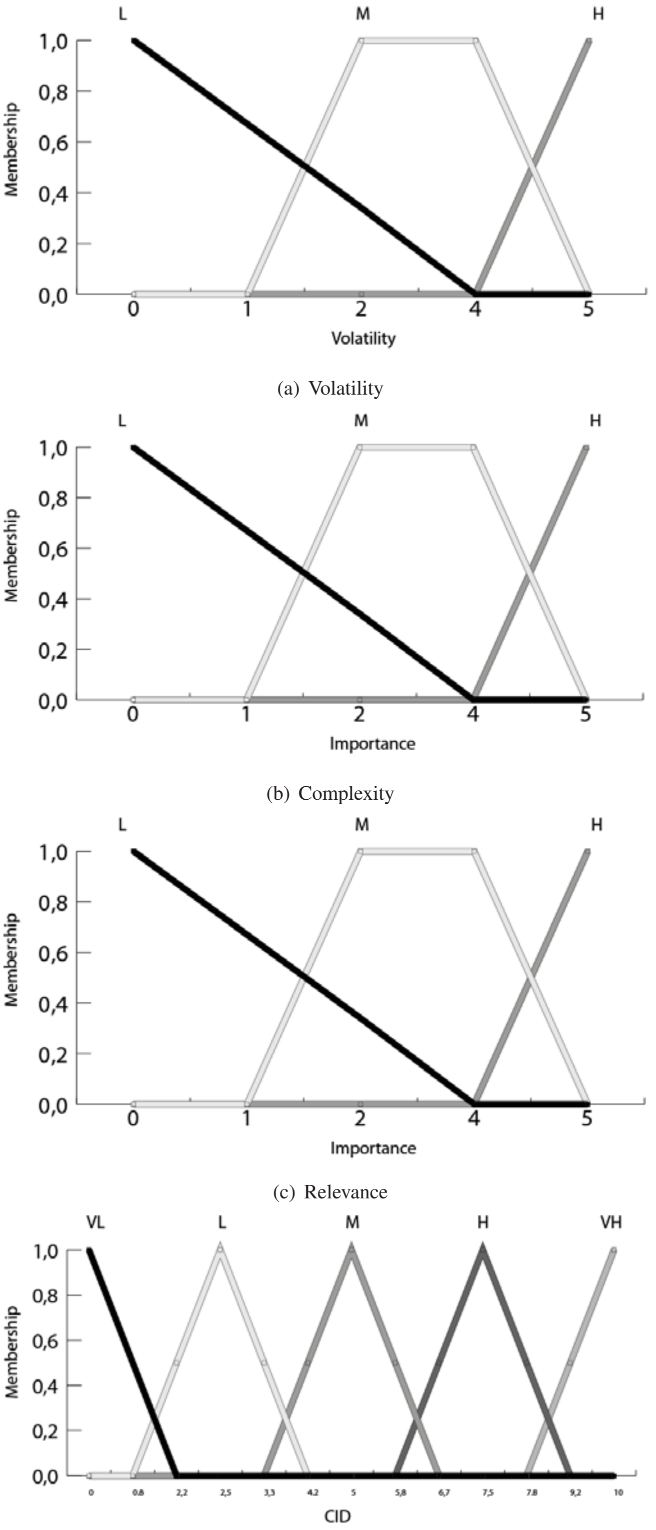
The analysis of the results shows some interesting data:

- The results from the Tester and our approach, related to ReqG system, was exactly the same. This was possible due to our calibration of the rule base made by using the tester feeling. So, it was the expected result.
- The results from the Tester and our approach, related to JBook system, was almost the same. There was only two mistakes among 13 possibilities. This indicates that our approach has mimicked the Tester behavior.
- Once in both cases our proposed fuzzy inference system mimicked the Tester behavior, we also reached the other objective, that was generalize the knowledge of the expert.

The results described above validates our approach, since it is possible to capture a Tester feeling and use it for prioritization purposes. The data required to execute our approach is simple and could be obtained from automatic tools. Our approach does not requires the test execution to prioritize the test cases. This is a relevant behavior since the system source code changes all the time and it is necessary to prioritize test to be executed to assure the system behavior.

TABLE II
PRIORITIZATION SEQUENCE FOR REQG INFERRED BY THE FUZZY INFERENCE SYSTEM

Requirements	R	C	V	CTC	Tester
Project	5	3	3	7,15	1
Review	3	5	5	7,15	2
Requirement	5	2	2	6,76	3
Use Case	5	3	2	6,76	4
Member	3	2	4	4,48	5
Change History	2	4	2	4,26	6
Actor	4	2	2	4,26	7
Review Criteria	3	1	2	3,86	8



(a) Volatility

(b) Complexity

(c) Relevance

(d) CTC

Fig. 2. Distribution of the fuzzy sets

TABLE III
PRIORITIZATION SEQUENCE FOR JBOOK INFERRED BY THE FUZZY INFERENCE SYSTEM

Requirements	R	C	V	CTC	Tester
Loan Request	5	3	2	6,76	1
Loan Report	5	3	2	6,76	2
Publication Register	5	1	3	5	3
Loan Confirmation	5	1	1	5	4
Loan Finalization	5	2	1	5	5
Login	5	1	1	5	6
Book Registration	4	1	2	4,16	7
Loan Cancellation	4	2	1	4,16	8
Loan Cancellation Request	3	2	1	3,86	9
Label Tipping Report	2	1	2	3,41	13
Password Change	3	1	1	2,33	11
User Registration	3	1	1	2,33	12
Email Notification	2	1	1	2,12	10

VI. RELATED WORKS

The work developed by Kavitha et al [8] presents an approach to prioritize test cases based on requirements. The approach uses three factors to prioritize test cases: the requirement priority in the client view, the implementation complexity and the volatility of these requirements. For each requirement a factor named RFV (Requirement Factor Value) is calculated, representing the average of the considered factors. The weight of a test case is obtained through a formula that takes into account the RFV of all the requirements covered by the test case. The bigger the weight of a test case, the bigger its relevance. Our work also uses three factors to calculate the criticality of a test case, but we use a fuzzy system to mimic an expert tester.

The work of Srikanth et al [9] proposed a technique for test case prioritization named PORT (Prioritization of Requirements for Test). PORT prioritizes system test cases based upon four factors: requirements volatility, customer priority, implementation complexity, and fault proneness of the requirements. The work presents a case study on four projects developed by students. The results show that PORT prioritization at the system level improves the rate of detection of the faults considered in the study. The method to guide the fault injection is not presented in the study. The main difference from our work is the absence of an expert evaluation in order to generate a guide for prioritization, since this could present a bias depending on the faults used for evaluation.

Yoo and Harman [4] introduced the concept of Pareto multiobjective optimization to the problem of test case selection. They describe the benefits of Pareto multiobjective optimization, and present an empirical study that investigated the effectiveness of three algorithms for Pareto multiobjective test case selection. However, the technique is not suitable for a real scenario, since it requires several test executions to find the best selection. Test cases change all the time, being hard to find a static order that is not based on properties easily obtained in an automatic way.

Maia et al [5] proposed an alternative solution to order test cases. Such technique uses Reactive GRASP metaheuristic to prioritize test cases. They compare this metaheuristic with other search-based algorithms previously described in literature. Five programs were used in the experiments. This work has the same weakness of the previous work: the need to execute all the tests to find the best order.

As mentioned before, our work is based on a fuzzy inference system that uses common input data easily generated by automatic metric tools. The approach uses the expert knowledge to prioritize test cases and does not require the test execution to generate the suitable order. It qualifies each test case individually in order to infer its criticality. The bigger the criticality, the bigger the test case execution priority. Thus, the test case with the biggest inferred criticality must the first to be executed.

VII. CONCLUSION AND FUTURE WORKS

This paper presents a Mamdani fuzzy inference system approach to solve regression testing prioritization problem on software projects. The approach presented differs from related works cited in Section VI because the proposed fuzzy inference system qualifies each test case individually, calculating the **C**riticality of the **T**est **C**ase (CTC). Using the CTC, the test engineer can sort the test cases using a standard sort algorithm. In the approaches present in the related works are created a completely test case sequence. Any necessary change in the sequence requires a recalculation.

Our empirical evaluation, although simple, validates our approach, since it have shown that is possible to capture a Tester knowledge and use it for prioritization purposes. The input data required to execute our approach is simple and could be obtained from automatic tools like SVN and Sonar. Besides, our approach does not require the test execution to prioritize the test cases, like other approaches described in the Related Work Section.

Regression testing must be executed all the time. Any approach created to solve regression testing prioritization problem must be fast and simple to be used on real scenarios. Our approach meets this requirement.

At the time that this is a work in progress, some issues in the current stage of this work do not assure the optimality of the inferred results:

- The experiment have used only one Tester. It is necessary to use a test team to generalize our results;
- It is necessary to search for alternatives to calibrate membership functions and the rule base of the proposed fuzzy inference system. We have used a linear regression in this work, but further work is needed;
- The approach must be used in a real scenario. It is necessary to create a tool able to be used by a team during a software development project.

Despite the issues summarized above, we can conclude, in a preliminary way, that the proposed approach is much more intuitive and suitable to be used by test engineers than other kinds of approaches based only in metaheuristics. It is possible due to the fuzzy inference system rule base construction.

It mimics the test engineer knowledge simpler than other approaches heavily based on mathematical formulations.

We are developing some improvements for this work:

- a stable version of a tool that incorporates our approach and can be connected directly with SVN and Sonar tools;
- a new version of the proposed fuzzy inference system that will use Computational Intelligence to optimize the previous specified membership functions and rule base as well;
- studies to specify a metric to measure, in a formal way, the Relevance variable. The metric will be probably based on SQFD (Software Quality Function Deployment) [16].

VIII. Acknowledgments

We would like to thank CNPq (grant 560.128/2010) and INES, for partially supporting this work. Also, we would like to acknowledge the Infoway company by the important contributions to the research.

References

[1] R. Pressman, *Software Engineering: A Practionerís Approach*. McGraw Hill, 2006.

[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. The MIT Press, 2009.

[3] M. Harman, "Search-based software engineering," *Information and Software Technology*, pp. 833–839, 2001.

[4] S. Yoo and M. Harman, "Pareto efficient multi-objective test case selection," in *Proceedings of the International Symposium on Software Testing and Analysis*, 2007.

[5] C. L. B. Maia, F. G. Freitas, and J. T. Souza, "Applying search-based techniques for requirements-based test case prioritization," in *Anais do I Workshop Brasileiro de Otimizacão em Engenharia de Software*, 2010.

[6] C. L. B. Maia, R. A. F. Carmo, F. G. Freitas, G. A. L. Campos, and J. T. Souza, "Automated test case prioritization with reactive grasp," *Advances in Software Engineering - Special Issue on Software Test Automation*, 2010.

[7] E. H. Mamdani, "Application of Fuzzy Logic to Approximate Reasoning Using Linguistic Synthesis," *IEEE Transactions on Computers*, vol. 26, no. 12, pp. 1182–1191, 1977.

[8] R. Kavitha, V. Kavitha, and N. Kumar, "Requirement based test case prioritization," in *Communication Control and Computing Technologies (ICCCCT), 2010 IEEE International Conference on*, 2010.

[9] H. Srikanth, L. Williams, and J. Osbome, "System test case prioritization of new and regression test cases," in *International Symposium on Empirical Software Engineering*, 2005.

[10] IEEE, "IEEE Standard Glossary of Software Engineering Terminology," New York, USA, Sep. 1990.

[11] L. Zadeh, "Fuzzy Sets*," *Information and control*, vol. 8, no. 3, pp. 338–353, 1965.

[12] L. A. Zadeh, "Fuzzy logic = computing with words," *IEEE Transactions on Fuzzy Systems*, vol. 4, no. 2, 1996.

[13] L. B. LEAL, M. V. S. LEMOS, R. HOLANDA FILHO, R. A. L. RABELO, and F. BORGES, "A hybrid approach based on genetic fuzzy systems for wireless sensor networks," in *IEEE Congress on Evolutionary Computation (CEC)*, june 2011, pp. 965 –972.

[14] T. J. Ross, *Fuzzy Logic with Engineering Applications*. Wiley, 2004.

[15] M. H. Kutner, C. J. Nachtsheim, and J. Neter, *Applied Linear Regression Models*, fourth international ed. McGraw-Hill/Irwin, Sep. 2004.

[16] S. Haag, M. K. Raja, and L. L. Schkade, "Quality function deployment usage in software development," *Commun. ACM*, vol. 39, pp. 41–49, January 1996.

Parallel Path Execution for Software Testing Over Automated Test Cloud

Liu Wei, Liu Xiaoqiang, Li Feng, Gu Yulong
College of Computer Science and Technology
Donghua University
Shanghai, China
liuxq@dhu.edu.cn

Cai Lizhi, Yang Genxing, Liu Zhenyu
Shanghai Key Laboratory of Computer Software Testing &Evaluating
Shanghai, China

Abstract—This paper develops AT-Cloud, a platform for software testing over cloud, which is constructed based on open software Eucalyptus, Autotest and Django framework. Furthermore, a parallel path execution method for software function testing is proposed. Test scene flow graph is used to describe the test function point and their relationships, then parallel test paths can be derived from the graph according to prime path coverage criterion. Test paths scheduling and test results integrating algorithms on AT-Cloud are discussed to improve utilization of computing resources and testing productivity. The experiment shows that the prototype is practical.

Keywords-cloud computing, software function test, prime path coverage, parallel path execution.

I. Introduction

Software testing is resource-hungry, time-consuming and labor-intensive, which takes 40%-70% of software development costs, and even more for mission-critical application. Despite massive investments in quality assurance, serious code defects are routinely discovered after software has been released. Along with the expansion of software scale, high test coverage becomes more difficult. Software test automation is a way to reduce time and labor costs but still limited by test environment resources.

Cloud computing sets up a scalable computing environment and makes software test automation over cloud available, which may not only provides testing services on the Web, but also lead to a high test coverage in shorter time through test parallelization. Some efforts has been done on providing software test services with cloud computing. A prototype of testing as a service over cloud (TaaS) is discussed, which develops and evaluates the scalability of the platform of computing time on test task scheduling and test task processing over the cloud [1]. A software testing environment using cloud computing technology for dependable parallel and distributed system is provided, which adopts a open source system Eucalyptus and QEMU to build the cloud and emulate hardware faults flexibly [2, 3]. There are also some software testing application environments in industry. For example, SOASTA is a system which harnesses the power of cloud computing to provide cloud testing on Web [4].

In this paper, a cloud-based testing service experiment platform AT-Cloud is set up, which aims at supporting high-parallel test automation. AT-Cloud is built based on open source software, a cloud computing system Eucalyptus [5] and a software testing framework Autotest [6]. This paper mainly addressed the following issues:

A. The architecture of AT-Cloud and how the module Test_Manager developed by us integrates the open software Eucalyptus and Autotest.

B. Methods of decomposing software test into parallel test paths based on scene flow graph according to prime path coverage criterion for function test.

C. Scheduling parallel test paths and integrating the test results returned from each parallel test path over AT-Cloud.

This paper is organized as follows. Section 2 describes the architecture of AT-Cloud. Section 3 discusses the prime path coverage criterion and proposed a method of parallel test paths finding. Section 4 introduces the way of scheduling parallel test path, integrating and visualizing the test results. Section 5 gives the parallel path test experiments in AT-Cloud. Finally, we conclude the study in Section 6.

II. Architecture of At-Cloud

AT-Cloud runs as a Web service and can be accessed through Internet. Figure 1 shows the architecture of AT-Cloud.

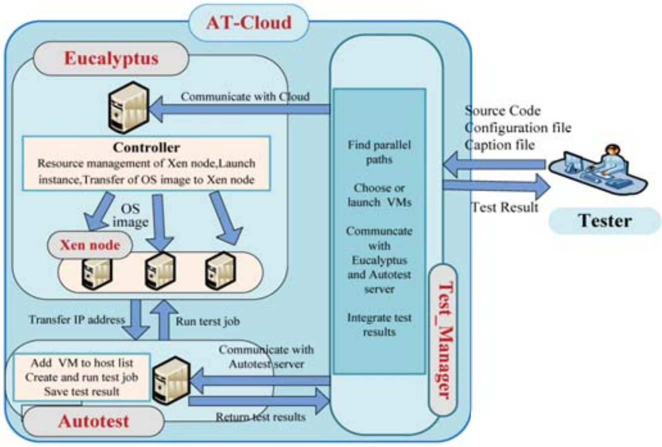


Figure 1 Architecture of AT-Cloud

AT-Cloud consists of the following components:

- Test_Manager, which communicates with Autotest Server and Eucalyptus, schedules parallel test paths and

This study is supported by Innovation Program of Shanghai Municipal Education Commission (12ZZ060) and Shanghai Foundation for Shanghai Key Laboratory (09DZ2272600).

integrate the test results.

- Autotest, which is a framework for automated testing.
- Eucalyptus, which manages Xen nodes using cloud controller, node controller and so on.
- Xen nodes, which makes virtual machine available by virtualization in cloud infrastructure.

A. *Test_Manager*

Test_Manager is a module developed using Web framework Django. It is the control center of AT-Cloud, which manages the communication among the Web, Eucalyptus and Autotest. The functions of Test_Manager are as follows:

1) Receives the test task, configuration file and test scene flow graph from tester. The test task includes test source code, test scripts based on test scene flow graph. The configuration file describes the hardware and software environment requirement of a test task.

2) Analyzes test scene flow graph and finds parallel paths according to the prime path coverage criterion.

3) Transfers emi-id, unique ID of the registered machine software image on the cloud, to the cloud controller.

4) Registers virtual machines to Autotest Server as clients and schedules parallel test paths to run on the registered VMs.

5) Collects test results produced by each test path from Autotest and integrates them together.

B. *Autotest*

Autotest is an open test framework for implementing automated test. The test procedures are shown as follows:

1) Autotest Server receives test source code, test scripts from Test_Manager.

2) Autotest Server receives VM IP address from cloud and adds the VM IP to Autotest Server's host list as an Autotest client.

3) Scheduled by Test_Manager, Autotest Server creates test job and run test job on Autotest client.

4) Autotest Server saves test results for every test.

C. *Eucalyptus*

AT-Cloud deploys virtual machine resource using Eucalyptus, which comprised five components: Cloud Controller, Walrus, Cluster Controller, Storage Controller, and Node Controller. The procedures of providing virtual machine resources VMs are as follows:

1) The cloud controller receives emi-id from Test_Manager.

2) According to the emi-id, Eucalyptus chooses or launches an instance of VMs.

3) After creating an VM instance, Eucalyptus assigns the instance an IP address, and this IP address will be transmitted to Autotest Server by Test_Manager.

D. *Xen*

AT-Cloud uses Xen as virtualization software. Xen is a virtual-machine monitor providing services that allow multiple computer operating systems to execute on the same computer hardware concurrently [7].

III. FINDING PARALLEL TEST PAHTS

Path test is a kind of structure test based on program source code in white-box test, which derives test path from program graph [8]. In this paper, path test is introduced for software function testing, modeling the test scene flow based on prime path coverage criterion.

A. *Test Scene Flow Graph and Test Path*

By decomposing software functions based on the scenes of a unit test, we can design the test cases according to the test equivalence class, and use a test scene flow graph to describe the function point and their relationships, which may have a number of test paths.

B. *Prime Path Coverage Criterion*

Prime path coverage criterion has the practical advantage of finding the parallel paths and improving the test coverage. In prime path coverage, tests should tour each prime path in a test scene flow graph. Prime path coverage requires touring all subpaths of length 0 (all nodes), of length 1 (all edges), length 2, 3, etc. Thus it subsumes node coverage, edge coverage and edge-pair coverage, which have a high path coverage rate [9]. The following are definitions about prime path coverage criterion:

1) Simple path: n_i and n_j are nodes in test scene flow graph G. A path form n_i to n_j is a simple path if no node appears more than once in the path, with the exception that the first and last nodes may be identical.

2) Prime path: A path from n_i to n_j is a prime path if it is a simple path and it does not appear as a proper subpath of any other simple path.

3) Prime path coverage (PPC) criterion: Test requirement contains each prime path in graph G.

C. *Finding Parallel Test Paths*

According to PPC criterion, after finding all prime paths in a graph, the parallel test paths can be obtained as follows.

```
1. Set up a test scene flow graph G
2. For path-length from 0 to the longest simple path in G,
find all possible prime path
      If the simple path's final node in G has no outgoing
           edges or the simple path's node is already in the
           path and not the first node
           Add the simple path to path-list
      Else
           Extending the simple path with every node that
           can be reached from the final node
3. If one path in the path-list is a proper sub path of another
   path in the path-list
      Remove the path from the path-list, finally all the
         prime paths will be found
4. For each prime path in the path-list
      Starting with the longest prime paths and extends
         them to the beginning and end nodes in the graph
```

D. An Example for Parallel Test Paths

Figure 2 shows the test scene flow graph of software about automated teller machine (ATM).

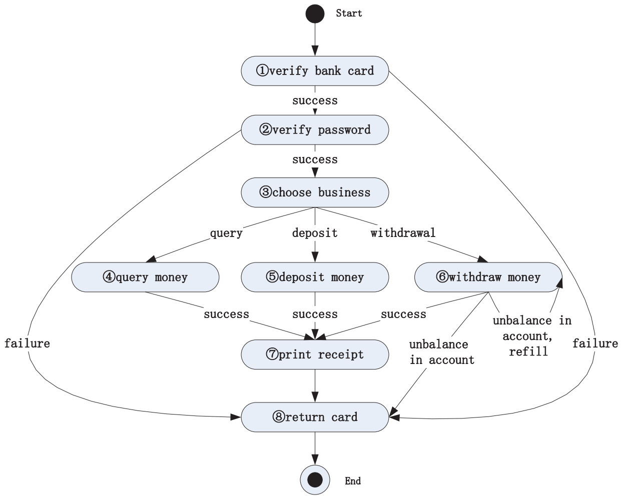


Figure 2 An example of test scene flow graph

It has eight nodes and twelve edges except the start node and end node. Seven prime paths can be found:

1) [1, 8]
2) [1, 2, 8]
3) [1, 2, 3, 4, 7, 8]
4) [1, 2, 3, 5, 7, 8]
5) [1, 2, 3, 6, 7, 8]
6) [1, 2, 3, 6, 8]
7) [6, 6]

Then we get the set of test paths:

1) [1, 8]
2) [1, 2, 8]
3) [1, 2, 3, 4, 7, 8]
4) [1, 2, 3, 5, 7, 8]
5) [1, 2, 3, 6, 7, 8]
6) [1, 2, 3, 6, 8]
7) [1, 2, 3, 6, 6, 7, 8]

For each test scene, we can design the test cases from different test equivalence class and set up a symbol to indicate the next scene as TABLE I showed, which contains SceneId, a serial number assigned to a test scene; CaseId, a serial number assigned to a test case; Prerequisites describes the values of the conditions and environment variable before the test is performed; TestData are input data; ExpectedResult are outputs or activities; NextScene indicates the following scene. Test_Manager store the test case in XML document as Figure 3 and analyze the XML document to generate test scripts.

TABLE I. TEST SCENE AND TEST CASES

SceneId	CaseId	Prerequisites	TestData	ExpectedResult	NextScene
1	1	100 exits in User.CardId	CardId =100	Output="Input Password"	2
1	2	101 not exits in User.CardId	CardId=101	Output= "Card error"	8
2	1	CardId=100 PS=123	PS=123		3
2	2	CardId=100 PS<>123	PS=123	Output="Password Error, again"	8
…	…	…	…	…	…

```
<testcase>
    <scene1>
        <case1>
            <Prerequisites> 100 exits in User.CardId
            </ Prerequisites >
            <TestData>CardID=100</TestData>
            <ExpectedResult>Output="Input Password"
            </ ExpectedResult >
           <NextScene>scene2</NextScene>
       </case1>
       ……
    </scene1>
    </scene2>
    ……
    <scene2>
      ……
</testcase>
```

Figure 3 Test cases in XML

IV. SCHEDULING PARALLEL TEST PATHS AND INTEGRATING TEST RESULTS

The test paths can be dispatched on AT-Cloud and be tested in parallel, which related with scheduling test paths and integrating test results.

A. Schedule Parallel Test Paths

The goal of test task scheduling is to find an assignment function to dispatch the test tasks for minimizing test time. This paper considers test path length as decision making criteria.

path-length: the number of nodes between beginning and end nodes in a test path.

priority: the longer path has higher priority; the front path near the start point of the graph has higher priority.

Given a set of test paths set S = {s_1, s_2, ..., s_m}, there is a set of VMs M in cloud, M = {m_1, m_2, ..., m_n} and a set T for fail test paths. The schedule procedure in AT-Cloud is as follows.

1) Sorting test paths

If S is not empty, then sort the test paths based on path's priority in descending order.

2) Choose or creating VMs

If machine set M is empty, which means VMs in AT-Cloud with the required environment for the test task is not available, notify the cloud to create a VM for the test task according to the quantity of test paths and the resource requirements. If M is not empty, choose a VM that satisfied the test requirement.

3) Dispatch test paths to VMs

Notify the cloud controller to get available VMs and dispatch test paths to the VMs according to their priority order. If a test path is dispatched, then remove it from set S.

4) Monitoring test status

Monitor all the test paths and machines status in AT-Cloud. If a VM is free and S is empty then terminate the VM to release cloud resource.

5) Recording fault test path

If any fault detected, related test path will be added into test path set T, and also report to the tester.

6) Scheduling the waiting list

After a test path has been finished, a VM will be free. If S is not empty, return to step 3.

Generally speaking, this process is repeated until all the test paths are scheduled for processing on VMs, AT-Cloud is automatically growing and shrinking resources as needed.

B. Integrate Test Result

Every test path on different VM may return a test result and all the test results should be integrated in to a test report. There are several descriptions:

1) The successful test results and the failure ones are integrated into different files respectively so as to gain a fast overview of program errors.

2) The errors are marked on the scene point of the test scene flow graph which visualize the test results and show an integrity view of the test results.

3) The starting time, ending time of the test, and also other test information are recorded in a test result file.

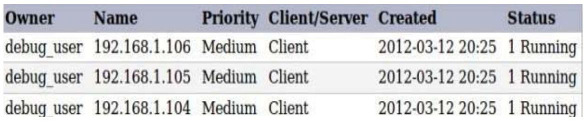
4) A tester can download the result files and also examine the test result through Test-Manager.

V. Experiments

In this section, we show an experiment of ATM software parallel path tests over AT-Cloud. The test paths are explained in section III.

Four physical machines are used in the experiment, each of which has 2 CPU and 2GB memory. One is installed Eucalyptus cloud controller, cluster controller, SC controller, one is installed Autotest server and our Test_Manager and the other two are install Xen and Eucalyptus NC server and running Centos5.5.

We prepare the software image which satisfies the test environment requirement according the system configuration file submitted by the tester from Test_Manager first. AT-Cloud creates three virtual machines and all the test paths are dispatched on different machines. Figure 4 shows the parallel test paths execution status on every Autotest client. It takes less time for parallel paths test over AT-Cloud than serial paths test on a single machine.



Owner	Name	Priority	Client/Server	Created	Status
debug_user	192.168.1.106	Medium	Client	2012-03-12 20:25	1 Running
debug_user	192.168.1.105	Medium	Client	2012-03-12 20:25	1 Running
debug_user	192.168.1.104	Medium	Client	2012-03-12 20:25	1 Running

Figure 4 Parallel test paths execution

VI. Conclusions

This paper proposed an automated testing platform over cloud called AT-Cloud and proposes a parallel path execution method for software function testing. Test scene flow graph is an effective way to describe the function points and test paths, which contributes to parallel paths finding and test case scripts generating.

AT-Cloud not only permits automatically created or selected a virtual machine according the test environment requirements and also schedule multi test paths to the scalable computing environment to implement parallel testing. Further, we intend to generate test cases automated based on test scene flow graph and consider more on test paths schedule algorithm, do more experiments on large scale software which can really show advantages of software test over Cloud.

Acknowledgment

This study is supported by Innovation Program of Shanghai Municipal Education Commission (12ZZ060) and Shanghai Foundation for Shanghai Key Laboratory (09DZ2272600).

References

[1] Lian Yu, Wei-Tek Tsai, and Xiangji Chen, Linqing Liu, Yan Zhao, Liangjie Tang, Testing as a Service over Cloud, IEEE International Symposium on Service Oriented System Engineering, 2010.

[2] T.Banzai ,H.Koizumi, R.Kanbayashi, T.Imada, H.Kimura, T.Hanawa, M.Sato, D-Cloud: Design of a software testing environment for reliable distributed system using cloud computing technology, in Proc.2nd International Symposiums on Cloud Computing in conjunction with CCCGrid 2010, May 2010.

[3] Toshilhiro Hanawa, Takayuki, Hitoshi Koizumi, Ryo Kanbayashi, Takayuki Imada, Mitsuhisa Sato, Large-Scale Software Testing Environment using Cloud Computing Technology for Dependable Parallel and Distributed Systems, Third International Conference on Software Testing, 2010 IEEE.

[4] SOASTA CloudTest Architecture, http://www.alldaytest.com/index.do?lan=cn.

[5] D.Nurmi, R.Wolski, C.Gregorczyk, G.Obertelli, S.Soman, L.Youseff, D.Zagorodnov, The Eucalyptus Open-source Cloud-computing System, IEEE/ACM International Symposium on Cluster Computing and the Grid, 2009.

[6] Autotest structure overview,
http://autotest.kernel.org/wiki/AutotestStructure.

[7] Xen overview, http://xen.org

[8] Paul Ammann, Jeff Offutt, Introduction to Software Testing, China Machine Press, 2008, pp.35- 42.

[9] Nan Li, Jeff Offutt, An Experimental Comparison of Four Unit Test Criteria: Mutation, Edge-Pair, All-uses and Prime Path Coverage, IEEE International Conference on Software Testing Verification and Validation Workshops, 2009.

An Empirical Study of Execution-Data Classification Based on Machine Learning

Dan Hao, Xingxia Wu, Lu Zhang
Key Laboratory of High Confidence Software Technologies, Ministry of Education
Institute of Software, School of Electronics Engineering and Computer Science, Peking University,
Beijing, 100871, P. R. China
{*haod, wuxx10, zhanglu*}*@sei.pku.edu.cn*

Abstract

As it may be difficult for users to distinguish a passing execution from a failing execution for a released software system, researchers have proposed to apply the Random Forest algorithm to classify remotely-collected program execution data. In general, execution-data classification can be viewed as a machine-learning problem, in which a trained learner needs to classify whether each execution is a passing execution or a failing execution. In this paper, we report an empirical study that further investigates various issues in execution-data classification based on machine learning. Compared with previous research, our study further investigates the impact of the following issues: different machine-learning algorithms, the numbers of training instances to construct a classification model, and different types of execution data.

1. Introduction

After a software system is delivered to its different users, it may be still necessary to analyze or measure the released software system to improve software quality. Although a software system usually has been tested before delivery, the testing process cannot guarantee the reliability of the release. Before delivering a software system, developers need to test the software system in-house, assuming that the software system will be used in the field as it is tested. However, this assumption may not always be satisfied [9], due to varied running environments or unexpected inputs for instance. Consequently, a software system still needs to be tested and analyzed after it is delivered.

However, as a released software system runs on sites of remote users, the testing and analysis activities on a released software system are different from those in-house. For instance, developers can hardly detect failures directly based on the behavior of a released software system because its behavior occurs on the sites of remote users. Moreover, analysis on a released software system may require instrumentation, and only **lightweight** instrumentation is allowed so as to incur as little as possible extra running cost. Consequently, several techniques [6, 7, 11] are proposed to support Remote execution Analysis and Measurement of Software Systems (abbreviated as RAMSS by Haran and colleagues [5]). RAMSS techniques usually collect data by instrumenting instances of a software system used by different remote users and then analyze the collected execution data.

In RAMSS, a fundamental issue is to distinguish a failing execution from a passing execution. Addressing this issue helps developers understand the behavior of a released software system. For instance, developers know how often failure occurs in the field when remote users run the released software system by counting the numbers of passing executions and failing executions. Moreover, distinguishing a failing execution from a passing execution is also important for developers to debug a released software system because such information helps developers know in what circumstances failures occur.

To automate distinguishing failing executions from passing ones, Haran and colleagues [5] proposed to collect execution data in the field and distinguish a failing execution from a passing execution by applying a machine-learning algorithm (i.e., the Random Forest algorithm) to the collected execution data. According to the empirical study conducted by Haran and colleagues [5], their approach is promising, as it can reliably and efficiently classify the execution data. However, there are several important factors not addressed in their study. First, their study investigated only one machine-learning algorithm, but there are quite a few machine-learning algorithms for classification. Second, their study did not investigate the impact of the number of instances for constructing the classification model, but this number may be an important factor of the overhead of execution-data classification. Third, their study investigated execution data only in the form of counts for methods, statements and so on, but did not investigate execution data in the form of coverage, which is widely used in practice.

In this paper, we report an empirical study that evaluates the impact of these three factors on the effectiveness of execution-data classification. The aim of our empirical study is to provide evidence on the determination of the three important factors in practice. For the first factor, we investigate three popular machine-learning algorithms: the Random Forest algorithm (RF), the Sequential Minimal Optimization algorithm(SMO), and the Naive Bayes algorithm(NB). For the second factor, we investigate training sets of different sizes. For the third factor, we investigate five types of execution data: the statement count, the statement coverage, the method count, the method coverage, and the branch coverage. The main contributions of this paper are as follows. First, to our knowledge, our study is the first empirical study on the impact of the three factors in execution-data classification based on machine learning. Second, our study provides guidelines for the application of execution-data classification based on machine learning regarding the choice of machine-learning algorithms, the number of training instances, and the type of execution data.

2. Execution-Data Classification Based on Machine Learning

Haran and colleagues [5] proposed to collect execution data by instrumenting instances of a released software system and then classify passing executions and failing executions based on these collected execution data using the Random Forest algorithm, which is an implementation of tree-based classification. This technique can be viewed as an application of a classification algorithm in machine learning for execution-data classification. In the following, we generalize Haran and colleagues' technique to execution-data classification based on machine learning.

For the ease of presentation, let us consider the execution data based on the statement count. For a program consisting of n statements (denoted as ST=$\{st_1, st_2, ..., st_n\}$), the execution data of a test case (denoted as t) is represented as a series of numbers, each of which is the number of times a statement is executed when using t to test the program. All the test cases (whose execution data are already obtained) are divided into two sets: the training set and the testing set[1]. Each test case in the training set is also associated with a label that indicates whether the test case is a passing one or a failing one. For test cases in the testing set, it is unknown whether each such test case is a passing one or a failing one. The problem of execution-data classification is to use the training set to predict whether each test case in the testing set is a passing one or a failing one.

Table 1. Subjects

Program	Ver.	Test	TrainingSet	LOC(Exe)	Met.	Bra.
print_tokens	7	4,072	41∼203	565(203)	18	35
print_tokens2	10	4,057	41∼203	510(203)	18	70
replace	32	5,542	58∼289	563(289)	21	61
schedule	9	2,627	33∼162	412(162)	18	25
schedule2	10	2,683	29∼144	307(144)	16	31
tcas	41	1,592	14∼67	173(67)	9	16
tot_info	23	1,026	14∼136	406(136)	7	36
Space	38	13,585	1,244∼6,218	9,564(6218)	136	530

3. Experimental Study

In this experiment, we plan to evaluate the impact of three factors in execution-data classification so as to answer the following three research questions. **RQ1—** which machine-learning algorithm produces the best results in execution-data classification? **RQ2—** what is the minimal number of training instances that are needed to produce good enough results in execution-data classification? **RQ3—** which type of execution-data produces the best results in execution-data classification?

Table 1 presents the details of the eight C subjects used in this experiment, whose source code and test collections are available from Software-artifact Infrastructure Repository (http://sir.unl.edu/portal/index.php) [3]. Specifically, the table gives the name of each subject, number of faulty versions, number of test cases in the test collections, the range of the numbers of training instances used to construct a classification model in our empirical study, the number of lines of code, and the number of branch statements. Moreover, the "Exe" (abbreviation of executable statements) within parentheses represents the number of executable statements, which presents the real size of a subject. As the faulty programs in Table 1 are all single-fault programs, we totally constructed 80 multiple-fault programs for the eight subjects by extracting the faults from the single-fault programs of each subject and then seeding more than one faults into each subject.

3.1. Independent and Dependent Variables

The independent variables come from the three research questions and are defined as follows.

The first independent variable *Algorithm* refers to the machine-learning algorithm that is used to build a classification model. Many algorithms have been proposed in the literature of machine learning. Here we investigate the following three representative algorithms: Random Forest (RF) [2], Naive Bayes (NB) [8], and Sequential Minimal Optimization (SMO) [12], because these algorithms have been found to be effective in the literature of machine learning and have been widely used. Moreover, these algorithms

[1] The training set and the testing set are two terminologies in machine learning. Note that the testing set is not related to software testing.

are implemented on Weka 3.6.1 by using its default setting since this experiment is designed to answer the three research questions.

The second independent variable *TrainingSet* refers to the number of instances (i.e., executions) in a training set. As the number of available training instances may be associated with the scale of a program, we use the ratio between the number of training instances and the number of executable statements to represent *TrainingSet*, which is set to be 20%, 40%, 50%, 60%, 80%, and 100% in our empirical study.

The third independent variable *Attribute* refers to which type of execution data has been collected during the execution of the program and used to classify the execution data. In our empirical study, the values of variable $Attribute$ can be any of the following: statement count (abbreviated as #statements, which is the number of times each statement has been executed), statement coverage (abbreviated as ?statement, which is whether each statement has been executed), method count (abbreviated as #method, which is the number of times each method has been executed), method coverage (abbreviated as ?method, which is whether each method has been executed), and branch coverage (abbreviated as ?branch, which is whether each branch has been executed). To record the execution information on branches when running the program, we used "GCOV" command of GCC, whose collected branches are predicates within a branch condition, not the whole branch condition. Moreover, the predicates whose value is true or false are taken as different branches.

The dependent variables in our empirical study are the results of execution-data classification measured by ROC analysis [4] since it has advantages overs other measures like precision-recall and accuracy. Specifically, our study uses the "Area under a ROC curve (usually abbreviated as AUC)" to measure the performance of an execution-data classification approach. The AUC is always between 0 and 1. The bigger the AUC is, the higher performance the corresponding classification approach has. Moreover, a realistic classifier is always no less than 0.5 since the AUC for the random guessing curve is 0.5.

3.2. Process

First, we ran each subject with its test cases by recording the five types of execution data as well as the outcome of an execution (i.e., whether it is passing or failing).

Second, we took all the test cases with their execution information (i.e., execution data and outcome) as instances and split all the instances into a training set and a testing set. Although all the test cases are labeled with either passing or failing, our experiment randomly selected some test cases into a training set, and took the rest test cases as a testing set. The training set is the set of test cases whose execution data and outcome are taken as input to build a classifier, whereas the testing set is the set of test cases that are used to verify whether the classified outcome based on the classifier is correct. Since all the test cases are known to be passing or failing in our empirical study before building the classifier, we know whether the classification is correct or not by comparing the classification result with its actual outcome.

Moreover, for each faulty program, our empirical study randomly selected $n * 20\%$, $n * 40\%$, $n * 50\%$, $n * 60\%$, $n * 80\%$, or n test cases from its whole test collection as a training set and took the rest test cases as a testing set, where n is the number of executable statements for each subject shown by Table 1. To reduce the influence of random selection, we repeated each test-selection process 100 times.

Third, we applied each of the three machine-learning algorithms to each training set and recorded the classified outcome of test cases in its corresponding testing set. As the outcome of each test case is known, we know whether the classification is correct. We recorded the number of passing test cases that have been **correctly** classified to be passing, the number of passing test cases that have been classified to be failing, the number of failing test cases that have been **correctly** classified to be failing, and the number of failing test cases that have been classified to be passing. Then for each faulty program with a machine-learning algorithm, we calculated its corresponding AUC. As each subject has several faulty versions, we calculated their average AUC as the result of the corresponding subject. Our experiment is performed on an Intel E7300 Dual-Core Processor 2.66GHz with 2G memory.

3.3. Threats to Validity

The construct threat of our empirical study lies in the measures, although ROC analysis has been widely used in evaluating classifiers in machine learning because it has advantages over the accuracy, error rate, precision, recall and can decouple classifier performance from class skew and error costs [4]. The main external threat comes from the subjects and the seeded faults. Although these subjects including the faults have been widely used in the literature of software testing and analysis, we will perform more experiments on larger programs with real faults.

3.4. Results and Analysis

In our empirical study we exclude the faulty programs whose corresponding test collection has less than 5% failing test cases because its percentage of correct classification would be larger than 95% if the test cases are always classified to be "passing". After excluding such biased data,

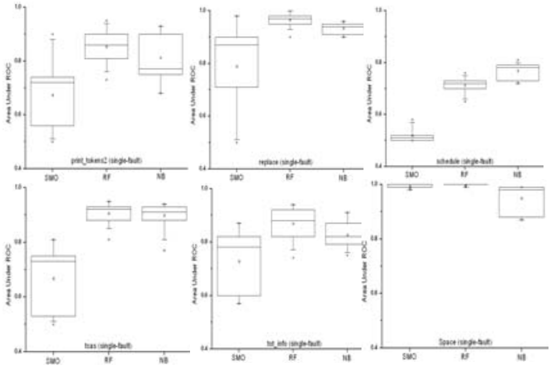


Figure 1. Boxplots of average AUC results for various machine-learning algorithms

Table 2. Paired Samples T-Test on Machine-Learning Algorithms (α=0.05)

Single-Fault Programs	t-value	Sig.	Result
SMO - RF	-18.767	0.000	*Reject*
SMO - NB	-14.576	0.000	*Reject*
RF - NB	3.970	0.000	*Reject*
Multiple-Fault Programs	**t-value**	**Sig.**	**Result**
SMO - RF	-30.602	0.000	*Reject*
SMO - NB	-16.362	0.000	*Reject*
RF - NB	6.950	0.000	*Reject*

our empirical study has only the results of six single-fault programs except for print_tokens and schedule2, and seven multiple-fault programs except for Space. Our empirical study does not acquire the results of execution-data classification for Space based on its statement coverage or statement count because the maximal memory of Java Virtual Machine is not large enough to classify executions which have a large number of attributes (i.e., the statement coverage or the statement count).

(1)RQ1: Machine-Learning Algorithms

Figure 1 shows the distribution of the average AUC of the execution-data classification approach with the same machine-learning algorithm for each single-fault subject. The horizontal axis represents the three machine-learning algorithms, whereas the vertical axis represents the average AUC. According to Figure 1, the classification approach with SMO is usually much less effective than the approach with RF. The classification approach with RF produces close AUC results to the approach with NB. Moreover, sometimes (e.g., for replace) the execution-data classification approach with RF is better than the execution-data classification approach with NB, whereas sometimes (e.g., for schedule) the latter approach is better than the former.

To further compare the three machine-learning algorithms, we performed the paired samples t-test on the average AUC results by comparing each pair of results of the execution-data classification approaches with the same subject, the same percentage of training instances, and the same type of execution data, but different machine-learning algorithms. The results are shown by Table 2. Here the t-test is performed separately on results of single-fault programs and multiple-fault programs because the machine-learning based execution-data classification approach is intuitively more effective to classify an execution for single-fault programs than for multiple-fault programs considering the impact of multiple faults.

According to this table, the hypothesis that neither RF nor NB is superior to the other in execution-data classification is rejected (for both single-fault and multiple-fault programs). Moreover, execution-data classification using RF wins the classification approach using NB since its calculated t-value is positive. That is, for single-fault and multiple-fault programs, execution-data classification with RF is significantly better than the approach with NB. Similarly, execution-data classification with either RF or NB is significantly better than the approach with SMO.

(2)RQ2: Training Set

According to the average AUC, with the increase of the number of training instances (i.e., variable *TrainingSet* increases from 20% to 100%), the average AUC of each subject usually increased except for a few data.

However, it is not practical to construct a classifier based on a large number of training instances (i.e., execution data with known outcome) in execution-data classification. Thus, we are more interested in the experimental results of execution-data classification using a small number of training instances. According to the experimental results, as we increase the number of training instances, the average AUC results increase slightly. That is, the execution-data classification approach fed with 20% training instances has close AUC results to the approach with 100% training instances, which produces reliable classification results. Consequently, for any subject whose number of executable statements is n, execution-data classification has been evaluated to be reliable even if the number of training instances is $n * 20\%$.

(3)RQ3: Type of Execution Data

To show the influence of the types of execution data on AUC results of execution-data classification, we draw some boxplots for single-fault programs by statically analyzing average AUC results of same type of execution data. For each program, the five boxplots for various types of execution data have observable difference. For instance, the AUC results of execution-data classification with the branch coverage usually distribute in a smaller range than the approach with any of the other types of execution data. That is, execution-data classification with the branch coverage is more stable than the approach with the other type of execution-data although some other types of execution data

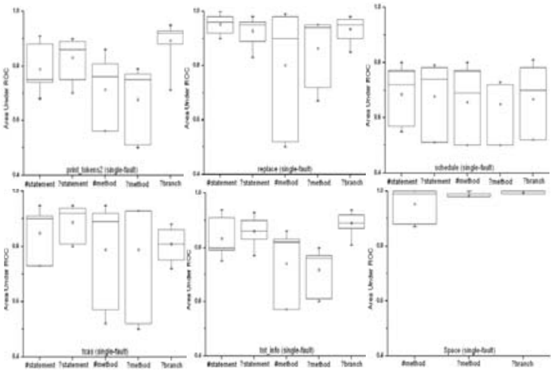


Figure 2. Boxplots of average AUC results for various type of execution data

Table 3. Paired Samples T-Test on Types of Execution Data (α=0.05)

Single-Fault Programs	t-value	Sig.	Result
#statement - ?statement	-2.586	0.011	*Reject*
#method - ?method	-0.790	0.432	Not Reject
?branch - ?statement	0.437	0.663	Not Reject
?branch - ?method	7.832	0.000	*Reject*
?branch - #statement	2.318	0.023	*Reject*
?branch - #method	7.736	0.000	*Reject*
#statement - #method	6.331	0.000	*Reject*
?statement - ?method	10.281	0.000	*Reject*
Multiple-Fault Programs	**t-value**	**Sig.**	**Result**
#statement - ?statement	-6.302	0.000	*Reject*
#method - ?method	2.308	0.044	*Reject*
?branch - ?statement	6.418	0.000	*Reject*
?branch - ?method	16.013	0.000	*Reject*
?branch - #statement	7.607	0.000	*Reject*
?branch - #method	13.126	0.000	*Reject*
#statement - #method	10.582	0.000	*Reject*
?statement - ?method	16.582	0.000	*Reject*

(e.g., the statement count, the statement coverage) provide more information than the branch coverage. Moreover, the distribution area of AUC results of execution-data classification with the branch coverage is mostly higher than the approach with any of the other four types of execution data.

Table 3 presents the paired samples t-test on the average AUC results for various types of execution data. According to this table, execution-data classification with the statement coverage is significantly better than the approach with the statement count in both single-fault programs and multiple-fault programs. Moreover, the approach with the branch coverage is significantly better than the approach with the statement count, the method count, or the method coverage in both single-fault and multiple-fault programs. Although in single-fault programs, the execution-data classification approach with the branch coverage has no significant difference with the approach with the statement coverage, the former wins the latter in most comparisons. Moreover, in multiple-fault programs, the former is significantly better than the latter.

3.5. Summary and Discussion

Execution-data classification has been evaluated to be effective in our empirical study because this approach correctly classified most executions in our empirical study. Among the three machine-learning algorithms, RF is significantly better than NB in correctly classifying executions, while both of them are significantly better than SMO. Execution-data classification usually produces better (at least no worse) results as we increase the number of training instances. Moreover, even if the training set contains a small number of instances (i.e., one-fifth of the executable statements contained by a subject), execution-data classification is still reliable. Execution-data classification with execution information on branches is usually better than the approach with execution information on statements or methods. Specifically, the former is significantly better than the approach with the method coverage, the method count, or the statement count. Moreover, execution-data classification with execution information on the statement information is significantly better than the approach with execution information on the method information. Execution-data classification with execution information on the statement coverage is significantly better than the approach with execution information on the statement count.

Besides the preceding conclusion, we discuss some issues involved in applying execution-data classification based on machine learning. The first issue is the computational cost of the machine-learning algorithms. Our empirical study evaluated the classification performance of machine-learning algorithms, and found their computational costs are different during the experiment. To applying execution-data classification in practice, we may need to consider the computational cost as well as the classification performance of machine-learning algorithms. Among the three machine-learning algorithms, SMO is much costly than RF and NB. The second issue is the type of execution data. As software systems have been delivered to remote users, it is not preferable to collect execution data by heavyweight instrumentation because it may significantly hamper the use of software systems. According to Table 1, the number of statements is much larger than that of methods and that of branches. Therefore, it is definitely less costly to instrument a program at methods or branches than at statements. Moreover, the execution data on statements occupy more space than the execution data on methods or branches. Consequently, considering the instrumentation cost and the accuracy of execution-data classification, execution-data classification with execution information on branches is more applicable than the approaches with either statements or methods.

4. Related Work

As the testing and analysis activities on released software systems are different from those in-house, researchers have proposed some techniques to facilitate testing and analysis of released software systems. Pavlopoulou and Young [11] proposed a residual test coverage monitoring tool to selectively instrument a Java program under test so that the performance overhead is acceptable. As a software system has distributed in sites of many remote users, a monitoring task can be divided into several subtasks, and each subtask is assigned to different instances of the software system and can be implemented by minimal instrumentation. This approach, called the Gamma system, is implemented by Orso and colleagues [1, 10]. The preceding research aims to propose different instrumentation strategies in monitoring a released software system. However, our work uses execution data, which is collected by these instrumentation techniques, to distinguish failing executions from passing ones.

The execution data collected from the field can be analyzed and used for the purpose of software maintenance and evolution. Liblit and colleagues [6, 7] proposed to gather executions by using Bernoulli process and then isolate bugs based on the sampling executions. Orsa and colleagues [9] investigated the use of execution-data from the field to support and improve impact analysis and regression testing in-house. Although both of these research and ours use the execution data, they have different aims. These work aims at fault localization, impact analysis or testing, whereas our work aims at classifying executions.

5. Conclusion

In this paper, we performed an empirical study on execution-data classification by varying the three factors: machine learning algorithms, number of training instances, and type of execution data. Among the three machine-learning algorithms, the Random Forest algorithm makes the classification approach produce significantly better results than the Naive Bayes algorithm. Either of these algorithms makes the classification approach produce significantly better results than the Sequential Minimal Optimization algorithm. As we increased the number of training instances, the classification model usually produces better classification results. However, execution-data classification approach still correctly classifies most execution data when fed with a small number of training instances (i.e., 1/5 of the number of executable statements contained by a program). Moreover, when the type of execution data is branch coverage, the corresponding classification approach produces better results than the approach with the other types of execution data.

Acknowledgements

This work is supported by the National Basic Research Program of China under Grant No. 2009CB320703, Science Fund for Creative Research Groups of China under Grant No. 60821003, the National Natural Science Foundation of China under Grant No. 91118004, and the National Natural Science Foundation of China under Grant No. 60803012.

References

[1] J. Bowring, A. Orso, and M. J. Harrold. Monitoring deployed software using software tomograph. In *Proc. ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering*, pages 2–8, 2002.

[2] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, Oct. 2001.

[3] H. Do, S. Elbaum, and G. Rothermel. Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact. *Empirical Software Engineering*, 10(4):405–435, 2005.

[4] T. Fawcett. An introduction to roc analysis. *Pattern Recognition Letters*, 27:861–874, 2006.

[5] M. Haran, A. Karr, A. Orso, A. Porter, and A. Sanil. Applying classification techniques to remotely-collected program execution data. In *Proc. 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 146–155, 2005.

[6] B. Liblit, A. Aiken, A. X. Zheng, and M. I. Jordan. Bug isolation via remote program sampling. In *Proc. ACM SIGPLAN 2003 Conference on Programming Languages Design and Implementation*, pages 141–154, 2003.

[7] B. Liblit, M. Naik, A. X. Zheng, A. Aiken, and M. I. Jordan. Scalable statistical bug isolation. In *Proc. 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 15–26, June 2005.

[8] A. McCallum and K. Nigam. A comparison of event models for naive bayes text classification. In *Proc. AAAI-98 Workshop on Learning for Text Categorization*, pages 41–48, 1998.

[9] A. Orso, T. Apiwattanaapong, and M. J. Harrold. Leveraging field data for impact analysis and regression testing. In *Proc. 10th ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pages 128–137, 2003.

[10] A. Orso, D. Liang, M. J. Harrold, and R. Lipton. Gamma system: Continuous evolution of software after deployment. In *Proc. the International Symposium on Software Testing and Analysis*, pages 65–69, July 2002.

[11] C. Pavlopoulou and M. Young. Residual test coverage monitoring. In *Proc. 21st International Conference on Software Engineering*, pages 277–284, May 1999.

[12] J. C. Platt. Fast training of support vector machines using sequential minimal optimization. *Advances in kernel methods: support vector learning*, pages 185 – 208, 1999.

Identification of Design Patterns Using Dependence Analysis

Wentao Ma, Xiaoyu Zhou, Xiaofang Qi
School of Computer Science and Engineering
Southeast University
Key Lab of Computer Network and Information Integration
(Southeast University), Ministry of Education
Nanjing, China

Ju Qian
College of Computer Science and Technology
Nanjing University of Aeronautics and Astronautic
Nanjing, China

Lei Xu, Rui Yang
Department of Computer Science and Technology
Nanjing University
Nanjing, China

***Abstract*—Identification of design pattern instances in source codes facilitates software understanding and maintenance. Existing researches use three kinds of information in the identification: structure, control flow, and class or method names. In this paper, data flow information is used in identification. An approach is presented to identify instances of *Adapter, State, Strategy* and *Observer* patterns based on program dependence analysis. It brings more precise and detailed identification result. The effectiveness of the approach is illustrated by experiments on open source programs.**

Keywords-design pattern; program dependence; program comprehension; reverse engineering;

I. Introduction

GoF design patterns are widely used in practice [1]. Identification of design pattern instances (hereinafter referred to as "pattern instances" for short) in source codes recovers the design-code traceability, facilitates software understanding and maintenance.

Ideally, identification of pattern instances should identify all the classes, methods and variables that play roles in the design patterns.

Early researches focus on identifying pattern instances by structure information [9]. These approaches cannot distinguish design patterns having similar structures, such as *State* and *Strategy* pattern [5][6]. They are also not good at identifying methods playing roles in patterns.

Recently, some researches identify pattern instances by checking whether call sequences of the analyzed source codes are in accordance with that of the design patterns [7][11]. These approaches are more capable in identifying methods of design patterns. But they can't identify design patterns that have no special characteristics in method call sequences, such as *Adapter* pattern [5][8][13].

Some few researchers identify pattern instances using semantics of the classes or methods names [10][12]. These approaches are not mainstream in pattern instance identification research.

Existing researches can only identified two kinds of variables that play roles in design patterns: (1) field of one class that refers to another class in the pattern instance; (2) method parameter or return value whose type is the class in the pattern instances. Many other variables playing roles in design patterns cannot be identified in these researches.

Design patterns should be identified from multiple perspectives. The more perspectives are used, the more precise and detailed result will be obtained. As far as we know, except for our own work [2], data flow information has not yet been used in design pattern identification research. In this paper, we identify pattern instances using dependence relations in the analyzed codes. In this paper:

1) We present the characteristic of dependence relations in adapting method. Both *Adapter* and *State* pattern has adapting method, but *Strategy* pattern doesn't. Based on the distinguishment, we identify *Adapter* pattern instances, and distinguish *State* pattern instances from that of *Strategy*.

2) Dependence relations of variables within *Notify* method of *Observer* pattern is pres ented. Using the relations, we identify *Observer* instances and some variables playing roles in the pattern, such as the event objects notified to *Observer*s, and the variables that act as targets or results of observation.

Supported partially by the National Natural Science Foundation of China under Grant No. 90818027, and No. 60903026, and No. 61003156, and partially supported by the Fundamental Research Funds for the Central Universities under Grant No.1116020205, and No.1118020203.

Correspondence to: Xiaoyu Zhou, School of Computer Science and Engineering, Southeast University, Nanjing, China. E-mail: zhouxy@seu.edu.cn.

To validate the proposed approach, we conduct an experimental study on a popular open source software JHotDraw. We compare the result with that reported by some other researches, and find that our identification of *Adapter* pattern instances is more precise, and identification result of *State* and *Strategy* pattern instances is much more precise than that of existing researches. For identification of *Observer* pattern instances, our approach has relatively high precision. Our approach can identify instances that cannot be identified by other methods, and report more detailed information of the pattern instances, but it cannot identify *Observer* instances that hasn't data dependence relations between *Subject* and *Observer*.

The rest of this paper is organized as follows: Section II introduces the implementation of dependence analysis in our tool JDP-Detector; Section III and IV introduce approaches to identifying adapting method and *Notify* method of observer pattern respectively; Section V compares results of our experimentation with that of other researches; Section VI concludes the paper.

II. DEPENDENCE ANALYSIS AND ITS IMPLEMENTATION

We use two kinds of dependence relations in our approach: control dependence and data dependence. Control dependence relations are computed using the approach in [15]. Data dependence relations are computed directly from define-use relations [3]. The dependence relations is expressed by program dependence graph (PDG). The Inter-procedural dependence is expressed by system dependence graph (SDG), an extension of PDG [14].

Construction of SDG is based on information of method call. We use points-to analysis to determine the callee of polymorphic method call. As a pattern normally involves only several classes, we adapt Soot and implement a local points-to analysis that can start from entry of any method [4].

We develop a tool named JDP-Detector based on Soot to identify pattern instances in Java system. JDP-Detector takes *.class* files as input, and transforms them into Jimple intermediate representation [4]. Analysis is made on the Jimple codes. JDP-Detector firstly identify pattern instance candidates using structure information, and then make dependence analysis on these candidates to get more precise result.

III. IDENTIFICATION OF ADAPTING METHOD

A. Dependency Characteristics of Adapting Method

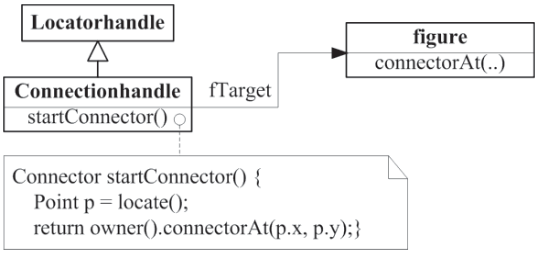
In *Adapter* pattern, class *Adapter* inherits from *Target*. Method of *Adaptee* has fulfilled the function required by *Target*, but has not the required interface, so the only work the adapting method of *Adapter* needs to do is to call the method of *Adaptee* and adapt the *Adptee*'s interface to that of the *Target*.

So, strictly speaking, an adapting method m_1 consists of three parts: (1) a call to method m_2; (2) statements (if available) before the call, whose only duty is to prepare parameters for m_2; (3) statements (if available) after the call, whose only duty is to transform the output of m_2 to output of m_1.

Based on the observation, we identify the adapting method according to the following conditions: (1) m_1 calls m_2; (2) the call statement depends on all the statements (when available) before the call and part or all of m_1's parameters; (3) the output of m_1 depends on the call statement and all the statements (when available) after the call; (4) the input and output of m_2 forms cut-set of m_1's data dependence graph. Condition 4 means that the dataflow paths cannot bypass the call statement.

B. Case Study

Fig. 1 illustrates the structure of an *Adapter* instance identified by JDP-Detector in JHotDraw 5.1. In the instance, *startConnector* is the *Adapter* interface and *connectorAt* is the *Adaptee* interface. Fig. 2 shows the PDG of *startConnector*, in which the yellow node represents a call to *connectorAt*, the green nodes represent the parameters of *startConnector* and the statements before the call to *connectorAt*, the red node represents the output of *startConnector*. As what can be seen from Fig. 2, the PDG of *startConnector* matches the dependence relation characteristic of adapting method.



```
Connector startConnector() {
   Point p = locate();
   return owner().connectorAt(p.x, p.y);}
```

Figure 1. An *Object Adapter* Instance in JHotDraw 5.1

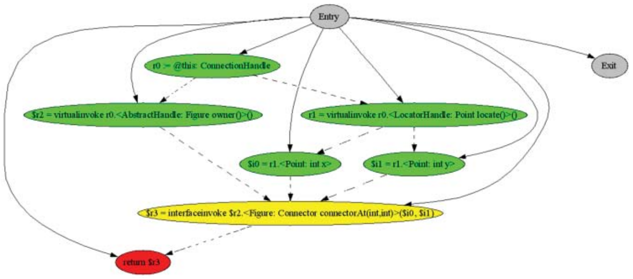

Entry
Exit
r0 := @this: ConnectionHandle
$r2 = virtualinvoke r0.<AbstractHandle: Figure owner()>()
r1 = virtualinvoke r0.<LocatorHandle: Point locate()>()
$i0 = r1.<Point: int x>
$i1 = r1.<Point: int y>
$r3 = interfaceinvoke $r2.<Figure: Connector connectorAt(int,int)>($i0, $i1)
return $r3

Figure 2. The Dependence Graph of *startConnector*

IV. IDENTIFICATION OF *NOTIFY* METHOD

A. Dependence Relation of Notify Method

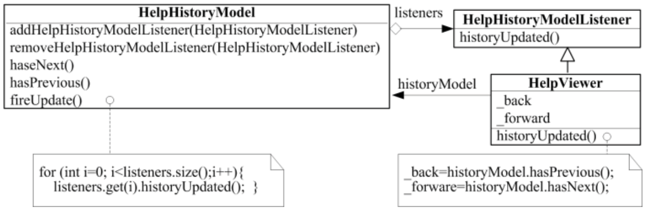
The intent of *Observer* pattern is to "*define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically*" [1] . When *Subject* changes its state, its *Notify* method invokes *Update* methods of all the *Observer*s.

In some cases, the *Subject* only informs the *Observer*s that an event happens. As a classical implementation, the *Subject* creates an event object, and sends it to *Observer* as a parameter of *Update*. In some other cases, the *Observer* needs to update values of its own fields to keep consistent with that of the *Subject*.

Our approach checks whether there is one of the following dependence relations in SDG of *Notify* candidate: (1) *Notify* method creates an object, and a parameter of *Update* depends on the object; (2) some fields of *Observer* depend on fields of *Subject*.

B. Case Study

Fig. 3 shows an *Observer* instance identified by JDP-Detector from open source software JEdit 4.3.2. In the instance, *HelpHistoryModel* and *HelpHistoryModelListener* act as *Subject* and *Observer*, respectively, and *fireUpdate* is the candidate of *Notify* method. JDP-Detector finds that there are dependence paths between fields of *HelpHistoryModel* and *HelpHistoryModelListener* in SDG of *fireUpdate*. This indicates that *fireUpdate* acts as *Notify* method. Correspondence between *Observer* fields and *Subject* fields can be observed by following the dependence paths: *HelpHistoryModelListener._back* is the observed result of target *HelpHistoryModel.historyPos*, and *HelpHistoryModelListener._forward* is the observed result of targets *HelpHistoryModel.history* and *HelpHistoryModel.historyPos*.



```
for (int i=0; i<listeners.size();i++){
   listeners.get(i).historyUpdated();  }
```

```
_back=historyModel.hasPrevious();
_forware=historyModel.hasNext();
```

Figure 3. An *Observer* Instance in JEdit 4.3.2

V. EXPERIMENT

This section reports the results of pattern instance identification in JHotDraw 5.1 (8.3 KLOC, 173 classes) using JDP-Detector, and compares it with the results reported by some other researches taking JHotDraw 5.1 as experimental subject.

A. Identification of Adapter Pattern Instances

JDP-Detector finds 17 *Adapter* instances, 14 of which are TP (true positive), and the precision is 82.35%. The 3 FP (false positive) are two *Strategy* instances and a *Decorator* instance. These patterns' structures are similar to that of *Adapter*, and adapting behavior is permitted to exist in the two patterns.

M. von Detten et al. obtain 67 instances using fuzzy metric [13]. They do not manually check the result, only figure out that the result contains some repeated instances, and instances with low membership degree is not eliminated. A. De Lucia et al. get 41 instances using structure information [8]. N. Tsantalis et al. identify pattern instances using similar scoring of structure graph, and recognizes 23 instances without distinguishing *Strategy* pattern from *Command* pattern [5]. Result of [5] is a subset of that of [8], and our result is a subset of that of [5]. In other words, we adopt a stricter identification standard.

B. Identification of State and Strategy Instances

State instances and *Strategy* instances are difficult to distinguish in existing researches. We believe that, in *State* pattern, *Context* has an adapting method, which delegates its duty to *State,* while in *Strategy* pattern, *Context* only invokes the algorithm encapsulated in *Strategy*, the invocation doesn't need to be an adapting method. This identification standard is not apparent according to descriptions in [1], but it brings good identification result in our experiment.

JDP-Detector identifies 37 *Strategy* instances, 35 of which are TP, and the precision is 94.6%. It also identifies 8 State instances, 6 of which are TP, and the precision is 75%.

Table I shows the recognized *State* instances. Manual analysis reveals that instances 3 and 8 in Table I are *Adapter* and *Strategy* instances respectively.

TABLE I. *STATE* INSTANCES IDENTIFED BY JDP-DETECTOR

	Context	State	Manual
1	AbstractConnector	Figure	T
2	AbstractTool	DrawingView	T
3	ConnectionHandle	Figure	F
4	LocatorHandle	Locator	T
5	PolygonHandle	Locator	T
6	StandardDrawingView	Drawing	T
7	StandardDrawingView	DrawingEditor	T
8	StandardDrawingView	Painter	F

Due to space limitation, the recognized *Strategy* instances can't be listed, two FP of which are *Command* and *Adapter* instances respectively. The *Command* instance is recognized because it has similar structure with *Strategy* pattern. The *Adapter* instance (*ChangeConnectionHandle*/*Figure*) recognized as *Strategy* instance is the same one as item 3 of Table I, as *ChangeConnectionHandle* is a subclass of *ConnectionHandle*. This FP is identified because JDP-Detector fails to take the inherited methods of the analyzed class into account.

N. Tsantalis et al. identify 22 instances using DPD without distinguishing *State* from *Strategy* [5]. Y. Guéhéneuc et al. infer structure characteristic of design pattern using machine learning and get 2 *State* instances and 4 *Strategy* instances [6]. A. De Lucia et al. get 43 *Strategy* instances (the precision is 46.51%) and 36 *State* instances (the precision is 5.6%) using both structure information and method call sequence information [7]. Based on the comparison, we can conclude that our approach is much more successful in distinguishing *State* instances from *Strategy* instances.

C. Identification of Observer Instances

As illustrated in Table II, JDP-Detector identifies 4 *Observer* instances, and reports the notified event objects in each instances. Three of them are TP, and the precision is 75%. Instance 4 in Table II is a FP, which contains event notifying behavior, but no one-to-many broadcast behavior.

A. De Lucia et al. identify 9 *Observer* instances using their tool DPRE [7]. Five of the nine instances are TP, and the precision is 55.56%. We compare these instances with our result, and find that three of their instances (*Drawing*/*DrawingChangeListener*, *StandardDrawing*/*DrawingChangeListener*, *Drawing*/*DrawingView*) are actually the same one as instance 2 of Table II. Moreover, we find a instance (*Connector*/*ConnectorFigure*) identified by DPRE is not a TP, because it neither contains broadcast of *Subject* to *Observer* nor event notification or data observation. Thus, in fact, DPRE identifies 7 instances, and only 2 of which is TP, so the precision is 28.6%.

TABLE II. *OBSERVER* INSTANCES IDENTIFIED BY JDP-DETECTOR

	Subject/Observer	Event Object	Manual
1	Figure/FigureChangeListener	FigureChangeEvent	T
2	StandardDrawing/ StandardDrawingView	DrawingChangeEvent	T
3	StandardDrawingView/ AbstractTool	MouseEvent	T
4	StandardDrawing/ FigureChangeEventMulticaster	FigureChangeEvent	F

We download N. Tsantalis's tool DPD 4.5 [5], and recognize 3 *Observer* instances (*StandardDrawing*/*DrawingChangeListener, StandardDrawingView*/*Painter, StandardDrawingView*/*Figure*) with it. The first one of their instances is the same as instance 2 of Table II. The other two instances is not included in our result. The second one of their instances is excluded by structure analysis of JDP-Detector. For the third instance, *Notify* method of *StandardDrawingView* passes its parameter instead of its fields to *Update* methods of *Figure*. The dependence path of the data transfer process is included in the SDG calculated by JDP-Detector, but is neglected by JDP-Detector.

Based on the comparison, we conclude that except for some defects in implementation of JDP-Detector, our approach is effective in revealing the data flow of the *Observer* pattern, and it identifies an instance (item 3 in Table II) that is not identified by DPD and DPRE.

D. Time Consumption of the Identification Process

Our experiment environment is as following: Intel Core(TM)2 Quad CPU Q8400 @ 2.66GHz processor, 3GB RAM, Windows XP operation system. Identification of *Adapter*, *Strategy*, *State*, and *Observer* instances in JHotDraw 5.1 by JDP-Detector uses 5.217s, 1.793s, 7.39s and 14.082s, respectively.

VI. CONCLUSION

In this paper, we propose an approach to identify design pattern instances in source codes using dependency information. Our approach improves the recognition precision (especially in distinguishing *State* instances from *Strategy* instances), and identifies the variables playing roles in design patterns that cannot be identified by existing approaches.

Compared with the approaches using method call sequences, our approach reflects the effect of method call but has less constraint to the method call sequence, and is somewhat more flexible.

Using data flow information, we can describe the characteristics of design patterns more detailed, which brings high identification precision, but may sometimes seems too strict. And furthermore, our approach can only identify design patterns that has obvious data flow characteristics.

REFERENCES

[1] E. Gamma, H. Richard, R. Johnson, and J, Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software. Boston: Addison-Wesley Longman, 1995.

[2] X. Zhou, J. Q ian, L. Chen, and B. Xu, "Identification of centrally managed aggregations in design patterns using shape analysis,". Journal of Software, Vol.21, No. 11, pp. 2725−2737, November 2010.

[3] A. V. Aho, M. S. Lam, R. Sethi and J. D. Ullman, Compilers: Principles, Techniques and Tools, 2nd ed., Addison-Wesley Longman Publishing Co., Inc., Boston, MA, 2007.

[4] R. Vallee-Rai, L. Hendren, V. Sundaresan, P. Lam, E. Gagnon and P. Co, "Soot — A Java optimization framework," In: IBM Centre for Advanced Studies Conference(CASCON), November 1999.

[5] N. Tsantalis, A. Chatzigeorgiou and G. Stephanides, "Design pattern detection using similarity scoring," IEEE Transactions on software engineering, Vol. 32, Issue 11, pp. 896-909, November 2006.

[6] Y. Guéhéneuc, J. Guyomarc'h and H. Sahraoui, "Improving design-pattern identification: a new approach and an exploratory study," Journal of Software Quality Control. Vol. 18, Issue 1, pp. 145–174, March, 2010.

[7] A. D. Lucia, V. Deufemia, C. Gravino and M. Risi, "Improving behavioral design pattern detection through model checking," In: 14th European Conference on Software Maintenance and Reengineering (CSMR), pp. 176-185, March 2010.

[8] A. D. Lucia, V. Deufemia, C. Gravino and M. Risi, "Design pattern recovery through visual language parsing and source code analysis," Journal of Systems and Software, Vol. 82, Issue 7, pp. 1177–1193, July 2009.

[9] N. Pettersson, Welf and J. Nivre, "Evaluation of a ccuracy in design pattern occurrence detection," IEEE Trans. Software Eng. Vol. 36, No. 4, pp. 575-590, July/August, 2010.

[10] S. Romano, G. Scanniello, M. Risi and C. Gravino, "Clustering and lexical information support for the recovery of design pattern in source code," In: 27th IEEE International Conference on Software Maintenance (ICSM), pp. 500-503, November, 2011.

[11] J. Y. Guéhéneuc and G. Antoniol, "Identification of behavioural and creational design motifs through dynamic analysis," Journal of Software Maintenance and Evolution: Research and Practice, Vol. 22, Issue 8, pp. 597‑627, December 2010.

[12] G. Rasool, I. Philippow and P. Mäder, "Design pattern recovery based on annotations," Advances in Engineering Software, Vol. 41, Issue 4, pp. 519–526, April 2010.

[13] M. V. Detten and D. Travkin, "An evaluation of the Reclipse tool suite based on the static analysis of JHotDraw," Technical Report tr-ri-10-322, Software Engineering Group, Heinz Nixdorf Institute, University of Paderborn, Oct 2010.

[14] S. Horwitz, T. Reps and D. Binkley, "Interprocedural slicing using dependence graphs," ACM Transactions on Programming Languages and Systems (TOPLAS), vol. 12, Issue 1, pp. 26-60, January 1990.

[15] J. Ferrante, K. Ottenstein, and J. Warren, "The program dependence graph and its use in optimization". ACM Trans. Program. Lang. Syst. Vol. 9, Issue 3, pp. 319-349, July 1987.

Slicing Concurrent Interprocedural Programs Based on Program Reachability Graphs

Xiaofang Qi Xiaojing Xu Peng Wang
School of Computer Science and Engineering
Southeast University
Nanjing, China
xfqi@seu.edu.cn, xiaojingliangxiao@163.com, pwang@seu.edu.cn

***Abstract*—Program slicing is an effective technique for analyzing concurrent programs. However, the slice is usually not precise when a traditional closure based slicing algorithm is applied to concurrent interprocedural programs. In this article, we present a three-phase algorithm for slicing concurrent interprocedural programs. In the algorithm, procedures involved in thread interactions are inlined and threaded interaction reachability graphs are further constructed to analyze dependences globally while allowing other procedures being handled as sequential programs. Our algorithm copes with the three imprecision problems: time travel, context insensitivity, and shared variable redefinition. As a result, a more precise slice is obtained with our algorithm because it solves the shared variable redefinition problem that could not be handled by existing high-precision algorithms. Preliminary experimental results show that our algorithm with the optimization of partial order technique is expected to have better performance than Nanda's classical high-precision slicing algorithms.**

Keywords: program slicing; concurrency;reachability analysis; context-sensitivity; dependence analysis

1. INTRODUCTION

With the introduction of multi-core processor architecture, concurrent programs are developed and used widely [1]. However, the analysis of concurrent programs is difficult due to its nondeterministic behaviors. Since most tools for sequential programs cannot be directly applied to concurrent programs, enabling techniques and tools for analyzing concurrent programs are required [1].

Program slicing is an important analysis technique that extracts the relevant parts that may affect the computation of a variable of interest[1-10]. It is very useful in various software engineering activities. As the dependence in sequential programs is transitive, the slice is often computed by traversing a system dependence graph [4]. However, when this closure based slicing algorithm is applied to concurrent interprocedural programs, three main issues arise and make the slice imprecise.

One is so-called time travel first found by Krinke [2]. When a shared variable defined in a thread is used in a concurrently executing thread, there exists interference dependence. As the order of concurrently executing statements is not deterministic, the dependence sequence including interference may be not in a valid execution chronology. Redundant statements are added to the slice.

As we know, the context-insensitive issue has been tackled in sequential programs with a two-phase algorithm [4]. But it occurs again in concurrent programs because of interference dependence. Nanda has presented an iterated two-phase slicing algorithm to ensure the correctness of the slice [3]. Unfortunately, the slice is still context-insensitive as the dependence sequences including interference might be not in a legal call/return sequence.

In addition, interference dependence may incur a third imprecision problem when a thread kills the definition of a shared variable in a different thread, i.e., the variable is redefined. Suppose that one node denoted by s is interference dependent on another node denoted by s' due to reference and definition of the variable v. It is possible that there exists some node that redefines v and may execute between s and s'. In such case, even if dependence sequence is in a valid execution chronology and a legal call/return sequence, the dependence sequence may still not be transitive and then lead to imprecision.

To date, researchers have proposed some precise algorithms to cope with the first two imprecision problems but not the third [1, 2, 3, 5, 9]. The goal of this paper is to present a novel algorithm that performs slicing concurrent interprocedural programs more precisely than previously reported. More precise slices are obtained with our algorithm as it handles the three imprecision problems while handling the shared variable redefinition imprecision problem that could not be done previously.

The remaining sections are organized as follows. In section 2, we motivate the issues of imprecision with an example. In section 3, we give our previous work. In section 4, we present a three-phase slicing algorithm for concurrent interprocedural programs and give our experimental results. Related work is discussed in Section 5, followed by conclusion.

2. MOTIVATION

Consider the simple and classical concurrency model used by Krinke and Nanda [2, 3]. In this model, concurrent programs consist of threads which share the same address space and communicate through shared variables. Shared variables are read and written atomically. Parallelism is expressed by *cobegin-coend* at the language level. Threads are generated by a *cobegin* and synchronized at the corresponding *coend*. For description purposes, *cobegin*, *coend* and any statement using shared variables is called interaction statement.

```
S1  a=1;
S2  b=5;

cobegin
{
S3  read x;
S4  call f(x);
S5  a=x;
S6  call g( );
S7  y=a+x;
}
{
S8  a=a+1;
}
coend

S9  proc f(u)
{
S10  u=5*u+10;;
}

S11  proc g( )
{
S12  a=b/10-15;
}
```

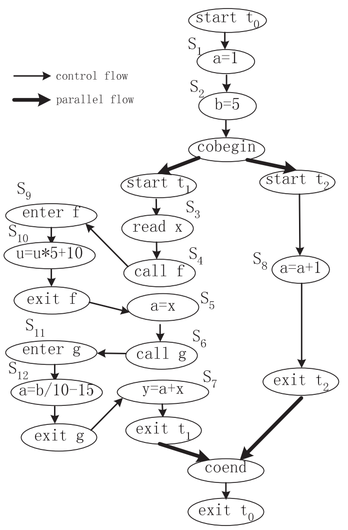


Fig.1 A Threaded Interprocedural Control Flow Graph (TICFG)

Each procedure has a control flow, which can be represented by a control flow graph (CFG). If we add the call edge and return edges, the CFGs for all procedures in the thread will be connected into an interprocedural control flow graph (ICFG) [2]. Similarly, if we add parallel flow edges, the ICFGs for all threads in the concurrent program will be connected into a threaded interprocedural control flow graph (TICFG) [2, 3]. Fig.1 shows an example. An interprocedural threaded witness is an ordered sequence of nodes such that any subsequence of nodes belonging to the same thread forms a realizable path, in which returns are matched with the corresponding calls [3].

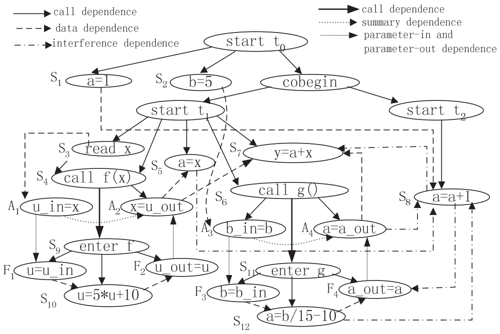


Fig.2 A Threaded System Dependence Graph(TSDG)

An System dependence graph(SDG) connects program dependence graphs of each procedure with call dependence edges and parameter dependence edges[4]. A threaded system dependence graph(TSDG) connects the SDGs of all threads with interference dependence edges. Fig.2 gives the TSDG of the program in Fig.1.

Examples for the first two imprecision issues can be found in [3]. In this paper, we only give an example in Fig.1 to show the third problem, i.e., shared variable redefinition imprecision. In Fig.2, S_8 is interference dependent on S_5 and S_5 is interference dependent on S_8. $<S_5, S_8, S_7>$ is an interprocedural threaded witness. However, S_7 is not dependent on S_5 because there are two possible executions, i.e., S_8 executes before or after S_{12}. In the former case, S_7 is dependent on S_{12} which kills the definition of variable *a* in S_8. In the latter case, S_7 is dependent on S_8 and yet S_8 is dependent on S_{12} which kills the definition of variable *a* in S_5. Therefore, S_7 is not dependent on S_5 in all paths. $<S_1, S_8, S_7>$ is another case showing that.

Most researchers have employed TICFG as program representation and focus on tackling the first two issues of imprecision, but not the third [1, 2, 3, 5, 9, 10]. From the example, we find that the parallel flow is simply handled as branch flow in TICFG, and the data dependence between parallel flows is not analyzed globally. In the next section, a threaded interaction reachability graph is proposed for such global analysis.

3. Dependence Analysis Based on Threaded Interaction Reachability Graph

Previous reachability analysis serializes only synchronization activities and does not handle parallel data flow[6]. We have extended serialization to parallel data flow to construct a threaded interaction reachability graph(TIRG). Then concurrent programs can be analyzed globally based on the TIRG.

3.1 Threaded Interaction Graph

To enhance the efficiency, we realize reachability analysis in the statement block level, rather than in the statement level. We first construct a block based flow graph for each thread, then conduct reachability analysis.

Interaction statements divide a thread into some various blocks, which are called thread regions in this paper. Given the CFG of a thread, if we search paths from the program point of the entry node or the point that immediately follows one interaction statement to the nearest interaction statement or exit node, the traversed single-entry and multi-exit subgraph would correspond to a thread region. Each thread region is represented by one node and each edge corresponds to one interaction. Such coarse grained representation of the thread is called threaded interaction graph(TIG).

A *threaded interaction graph* is a directed graph G_I=$<N_I, E_I, n_s, F_I, L_I>$, where N_I is the set of thread regions, E_I is the set of interaction activities, L_I is a function that assigns a label to each edge for indicating the type of the interaction, n_s belongs to N_I and is the entry thread region, F_I is the set of terminal states representing exit thread regions. In the threaded interaction graph, the label $t\rightarrow(t_{k1}, t_{k2}, \ldots, t_{kr})$ indicates that the thread t activates its child threads $t_{k1}, t_{k2}, \ldots, t_{kr}$ while $t\leftarrow(t_{k1}, t_{k2}, \ldots, t_k)$ indicates that t awaits the termination of its children $t_{k1}, t_{k2}, \ldots, t_{kr}$. The interaction involved in reads and writes of shared variables is labeled as the corresponding statement label.

Programs are required to be preprocessed and transformed before constructing TIGs. For each thread, procedures can be classified into two sets. One set includes all procedures involved in interactions. Each of them is required to be inlined at each call site. These procedures include interaction statements or call other procedures including interaction statements. In the latter case, procedures could be direct or indirectly called. To avoid infinite inlining, each cycle in the CFG of a recursive procedure needs to be folded into one node. Sets of variable definition and reference in such folded nodes include variables of definitions and references in each node in the cycle respectively.

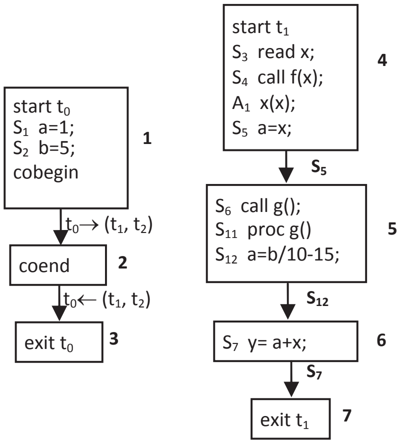


Fig.3 A Threaded Interaction Graph(TIG)

The other set includes all procedures not involved in interactions. None of them is required to be inlined. Each call statement is treated as a normal statement[4]. It is required to add one additional node for each parameter defined in the procedure at the call site. Let *s* be such an additional node, a_o be the actual parameter that are defined in the procedure. Then the definition variable is a_o, and the reference variables are the actual parameters that a_o is transitively dependent on. Dependence between parameters is identified by summary edges [4].

The TIGs for the program in Fig.1 are shown in Fig.3, in which g() is inlined. Since procedure f() does not involve any interaction, the actual parameter node A_1 is added after S_4. A_1 defines and uses variable x. So A_1 is represented as x(x). The block from the entry of t_0 to the nearest interaction statement *cobegin* forms a thread region marked as 1. Then *coend* forms thread region 2. Finally exit t_0 forms thread region 3. As the *cobegin* in t_0 activates thread t_1 and t_2, the edge from thread region 1 to 2 is labeled as $t_0 \rightarrow (t_1, t_2)$. Similarly, the edge from 2 to 3 is labeled as $t_0 \leftarrow (t_1, t_2)$ as *coend* awaits the termination of t_1 and t_2. In thread t_1, as S_5 and S_{12} are the interaction statements, the codes from the entry of t_2 to S_5 and from S_6 to S_{12} form thread regions 4 and 5 respectively. Edges from 4 to 5 and from 5 to 6 are labeled as S_5 and S_{12} respectively as they are involved in shared variables. Thread region 6, 7, 8 and 9 are generated accordingly.

3.2 Threaded Interaction Reachability Graph

Given the TIGs of *p* threads of a concurrent program, where the *i*th TIG is $G_I^i = <N_I^i, E_I^i, n_s^i, F_I^i, L_I^i>$ $(0 \leq i < p)$ and the main thread is t_0, we can generate threaded interaction reachability graph(TIRG). A TIRG is a flow graph whose nodes are *p*-tuples of TIG-nodes, one for each thread in the program, and the edges correspond to the interactions between threads.

As each component of a TIRG-node is a TIG-node which represents the control point of the corresponding thread, a TIRG-node represents a global execution state of the program, which is also called a reachable marking. The initial state is a distinguished TIRG-node $m_s = (n_s^0, \perp, \ldots, \perp)$, where $\perp$ represents the state that a thread is not activated or has been terminated. During reachability analysis, we begin from m_s and each new TIRG-node is generated via an interaction. The generated TIRG-node is the same as its immediate proceeding TIRG-node except in the components whose threads are involved in the corresponding interaction. There are three types of interaction, i.e., thread activation, thread waiting for termination, and reads and writes of shared variables.

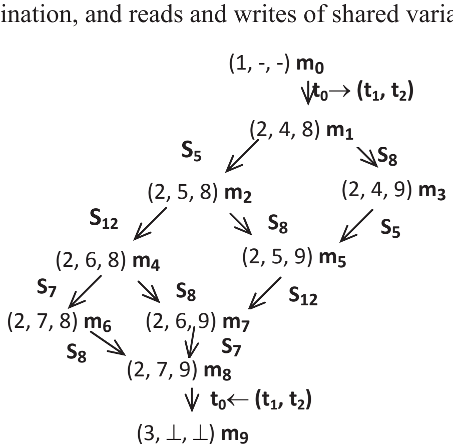


Fig.4 Threaded interaction graph of the example in Fig.1

A *threaded interaction reachability graph* is a directed graph $G_R = <M, E_R, m_s, M_F, L_R>$, where M is the set of TIRG-nodes, E_R is the set of interaction activities, L_R is a function that assigns a label to each edge for indicating thread region components involved in the interaction and the type of the interaction, m_s is the entry state, and M_F is the set of terminal states.

In Fig.4, from the initial state $m_0 = (1, \perp, \perp)$, t_0 activates t_1 and t_2, then $m_1 = (2, 4, 8)$ is generated. From m_1, as S_5 and S_8 are possible to execute, there are two successors of m_1. If S_5 executes, only the second component is modified from 4 to 5 and then $m_2 = (2, 5, 8)$ is generated. If S_8 executes, $m_3 = (2, 4, 9)$ is generated, which is the same as m_1 except the third component involved in the interaction S_8. From m_8, t_0 awaits the termination of t_1 and t_2, and then $m_9 = (3, \perp, \perp)$ is generated.

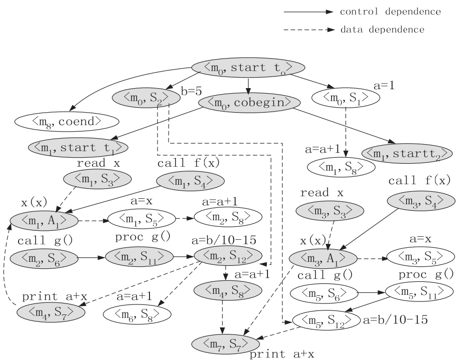
3.3 M-S pair program dependence graph

A statement executing in different program states implies that it may be in different interaction contexts, i.e., shared variables may have different definitions since different interaction sequences are executed. In Fig.4, S_8 can execute in m_2 or m_4. The definition of *a* in S_5 reaches S_8 in m_2 while the definition of *a* in S_{12} reaches S_8 in m_4. To discriminate a statement in different interaction contexts, we combine a statement with program state to generate M-S pair to represent different instances of one statement. Given a M-S pair Λ, the functions $m(\Lambda)$ and $s(\Lambda)$ return the state and the statement in Λ respectively. For a reachable state m, all statements appearing in their thread regions may execute in m and only these statements can be combined with m or they are meaningless otherwise.

An execution of a concurrent program can be represented by an ordered statement sequence. If each statement is combined with its corresponding program state, an M-S pair sequence, also called by M-S trace, is generated. In Fig.3 and Fig.4, along with the leftmost path in TIRG, we can get the following M-S trace:

L=(<m_0, start t_0>, <m_0, S_1>, <m_0, S_2>, <m_0, cobegin>, <m_1, start t_2>, <m_1, S_8>, <m_3, start t_1>, $\underline{\langle m_3, S_3\rangle}$, $\underline{\langle m_3, S_4\rangle}$, <$m_3$, A_1>, <m_3, S_5>, <m_5, S_6>, <m_5, S_{11}>, <m_5, S_{12}>, <m_7, S_7>, <m_8, coend>).

Assume that L= $(\Lambda_1, \Lambda_2, \ldots, \Lambda_n)$ is a M-S pair sequence, for all $1\le i<n$: Λ_{i+1} is control or data dependent on Λ_i, Λ_n is *transitively dependent* on Λ_1 if there is a M-S trace L′ such that all dependences in L arise in L′. TIRG serializes all interactions, and dependence analysis along TIRG is based on a specific execution order of interactions as in sequential programs. So, dependence sequence in MSDG is transitive, which has been proved in [7].



(m_1, start t_1)→(m_1, S_3), (m_1, start t_1)→(m_1, S_4), (m_1, start t_1)→(m_1, S_5), (m_1, start t_1)→(m_2, S_6), (m_1, start t_1)→(m_4, S_7), (m_1, start t_1)→(m_7, S_7), (m_1, start t_1)→(m_3, S_3), (m_1, start t_1)→(m_3, S_4), (m_1, start t_1)→(m_3, S_5), (m_1, start t_1)→(m_5, S_6), (m_1, start t_2)→(m_2, S_8), (m_1, start t_2)→(m_6, S_8), (m_1, start t_2)→(m_4, S_8), (m_1, start t_2)→(m_1, S_8).

Fig.5 MSDG for the program in Fig.1

An execution of a concurrent program may be represented by more than one M-S trace which has equivalent behaviors. We choose a representative where each statement not involved in interactions combines with the same state as the interaction statement. As shown in Fig.3 and Fig.4, S_3 is not an interaction statement, but it can execute in m_1 or m_3. If <m_3, S_3> is replaced with <m_1, S_3> in L, we can get another M-S trace completely equivalent with L. For simplicity, Only L is selected to be analyzed because S_3 and S_5 combine with m_3.

As TIRG serializes all interactions, interference data dependence is not necessarily treated specially. Hence, dependences based on TIRG are classified into control and data dependence. Assume that L= $(\Lambda_1, \Lambda_2, \ldots, \Lambda_n)$ is a M-S trace of a concurrent program CP, Λ' is control dependent on Λ, denoted by CD(Λ, Λ'), if the execution of $s(\Lambda')$ is determined by the execution of $s(\Lambda)$ in the execution of L. Λ' is data dependent on Λ, denoted by DD(Λ, Λ'), if $s(\Lambda')$ uses the variable defined in $s(\Lambda)$ in the execution of L.

A *M-S pair program dependence graph* (MSDG) of concurrent program CP is a directed graph G_D=<M, S, N_{MS}, E_D>, where M is the set of TIRGs of CP, S is the set of statements in CP, node set $N_{MS}\subseteq M\times S$, edge set $E_D=\{(\Lambda_i, \Lambda_j)|\ CD(\Lambda_i, \Lambda_j)\vee DD(\Lambda_i, \Lambda_j), \Lambda_i, \Lambda_j \in N_{MS}\}$.

The MSDG for the program in Fig.1 is shown in Fig.5. For clarity, some control dependences are not given in the graph, but listed with text.

4. Slicing Concurrent Interprocedural Programs

For some slicing criteria, only dependence graphs of procedures not required to be inlined are traversed; this is not true for others. For the former, traversing TSDG could be enough. For the latter, a slicing algorithm based on MSDG is required to ensure precision. To gain the two benefits, we present a hybrid three-phase slicing algorithm based on TSDG and MSDG.

4.1 The Three-phase Slicing Algorithm

The hybrid slicing algorithm, as shown in Algorithm 1, has three phases. In the algorithm, E_{dd}, E_{cd}, E_{pi}, E_{po}, E_s and E_c denote data, control, parameter-in, parameter-out, summary and call dependence edge respectively. MS(s_y) is the set of M-S pair in which statement component of each element is s_y. In the first phase, TSDG is traversed only in the calling procedures of slicing criteria until nodes in the procedures required to be inlined are reached. These last visited nodes are recorded in W_2 for the second phase. In the second phase, MSDG is traversed from the nodes whose statement components are recorded in W_2. All calling procedures of each traversed node are required to be considered in the first phase. For all nodes in W_2, traversing the calling procedures of each node continues in the second phase because the set MS(s_y) contains all instances of s_y as a result of inlining. In the last phase, TSDG is traversed only in the called procedures from the nodes in work list W_3, which records formal parameter nodes generated in the previous two phases.

In the algorithm, if the slice is relevant to the procedures required to be inlined, context-insensitive problem is tackled because the procedures involved in interactions are really inlined. If the slice is only relevant to the procedures not required to be inlined, slicing processes includes only the first and third phase like sequential programs [4]. Furthermore, due to transitive property in MSDG, the algorithm can solve the three imprecision problems and a more precise slice is obtained.

Now compute the slice of S_7 in the program in Fig.1. As S_7 is an interaction statement, the MSDG is traversed first from <m_4, S_7> and <m_7, S_7>. The visited nodes are grey shaded in Fig.5. A_1 is an additional node, which is mapped to formal parameter node F_2. We then traverse TSDG from F_2 and the visited nodes are {S_9, S_{10}}. Removing the states of these nodes in three phases, we get

$Slice(S_7)=\{S_7$, start t_0, start t_1, start t_2, cobegin, $S_{12}, S_{11}, S_6, S_4, S_3, S_8, S_9, S_{10}\}$. S_5 and S_1 are not added to the slice. If we observe $<m_1, S_5> \rightarrow \underline{<m_2, S_8>,}$ $<m_0, S_1> \rightarrow \underline{<m_1, S_8>,}$ $<m_2, S_{12}> \rightarrow \underline{<m_4, S_8>} \rightarrow <m_7, S_7>$, we can find that distinguishing S_8 in m_1, m_2 and m_4 cut the intransitive dependence $S_5 \rightarrow S_8 \rightarrow S_7$ and $S_1 \rightarrow S_8 \rightarrow S_7$ effectively, then solve the intransitivity problem caused by variable redefinition.

Let CP be a concurrent program with *n* statements, *p* threads, *c* interaction statements and *k* maximum procedure call depth. The maximum instance of interaction statements is c^k. The complexity of the slicing algorithm in the second phase is $O(n^2(c^k/p)^{2p})$ in the worst case. The complexity of the first and third phase is $O(n^2)$. So the complexity of three-phase slicing algorithm is $O(n^2(c^k/p)^{2p})$.

```
Input: the slicing criteria s, the TSDG and the MSDG
Output: the slice Slice for s
Initialization:
    $W_1$=Slice={s}, $W_2$=$W_3$=$\varnothing$;
//Phase 1, traverse TSDG in calling procedures
while $W_1 \neq \varnothing$ do
    remove next element $s_x$ from $W_1$;
    if $s_x$ is a node in procedures required to be inlined
        $W_2=W_2 \cup MS(s_x)$;
    else
        for each edge $(s_y, s_x) \in E_{dd} \cup E_{cd} \cup E_{pi} \cup E_s \cup E_c$ in TSDG
            if $s_y$ has not been visited
                mark $s_y$ with visited;
                $W_1=W1 \cup \{ s_y \}$, Slice=Slice$\cup \{ s_y \}$;
        for each edge $(s_y, s_x) \in E_{po}$ in TSDG
            if $s_y$ has not been visited
                mark $s_y$ with visited;
                $W_3=W_3 \cup \{ s_y \}$, Slice=Slice$\cup \{ s_y \}$;
end while;
//Phase 2, traverse MSDG
while $W_2 \neq \varnothing$ do
    remove next element $\Lambda$ from $W_2$;
    for each edge $(\Lambda', \Lambda)$ in MSDG
            if $\Lambda'$ has not been visited
            mark $\Lambda'$ with visited;
            $W_2=W_2 \cup \{\Lambda' \}$, Slice=Slice$\cup \{s(\Lambda')\}$;
            if $s(\Lambda')$ is an additional actual parameter node on x
                $W_3= W_3 \cup \{F(x) \}$;
                // F(x) is the formal parameter node of x
end while;
//Phase3, traverse TSDG in called procedures
while $W_3 \neq \varnothing$ do
    remove next element $s_x$ from $W_3$;
    for each edge $(s_y, s_x) \in E_{dd} \cup E_{cd} \cup E_{po} \cup E_s$ in TSDG
            if $s_y$ has not been visited
                mark $s_y$ with visited;
                $W_3=W_3 \cup \{ s_y \}$, Slice=Slice$\cup \{ s_y \}$;
 end while;
```

Algorithm 1 The Three-Phase Slicing Algorithm

4.2 Optimization And Experimental Evaluation

Reachability analysis linearizes concurrently executing statements by interleaving, which makes the complexity exponential. Actually, the behaviors of concurrent programs have the property of partial order, i.e., some different interleavings are equivalent. Only one of them is chosen for analysis to enhance the efficiency. With the technique of partial order, we can incompletely explore the state space and generate a reduced reachability graph, which includes all representatives of concurrent programs.

We have developed a prototype slicer for the simple concurrency model mentioned in section 2. We extracted the basic program information with Codesurfer, a well-known slicing tool for C/C++ programs [11] . As Codesurfer cannot analyze concurrent support mentioned in section 2, the input program should be transformed properly into C/C++ programs.

We have implemented our hybrid slicing algorithm based on a complete reachability graph and a reduced reachability graph. To obtain a reduced reachability graph, we have used two partial-order reduction techniques: persistent sets and sleep sets. The computing algorithms of the two sets were presented by Overman[8]. To compare the performance of our algorithm to other high-precision slicing algorithms, we also implemented Nanda's two context-sensitive slicing algorithms based on TSDG, which could handle the imprecision problems caused by time travel and context-insensitivity, but not shared variable redefinition [3]. The complexity of the two algorithms is exponential in the number of threads. It is $O(n^2(n^k/p)^{2p})$ where *n* is the number of the program, *p* is the number of threads and *k* is the maximum procedure call depth.

TABLE 1 NODES AND EDGES IN DEPENDENCE GRAPHS

Algorithms	**2 3 4**						**5**	
	nodes	***edges***	***nodes edges nodes edges***				***nodes***	***edges***
N	22 32 29 52 36 63						43	87
NO	22 32 29 52 36 63						43	87
R	41	82	181 665 853			5844	---	---
RO	41	66	171 417 747			2989	3283	22968

TABLE 2 TIME TO BUILD THE DEPENDENCE GRAPHS AND SLICE

Algorithms	**2 3 4**						**5**	
	build	***slice build slice build slice***					***build***	***slice***
N	0.999	0.146	2.000 0.706 2.000 2.461				2.001	8.332
NO	0.999	0.109	2.000 0.362 2.000 1.197				2.001	5.119
R	1.031	0.016	1.336 0.043 14.20 0.318				---	---
RO	1.140	0.016	1.812 0.031 9.150 0.171				268.6	1.110

TABLE 3 AVERAGE NODES IN SLICE AND WORKLIST

Algorithms	**2 3**				**4**		**5**	
	slice	***work***	***slice***	***work***	***slice***	***work***	***slice***	***work***
N	10.5	42.8	15.5	159.8	20.5 579.8 24.5			2074
NO	10.5	34.3	15.5	145.8	20.5 302.3 24.5			1149
R 10.5		20.5	15.5	89.8	20.5	472.5	---	---
RO	10.5	18.5	15.5	66	20.5 280.5 24.5			1319

The algorithms have been tested on 2.66GHz Intel 2 Core Q8400 with 3GB RAM running Windows XP. We tried out the slicer on classical producer/consumer programs with an increasing number of thread instances to observe how the four algorithms handle the combinatorial explosion of threads states. In the following three tables, N and NO denote Nanda's context-sensitive algorithm without any optimization and with restrictive state tuple optimization respectively. R and RO denote our hybrid slicing algorithms based on complete and reduced reachability graph respectively.

The sizes of the dependence graphs and the time to build them have been described in Table 1 and Table 2 respectively. The time is the total time for compiling and constructing the dependence graphs. Cases that take more than an hour to analyze dependence or run have been marked with '---'. So no data is given for the algorithm M when there are five threads. Clearly, the size and complexity of MSDG are decreased with the reduction optimization. As the number of threads is increased, more benefits of reduction are gained. The time consumed in reduction is a big fraction of total time when there are less than 3 threads. Reduction makes the total time decrease when there more threads.

In Table 2, we give the average time of slicing four nodes that are relevant to the shared variables. In Table 3, we give the average size of the computed slices and work nodes. Table 3 shows that the four high-precision algorithms are able to compute the slices with the same size since there is no imprecision problem caused by shared variable redefinition in the producer/consumer program. Although it takes more time to build MSDG than TSDG, the time in slicing with R and RO is much less than N and NO respectively. RO has gained best performance in the four algorithms. Combinatorial explosion of state tuples is one major factor that affects the performance of the algorithms. The numbers of work nodes are directly affected by the combinatorial explosion of state tuples. Table 3 shows that the numbers of work nodes with N and NO are roughly same as R and RO respectively. N and NO is slower than R and RO since it takes a long time to check whether a given set of statements in a thread may belong to a realizable path in N and NO while it is not required in the algorithms of R and RO.

5. RELATED WORK

Most of the work on slicing interprocedural concurrent programs is based on TICFG or some graphs in the like [1, 2, 3, 5, 9, 10]. As mentioned in section 1, parallel flow is treated as branch flow in TICFG and then global data dependence analysis based on TICFG is impossible. Current precisely slicing methods mainly cope with the first two sources of imprecision, i.e., in transitivity due to time travel and context-insensitivity. The precision is up to interprocedural threaded witness.

Zhao's simple two-phase slicing algorithm and Nanda's iterated two-phase slicing algorithm are not precise because they cannot cope with the three imprecision problems[3,10]. Chen has presented a polynomial algorithm which in lines procedures involved in interactions and partially handled the intransitivity problem due to time travel[1]. Krinke and Nanda's context-sensitive algorithms completely cope with the imprecision problems due to time travel and context-insensitivity by checking whether a dependence sequence is in a valid execution or not[2, 4]. So the precision of slice is significantly improved. The complexity of the two algorithm is also exponential in the number of threads. Giffhorn gives empirical evaluations on their work [5, 9]. However, Krinke and Nanda's approaches do not handle the intransitivity problem due to variable redefinitions. In contrast, our three-phase slicing algorithm copes with the three imprecision problems. Therefore, we can obtain a more precise slice than the others. Furthermore, the efficiency of our algorithm is higher than Krinke's and Nanda's in the worst case as the number of interaction statements is much less than that of statements.

6. CONCLUSION

In this paper, we present a three-phase algorithm for slicing concurrent interprocedural programs. The procedures involved in thread interactions are inlined and analyzed globally based on threaded interaction reachability graphs, while other procedures are handled as sequential programs. Our algorithm takes into account the three imprecision problems: time travel, context-insensitivity, and shared variable redefinition in contrast to existing algorithms that could not handle the variable redefinition problem. As a result, a more precise slice is obtained with our algorithm. Preliminary experimental results show that our algorithm with the optimization of partial order technique is expected to have better performance than Nanda's classical high-precision slicing algorithms.

ACKNOWLEDGMENTS

This work is supported by the National Science Foundation of China under Grant No.F020509, No.60873049 and No.60703086.

REFERENCES

[1]Z.Q. Chen, B. W. Xu . Slicing concurrent Java programs. ACM SIGPLAN Notices, 36(4): 41-47, 2001.

[2]J. Krinke. Context-sensitive slicing of concurrent programs. In Proceedings of ACM SIGSOFT Symposium on Foundations of Software Engineering 2003 FSE'2003), 178-187, 2003.

[3]M.G. Nanda, S . Ramesh. Inter-procedural slicing of multithreaded programs with applications to Java. ACM Transaction on Programming Language Systems, 28(6): 1088–1144, 2006.

[4]S. Horwitz, T. Reps, D. Binkley . Inter-procedural slicing using dependence graphs. ACM Transaction on Programming Language System, 12(1): 26-60, 1990.

[5]D. Giffhorn, C. Hammer. Precise slicing of concurrent programs. Automated Software Engineering, 16(2): 197-234, 2009.

[6]M. Pezze, R .N. Taylor, M . Young. Graph models for reachability analysis of concurrent programs. ACM Transaction on Software Engineering and Methodology, 4(2): 171-213. 1995.

[7]Xiaofang Qi, Xiaoyu Zhou, Xiaojing Xu, Yingzhou Zhang. Slicing Concurrent Programs Based on Program Reachability Graph. The 10th International Conference on Quality Software (QSIC2010).

[8]Godefroid P. Partial-Order Methods for the Verification of Concurrent Systems. Ph.D. thesis, University De Leige, Montefiore, Beljium, 1994.

[9]Giffhorn D. . Advanced chopping of sequential and concurrent programs. Software Quality Journal, 2011, 19:239-294.

[10] Zhao, J. J. Multithreaded dependence graphs for concurrent Java programs. In: Proc. of International Symposium on Software Engineering for Parallel and Distributed Systems, IEEE CS press, 1999, 13-23.

[11] http://www.grammatech.com/products/codesurfer/overview.html.

A Usage-Based Unified Resource Model

Yves Wautelet
Hogeschool-Universiteit Brussel
Stormstraat, 2,
1000 Brussels, Belgium
yves.wautelet@hubrussel.be

Samedi Heng
Université catholique de Louvain
Place des Doyens, 1,
1348 Louvain-la-Neuve, Belgium
samedi.heng@uclouvain.be

Manuel Kolp
Université catholique de Louvain
Place des Doyens, 1,
1348 Louvain-la-Neuve, Belgium
manuel.kolp@uclouvain.be

Abstract—**Most of the time engineering methodologies focus on the modeling of functional and non-functional requirements of the system-to-be with no or poor representation of resource elements. Resources are nevertheless central in most industrial domains and do have an important impact onto the performance and feasability of software requirements. That is why we propose, in this paper, an ontology for resource representation centered on its *Usage* i.e., the concepts of *functionality* for *Resource Objects* and *competency* for *Resource Agents*. This ontology does not tackle the particular problem of service level agreement which is a complementary dimension but rather focuses on how resources can be represented and handled at runtime. Heterogeneous resources can thus be represented in a unified manner within the context of "resource-intensive" domains where information systems are developed. Moreover, it could also be used to develop a specific heterogeneous monitoring system with, for instance, the agent technology so that it acts for interoperability purposes. The ontology proposal is applied on a case study in the industrial context of a steel industry, namely CARSID where lots of resources are collaborating to achieve defined services. The purpose is to show the applicability of the ontology in an industrial context where resources play a central role for the information system.**

I. Introduction

Poor focus on detailed representation of complex resources within software development methodologies results omitting taking into account important aspects of the software problem that could be represented using a common pattern to ease the work of both the software analyst and designer. We consequently propose in this paper a common ontology allowing to model each type of resource in a unified and standardized manner, i.e. independent of their *type*, *domain* and *interface*. Indeed, resources are useless when they do not provide an added value for a functional or operational execution so that the ontology is driven by the *Resource Usage* i.e., *functionalities* for *Resource Objects* and *competencies* for *Resource Agents*. The ontology proposes to centralize resources' offer and demand through their *Usages*. More specifically, we present an ontology for resource representation and handling that can be used as a general reference into resource intensive domain where information systems are developed. The contribution is located at design stage so that resources identified during analysis can be mapped into functionality and competency providers at runtime for dynamic resource allocation and reservation. In that perspective, a generic multi-agent system for resource-monitoring could be easily build up on its basis. This, within a functional software application, furnishes an intelligent – i.e., optimized on the basis of the available information – resource allocation mechanism. This ontology represents a contribution at design stage but is intended to be used in the context of a larger development methodology [1], [2] within the engineering of software services. Further developments as well as advanced strategies for resource monitoring as proposed in [3] are outside the scope of the paper and are left for future perspectives.

The rest of the paper is structured as follows. Section II discusses the structural choices of the ontology notably through the concepts of functionality and competency; the distinction between resources and actors and, finally, resource composition and hierarchy. Section III overviews the conceptual model itself while Section IV is devoted to a case study in the steel making industrial environment of CARSID. Finally, Section V briefly depicts related work and Section VI concludes the paper.

II. Modeling Resources

This section discusses the concepts of funtionality and competency, distinguishes resources from actors and discusses resource composition and hierarchy.

A. Modeling Resources with their Use

There are numerous definitions of the concept of *resource*. The Merriam-Webster dictionary refers it as *a person, asset, material, or capital which can be used to accomplish a goal*. While remaining basic, this definition has the interest of identifying the fact that a resource is not a process (something functional/operational) but should be used for the proper achievement (the goal) of such process through action(s). This consequently leads to the intuition that resources do own particular skills that are *required* in a functional context (for the realization of processes or services). Within a pool of resources, the requesting service is thus only interested by resources helping him achieving its functional requirements. Nevertheless, defining their type, resources are *inanimate* and do have a functional utility (this is the case of *Resource Objects*) or are able to *behave* (this is the case for *Resource Agents*). Resources' offer and demand can consequently not be centralized into a single concept so that *Resource Objects* are *functionality* providers while *Resource Agents* are *competency* providers. Resources *offer* functionalities and competencies while services *demand* them.

A *functionality* is the ability of a resource to be used for achieving a specific purpose. In this sense the resource owning the functionality is passively used during a process execution so that it is specific to *Resource Objects*. Moreover, the concept of *competency* used in human capital and business resources management allows stakeholders and stockholders to reach resources unification and integration to optimize the capacity needed to achieve the vision, mission and objectives of their organization.

Following [4], a competency is *an aptitude to know, to know how and to behave* while [5] defines competencies as *statements that someone, and more generally some resource, can demonstrate the application of a generic skill to some knowledge, with a certain degree of performance*. In other words, competencies are observable skills, knowledge or know-how defined in terms of behaviors and processes required for performing job and business activities in a successful way. They are consequently owned by resources able to behave i.e. *Resource Agents* (see section II for a complete justification and definition). Knowledge and learning are specific to human/organizational resources and artificial intelligence systems. Behavior can be considered as the ability owned by *Resource Agents*, making them useful in an operational context.

In the context of the proposed ontology, resources are thus components encapsulating functionalities and competencies. What fundamentally distinguishes those two concepts is that:

- a functionality is transparent in the sense that the environment is perfectly aware of what the *Resource Object* can deliver at defined quality level. The functional offer of Resource Objects can be said to be stable since they cannot acquire more functionalities - or lose existing functionalities - unless being transformed (or degraded);
- a competency is partially hidden since initial evaluation is necessary to assess the quality level at which a *Resource Agent* can deliver it; it is also evolutionary since it can be delivered at higher quality level later into the system life cycle thanks to the inherent ability to learn from *Resource Agents*.

Resources can nevertheless not always lead to an absolute and successful achievement of the functionalities or competencies they advertise so that a quality level should be associated to each of them. Competencies are indeed not neutral and are furnished following a quality level so that without being perfectly rational, their realization is non-deterministic and must be estimated. The probabilistic measure is, in our model, encapsulated in the concept of quality level; we indeed consider the probability as being an aspect of the quality ontology presented hereby. A *Resource Agent* providing a competency with a lower realization rate will be attributed a lower quality level following the particular quality ontology. Quality ontologies are domain specific and such a characterization is positioned into the proposed resource ontology but not instantiated (see Section III). Similarly, furnishing competencies sometimes requires *resource composition* – i.e., the use of a defined set of resources configured in a defined way – so that resource hierarchization is required (as detailed in Section III-C). Quality ontology proposal we can link this work to are provided in [6].

Moreover, competencies can also be *acquired* in the case of an individual or an organization through formal or non-formal procedures. This suggests *Resource Agents* evaluation through a formal process. For example, an ISO accreditation is, for an organization (which is here a *Resource Agent*) receiving it, a certification to have competencies to furnish a product or a service at some defined minimal quality level. In the same way, the follow-up and successful achievement of a study program in a given school or university is a process for a human (which is here a *Resource Agent*) to acquire and accreditate (through the obtention of a diploma) defined competencies at a specified quality level. The processes of acquisition and assessment of competencies within particular (or a particular set of) resources is outside the scope of this paper. Functionality and competency ontologies are domain specific and such a characterization is positioned into the proposed resource ontology but not instantiated (see Section III). Specific competency ontology proposals we can link this work to are provided into [4], [5].

B. Resources, Actors and Services

In high level analysis formalisms such as the i* modeling framework ([7]), human or machine resources can be modeled as actors depending on each other involved in the achievement services; this vision assumes an **intentional dimension** and two parties, the service consumer and the service provider. With respect to the service ontology presented in [8], this is the *Service Commitment* level (prescriptive level) but the resource ontology we propose in this paper is (also with respect to [8]) at *Service Process* (design and implementation levels). Following [9], Services are "high-level" elements i.e., coarse-grained granules of information that encapsulate an entire or a set of business processes. That is consequently the level where service consumers and service providers are specified and, since the proposed ontology is at service process level we do not specify them into the contractual aspects of resource reservation and use (see section III-B).

III. An Ontology for Resource Representation

We define in this section a conceptual model for resource representation. In this perspective, Figure 1 depicts the relevant concepts and their dependencies using a class model [10].

A. Basic Concepts

Services require, for their proper execution (or realization), a series of *Functionalities* and *Competencies* at a defined *QualityLevel* furnished by one or more *Resources*. The use of a *Resource* by a *Service* is only conceivable by setting up a *Contract* for a defined *period of time* and defined *QualityLevel* and *Cost*. All of the system *Resources* are kept in the *ResourceList* while their "real-time" availability is given

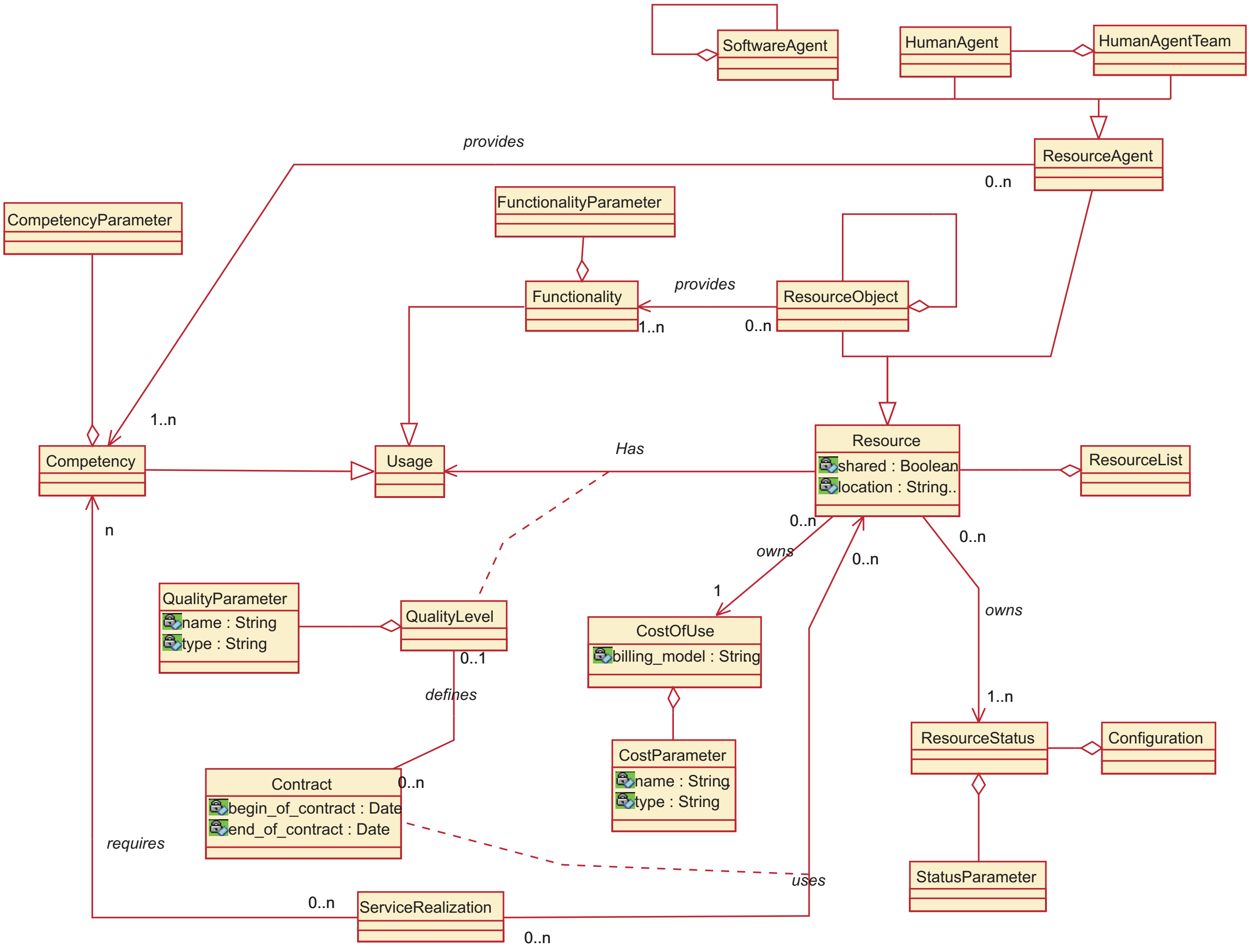


Fig. 1. An Ontology for Resource Representation.

through their *Status*. Those concepts are formally defined hereafter.

A tuple $\langle\{(us_i, q^r_{us_i}), \ldots, (us_{i+m}, q^r_{us_{i+m}})\}, Res^r\rangle$ is a *Resource* r, where us_i is a *Usage*. A resource furnishes *Usages* (which can be *Functionalities* or *Competencies* required by *Services* to contribute to their fulfilment) at *Quality Level* $q^r_{us_i}$ ($q^r_{us_i}$ follows a particular quality ontology). Res^r is assumed to contain all additional properties of the resource not relevant for the present discussion, yet necessary when implementing the solution. Resources belong to the set $\mathbb{RL}$ ($\mathbb{RL}$ stands for *Resource List*).

More precisely, resources deliver *Functionalities* if they are *Resource Objects* or *Capabilities* if they are *Resource Agents* (see Section II for a complete justification). A *Usage* is a generalization of *Functionality* and *Competency* and is compulsorily one or the other. That concept allows to tackle the problem in a generic manner. The reader should note that the distinction between *Resource Agent* and *Resource Object* is not in the sense of agent or object-orientation but in the sense of *Resource Objects* being inanimate; so even if the program is object-oriented, it is still classified as *Software Agent*, since what matters is modeling behavior.

The model is independent of any *Functionality* and *Competency* ontologies; it can for example be used through the competency ontologies defined in [4], [5]. The *Functionality-Parameter* and *CompetencyParamater* classes are assumed to contain all of the dimensions of these custom ontologies.

As previously discussed, Services are considered here as coarse-grained granules of information that encapsulate an entire or a set of business processes. With respect to the ontology proposed in [8], services are thus viewed at the level of *Service Processes* - i.e., design and implementation levels so that services' format and content depend on the implementation paradigm being used (procedural, object, agent, etc.). For that purpose, they are not further specified (e.g., in terms of pre/post condition and invariants) and defined here. Indeed, for example, within the i* framework, a *Service* can be a *Goal* or *Task* (while functionalities and competencies are at *Capability* level); in UML/J2EE technology [11] it can be a *Use Case* (while functionalities and competencies are at *Method Call* level). Further functional considerations of the *Service* are outside the scope of this ontology which tackles a lower level of abstraction (granularity).

The set of resources $\mathbb{RL}$ (*ResourceList*) can be used as yellow pages refeering the resources present into the system; their status – i.e., information about their availability at a given moment of time – must then be evaluated using the characterization that follows.

A *ResourceStatus*, $status_i$, is $\langle status_i^{pre}, \tau_i, status_i^{post} \rangle$, where $status_i^{pre}$ describes the preconditions for establishing a contract (following a particular ResourceStatus ontology), τ_i is a specification (following a particular API) on how the resource can be interfaced with and $status_i^{post}$ describes the postconditions to properly end up the contract (following the same ResourceStatus ontology).

The *ResourceStatus* ontology can be customely implemented within the model. The *StatusParameter* class is assumed to contain all of the dimensions of this custom ontology. A *Configuration* is a subset of *ResourceStatus* elements; this concept can be used to predefine quality levels before runtime for optimization purpose in the form of a "caching" system.

Quality and cost ontologies can be implemented within the model on a case by case basis. The *QualityParameter* and *CostParameter* classes are assumed to contain all of the dimensions of these custom ontologies.

B. Contractual Aspects

As previously evoked, the resource ontology assumes a "higher-level" (service) dimension. A resource *in use* within the realization of a defined service can, for *a time shift* be not available and has to be invoiced to the consuming process (or its cost center). Consequently such a transaction has to be tracked by a formal element managing the resource reservation and invoicing aspects: the *contract*. Finally, the use of a resource has a cost. Cost ontologies are domain specific and such a characterization is positioned here without being instantiated. A specific cost ontology proposal we can link this work to is provided into [12].

A *Contract* $\hat{cont}_i$ is $\langle Serv^s, us_i, res_i, res_i^{Qual}, res_i^{Cost}, res_i^{BeginTime}, res_i^{EndTime} \rangle$ associates a *Service* s requiring the *Usage* us_i (*Functionality* or *Competency*) provided by the selected *Resource* res_i, where:

- res_i^{Qual} specifies the minimal ensured quality level. Its definition follows a particular quality ontology. Whatever the specific quality ontology, expected qualities are likely to be specified as (at least) $res_i^{Qual} = \langle (p_1, d_1, v_1, u_1), \ldots, (p_r, d_r, v_r, u_r) \rangle$, where:
  - p_k is the name of the *QualityParameter*;
  - d_k gives the type of the parameter (e.g., *nominal*, *ordinal*, *interval*, *ratio*, ...);
  - v_k is the set of desired values of the parameter, or the constraint $<, \leq, =, \geq, >$ on the value of the parameter;
  - u_k is the unit of the property value.
- res_i^{Cost} specifies the contractual *cost of use* with respect to a particular cost ontology. Whatever the specific cost ontology, expected costs are likely to be specified as (at least) $res_i^{Cost} = \langle (n_1, t_1, b_1), \ldots, (n_r, t_r, b_r) \rangle$, where:
  - n_k is the name of the *CostParameter*;
  - t_k gives the type of the parameter (e.g., *nominal*, *ordinal*, *interval*, *ratio*, ...);
  - b_k is the billing model (*pay per use*, *subscription*,...).
- $res_i^{BeginTime}$ is the time the reservation of the resource starts for the particular contract;
- $res_i^{EndTime}$ is the time the reservation of the resource ends for the particular contract.

C. Resource Composition and Hierarchy

The ontology assumes a resources hierarchy in the sense that a *Resource Object* realizing a functionality or a *Software Agent* (which is a specialization of *Resource Agent*) realizing a capability can either be an *assembly* – i.e., a resource made of other resources advertising competencies – or be *atomic* – i.e., not made of other resources advertising competencies. This is materialized into the ontology by the composition association on the *Resource Object* and *Software Agent* concepts themselves (see Section III). Similarly, *Human Agents* cannot be compositions of themselves but a *HumanAgentsTeam* is a composition of several *HumanAgents*; both of these concepts are specializations of *Resource Agents* so that a composition link joins these two concepts into the meta-model.

IV. Case Study: Coking Process

The ontology is applied in this section within the realization of a particular service in the context of a supporting information system for a coking plant.

A. Context

CARSID, a steel production company located in the Walloon region, is developing a production management software system for its coking plant. The aim is to provide users, engineers and workers with tools for information management, process automation, resource and production planning, decision making, etc. Coking is the process of heating coal into ovens to transform it into coke and remove volatile matter from it. Metallurgical Coke is used as a fuel and reducing agent, principally by the blast furnaces, in the production of iron, steel, ferro-alloys, elemental phosphorus, calcium carbide and numerous other production processes. It is also used to produce carbon electrodes and to agglomerate sinter and iron ore pellets. The production of coke is only one step in the steel making process but details about the other phases of the production process are not necessary to understand the case study. For more information about this CARSID project, we refer the reader to [1].

B. Applying the Ontology

The illustration depicted in this section is in the context of the development of a software application made of several services; *Pushing* is the *Service Process* realization we will be concerned with here. Indeed, other services are not essential for the present discussion and the reader should just keep in mind that the description that will be given here is part of a much larger set of developments.

In a few word, the *Pushing* service represents the process by which the *Pusher Machine* pushes the red-hot *Coke* out of the *Oven* through the *Coke Guide* into the *Coke Car*.

Typically, during a coke pushing process, the pusher machine situated at the front side of the ovens battery, removes the front door, pushes the red-hot coke into the coke car situated underneath the oven; the coke guide is used to remove the oven back door and to correctly guide the red-hot coke coming out of the oven into the coke car. The upper part of the machine runs a mechanic arm pushing slowly the coke out of the oven. The coke car is positioned behind the open oven receiving the red-hot coke pushed out of the oven by the pusher machine.

A formal set of functionalities and competencies is required to define this industrial process as reported in the *Fulfillment* statement of the following specification:

Service Realization Pushing
Attribute input: material in oven
gc: GuideCoke
cc: CokeCar
Fulfillment cooking(input, o) $\wedge$ align(pm, gc, cc, position(o)) $\wedge$
guide(input) $\wedge$ collect(input) $\rightarrow \diamond$ cokeMaterial(cm) $\wedge$
emptyOven(o)

More precisely, *cooking(input, o)* represents the capability of the oven o to cook coke at a temperature between 1200 and 1350 °C during 16 to 20 hours to transform it into red-hot coke. Note that when this red-hot coke will be cold-down (through a passage at the quenching tower) it will be transformed into (metallurgical) coke, the finished product of the coking plant. Aligning the pushing machine, the guide coke and the coke car with the oven is a necessity to proceed with pushing the coke out of the oven, the reason why the functionality *align(pm, gc, cc, position(o))* is then specified. When proceeding with the pushing, the red-hot coke is guided by the coke guide (*guide(input)* capability) and collected into the coke car (*collect(input)* capability).

The list of resources used for achieving the required functionalities and competencies is given hereafter.

Resource Oven o
Type : Resource Agent
Has ovenId: O123, currentTemperature: 1250, maxCapacity: 10,
openDoorStatus: closed, exitDoorStatus: closed,
cookingStatus: active, cookingBeginTime: 18:00;
Provides evaluateCooking(input, this);

Resource PushingMachine pm
Type : Resource Agent
Has pmId: 1, maxPower: 10, status: available;
Provides align(this, position), pushing(input);

Resource cokeMaterial cm
Type : Resource Object
Has cokeId: lot1333, status: cooking;
Provides redHotCoke(input);

Resource guideCoke gc
Type : Resource Agent
Has gcId: gc1, status: available;
Provides align(this, position), guide(input);

Resource cokeCar cc
Type : Resource Agent
Has ccId: cc1, status: available;
Provides align(this, position), collect(input);

At runtime, the realization of the *Pushing* service implies the fulfillment of the *evaluateCooking(input, this)*, *align(pm, position(o))*, *align(gc, position(o))*, *align(cc, position(o))* and *collect(input)* capabilities leading to the series of contracts depicted hereafter.

Contract EvaluateCooking123
Fulfills evaluateCooking(input, o);
Using oven o, cokeMaterial cm;
QualityLevel N/A;
Cost N/A;

Contract Align123
Fulfills align(pm, position(o)), align(gc, position(o)),
align(cc, position(o));
Using pushingMachine pm, guideCoke gc, cokeCar cc, oven o;
Has beginTime: Date, endtime: Date;
QualityLevel N/A;
Cost Energy(pm(pos(x),pos(o)), gc(pos(x),pos(o)),
cc(pos(x),pos(o)));

Contract PushCokeLot1333
Fulfills collect(input);
Using oven o, cokeMaterial cm;
Has beginTime: Date, endtime: Date;
QualityLevel N/A;
Cost Energy(push);

C. Lessons Learned and Ontology Contributions

The use of the resource ontology as a pattern as well as a structured approach for resource representation and tracing among the analysis and design steps within the CARSID case study has allowed us:

- to improve the structural complexity. Indeed, the current application design has been enhanced with respect to metrics as the ones defined in [13];

- to provide a unified catalogue of available resources in the form of a yellow page system;
- to redefine resource as runtime dynamic elements. Object-oriented development used a static and passive description of the resources;
- to better understand resource utilization through the ex-post study of the contracts by production engineers. This allows to identify possible bottlenecks and re-optimize the production planning for adequate resource utilization;
- to compute and store into the caching system pre-positions for devices as the pusher machine, coke car and guide coke. Since several combinations of instances of these machines are available, the process is non-trivial;
- to express the process from a resource-based point of view which allows focusing on interoperability and evolutionary aspects.

V. Related Work

Different papers have proposed models to represent and monitor resources. Even heterogeneous resources have been dealt with – for example in [14] – their evocation has been always in the context of hardware resources. Indeed, [15] goes beyond the stage of defining an ontology for heterogeneous resources representation by proposing a complete architecture as well as an underlying management dimension with the use of a semantic repository. The proposal nevertheless only focuses on the particular context of grid computing and consequently hardware resources so that it cannot be of any help in terms of general business modeling. We nevertheless could envisage to include "our" ontology into their contribution to develop a larger resource monitoring system. Similarly [16] goes further in the idea of developing semantic resource management and within their management using a scheduling method but also remains in the context of hardware for grid computing. As evoked in Section II, competency-based modeling for resource monitoring has been developed in [5] but in the specific context of human resources. Our ontological frame extends the competency concept to semantically encompass these two resource types. Conceptual models as i* [7] include the resource concept in their definition but do not define advanced frames for forward engineering such concepts at design stages; the ontological frame proposed here is intended to be extended fill this gap and explicitly define traceability between the business analysis and software design stages.

VI. Conclusion

The new hardware resource sharing paradigm and distant software execution devices such as netbooks or smartphones paves the way to new heterogeneous resource allocation requirements supported by active software. To address this problem at best, an ontology that can easily and flexibly be included in a classical software engineering process, such as [1], [2], to develop resource-aware software systems has been presented in this paper. Section II has overviewed the structural choices of the ontology including the concepts of functionality and competency; the distinction between resources and actors and, finally, resource composition and hierarchy. Section III has presented the conceptual model itself while Section IV has been devoted to a case study. Finally, Section V has briefly depicted related work.

The conceptual model in this research proposes to centralize resources' offer and demand through the concepts of functionality and competency. The ontology furnishes a common semantic for consumers (i.e. services) to rent resources that can themselves advertise their offer so that consumption contracts can be set up.

Future work includes extending the ontology with a process covering the whole software development life cycle from (agent-based) analysis to service level agreements. The development of a dynamic dimension allowing to document and implement a multi-agent system resources management as well as the realization on a case study in the field of outbound logistics is currently under progress.

References

[1] Y. Wautelet and M. Kolp, "Goal driven iterative software project management," in *ICSOFT (2)*, M. J. E. Cuaresma, B. Shishkov, and J. Cordeiro, Eds. SciTePress, 2011, pp. 44–53.

[2] Y. Wautelet, S. Kiv, and M. Kolp, "An iterative process for component-based software development centered on agents," *T. Computational Collective Intelligence*, vol. 5, pp. 41–65, 2011.

[3] C. M. Jenkins and S. V. Rice, "Resource modeling in discrete-event simulation environments: A fifty-year perspective," in *Winter Simulation Conference*, 2009, pp. 755–766.

[4] M. Harzallah and F. Vernadat, "It-based competency modeling and management: from theory to practice in enterprise engineering and operations," *Comput. Ind.*, vol. 48, pp. 157–179, June 2002.

[5] G. Paquette, "An ontology and a software framework for competency modeling and management," *Educational Technology & Society*, vol. 10, no. 3, pp. 1–21, 2007.

[6] I. Jureta, C. Herssens, and S. Faulkner, "A comprehensive quality model for service-oriented systems," *Software Quality Journal*, vol. 17, no. 1, pp. 65–98, 2009.

[7] E. Yu, P. Giorgini, N. Maiden, J. Mylopoulos, and S. Fickas, *Social Modeling for Requirements Engineering*. MIT Press, 2011.

[8] R. Ferrario and N. Guarino, "Towards an ontological foundation for services science," in *FIS*, 2008, pp. 152–169.

[9] R. Haesen, M. Snoeck, W. Lemahieu, and S. Poelmans, "On the definition of service granularity and its architectural impact," in *CAiSE*, ser. Lecture Notes in Computer Science, Z. Bellahsene and M. Léonard, Eds., vol. 5074. Springer, 2008, pp. 375–389.

[10] OMG, "Omg unified modeling language (omg uml). version 2.4," Object Management Group, Tech. Rep., 2011.

[11] K. Ahmed and C. Umrysh, *Developing Enterprise Java Applications with J2EE(TM) and UML*. Addison-Wesley, 2001.

[12] K. Tham, M. S. Fox, and M. Gruninger, "A cost ontology for enterprise modelling," in *Proceedings of Third Workshop on Enabling Technologies-Infrastructures for Collaborative Enterprises*. IEEE Computer Society, 1994, pp. 197–210.

[13] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Trans. Software Eng.*, vol. 20, no. 6, pp. 476–493, 1994.

[14] Y. Yu and H. Jin, "An ontology-based host resources monitoring approach in grid environment," in *WAIM*, 2005, pp. 834–839.

[15] T. S. Somasundaram, R. A. Balachandar, V. Kandasamy, R. Buyya, R. Raman, N. Mohanram, and S. Varun, "Semantic-based grid resource discovery and its integration with the grid service broker," in *In ADCOM 2006: Proceedings of 14th International Conference on Advanced Computing & Communications*, 2006, pp. 84–89.

[16] A. C. Vidal, F. J. da Silva e Silva, S. T. Kofuji, and F. Kon, "Semanticsbased grid resource management," *MGC '07 Proceedings of the 5th international workshop on Middleware for grid computing*, 2007.

Petri Net Modeling of Application Server Performance for Web Services

M. Rahmani, A. Azadmanesh, H. Siy
College of Information Science & Technology
University of Nebraska-Omaha
Omaha, NE 68106.
Tel: +1(402) 554-2084
{crahmani, azad, hsiy}@unomaha.edu

***Abstract*—A study of failure rates of a web service that is deployed in a service-oriented architecture is presented. The study focuses on the HTTP requests that are rejected by the application server. The rejections may be caused by system overloading or mismanagement of configuration parameters. An analytical model and a Stochastic Activity Network (SAN) model are developed to predict the number of such failures. The performance of the models is compared against the experimental results in a LAN environment that is used as a test-bed. The models utilize the parameters extracted from the empirical testing such as the average response time and arrival rate of the web service requests. The accuracy of the SAN model suggests that the model can be beneficial to predict the rejection rate of web services and to better understand the application server performance for those cases that are difficult to replicate in a field study.**

Keywords- Application Server; Simulation Model; Service Oriented Architecture; Web Service; Petri Net; Load Testing

I. Introduction and Related work

Web services, such as electronic shopping, online auction, and banking services, have permeated our daily lives due to the ease of use, flexibility and reduction of cost in providing the services [1]. T hey are generically seen by many as applications that can be accessed over the Internet. A more precise definition of web services, provided by the World Wide Web Consortium (W3C) [2], defines it as a software system designed to support interoperable machine-to-machine interaction over a network. Web services are generally enabled through an application server software that communicates with a web server and a database server.

With the prevalence of web services in real-time and critical software domains, high level of reliability and performance are expected from them. As such, a number of approaches to reliability prediction and performance evaluations have been attempted. These approaches are often not new and have been applied to reliability and performance analysis of component-based software systems. Some of the most common approaches are based on Markov Chains, Tree-based, and Petri net models [3,4,5,6,7]. The majority of research works is theoretical with less emphasis on experimental analysis to support the theoretical results [4,8,9,10,11]. For example, Zhong and Qi [8] propose a Petri net model of performance and dependability of web service composition. However, the paper emphasizes on theoretical aspect of research and does not include any experimental analysis. In [9], authors presented a simulation-based model for performance analysis and evaluation of application servers using Petri nets. However, the numerical illustration of the presented simulation model is based on hypothetical input and not real experiments.

Some of the studies that use a combination of theoretical and experimental analyses include [12,13,14,15,16]. Xiao and Dohi [12] focused on the relationship between the Apache server error rate and system's performance and developed a probability model to describe this relationship. In [13], authors focused on a performance evaluation model of a web server system, which is based on a queuing model and evaluated the effectiveness of the model through the experiments in the lab environment. In [14], the authors proposed another queuing model to evaluate the performance parameters of a web server such as the response time and the blocking probability. The authors in [15] evaluate the response time and throughput of web services by collecting and analyzing large sets of data from geographically distributed locations. Goseva et al. [16] presented an empirical as well as theoretical study of architecture-based software reliability on a la rge open source application with 350,000 lines of C code. They emphasized on theoretical and experimental results on a large scale field study to test and analyze the architecture-based software reliability.

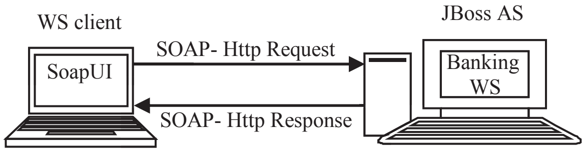
This paper is focused on the performance modeling of the JBoss application server [17] with respect to response time, throughput, and rejection (blocking) rate of web service requests. The study 1) introduces an analytical method to measure the rejection rate, 2) est ablishes an experimental environment to collect data by generating controlled loads of web service requests to JBoss, and 3) injects the experimental results into a Petri net model that simulates the JBoss performance behavior. In general, the main contribution of the paper is the measurement and comparison of web service rejection rate from analytical, empirical and simulation (Petri net) points of view. Other than the experimental and theoretical approaches, this study is dif ferent from many other relevant research works in that the focus is geared toward understanding the effect of configuration parameters such as the maximum HTTP thread pool and the queue size in the application server. Petri nets are chosen for this study because of the flexibility they provide to model systems with multi-threaded executions. In comparison, analytical approaches such as Markov models are difficult to express multi-threaded systems, at least because of the state-space explosion problem, as each Petri net marking becomes a single state in the Markov model. As Markov models can be seen as a finite state machine, they are more suitable for single-threaded systems.

The paper is organized as follows. Section II presents a) the description of experiments, b) an analytical model, and c) the Petri net model for HTTP request rejection rate in JBoss. Section III focuses on experimental and analytical results. Section IV offers some possible directions to the future research and finally section V contains some concluding remarks.

II. DESCRIPTION OF EXPERIMENT

A. Experimental Environment

The experimental environment consists of two hosts (client and server) remotely located from each other in a LAN environment. The host server on which JBoss is running is excluded from running other tasks to ensure the consistency of data sets collected. The Duke's Bank application [18] is transformed into a web service and deployed on JBoss. JBoss uses the Apache Tomcat as its default web server. The client host generates service requests to JBoss. The bandwidth of the LAN is shared with other users not relevant to this experiment. Tools like Wireshark [19] and Ping are used to measure the round trip delay (RTD), excluding the time spent in the hosts. In comparison to the time spent in the server, the RTD is observed to be so negligible that it is ignored in the experimental analyses. The system structure is illustrated in Fig. 1.



	Client	*Server*
OS	Windows 7	Windows Vista
Processor	1.67 GHz	2.66 GHz
RAM	1.00 GB	2.00 GB
Software	SoapUI (3.6.1)	JBoss AS (4.2.2)

Figure 1. The experimental setup.

SoapUI [20] is used to generate controlled service request loads to the server. SoapUI is an open source testing tool solution for service-oriented architectures. With a graphical user interface, SoapUI is capable of generating and mocking service requests in SOAP format. There are two main parameters in SoapUI load testing tool that can be set to control the workload of the application server: number of threads representing the virtual users, and the number of requests (runs) generated per thread. For example, if the number of threads is set to 20 and number of runs per thread is set to 10, then there are 20 clients, each sending 10 SOAP requests for a total of 200 requests. SoapUI measures several performance parameters such as the average response time, *avg*, transactions per second, *tps*, and the number of transaction requests failed in the process, which is denoted as err. *Avg* is the average time difference between the times a request is sent out until the response is received. *Tps*, also called arrival rate, is the average number of requests generated by the clients per second.

The experiments have been conducted with sixteen different load configurations. For each load test, SoapUI returns the values for *avg*, *tps*, and *err*. Each test is repeated 5 times and the average of the returned values are calculated. The total time *T* for each load test configuration is computed as follows:

$$T = \frac{cnt}{tps} \tag{1}$$

where *cnt* is the total number of requests for each test. Consequently, the error and success rates for each load are computed by:

$$error_{rate} = \frac{err}{cnt} \tag{2}$$

$$success_{rate} = \frac{cnt - err}{cnt} = 1 - error_{rate} \tag{3}$$

The error reports generated by SoapUI consist of all types of potential errors that are generated by the network, application server, and web service itself. However, this study considers only the errors that are generated by JBoss when the HTTP requests are rejected. Therefore, $error_{rate}$ is treated as $rejection_{rate}$.

B. The Analytical Model

The actual rate of requests rejected can be obtained from (2). This rate can be estimated in a different way. Recall that *avg* is the average response time for one request. Therefore, the service rate is 1/*avg*. In order for JBoss not to reject any incoming request, it should be able to use enough threads to keep up with the arrival rate. Thus, JBoss will reject no requests if the following holds:

$$\frac{1}{avg} \times threads \geq tps \Rightarrow \frac{1}{avg} \times threads - tps \geq 0$$

Otherwise some requests will be rejected. Thus, number of requests rejected per unit of time, i.e. $rejection_{rate}$, will be:

$$(tps - \frac{1}{avg} \times threads)$$

Consequently, the total number of requests rejected is:

$$rejection_{est} = (tps - \frac{1}{avg} \times threads) \times T \tag{4}$$

The accuracy of (4) depends on *threads*, which will be referred to as *threshold*. This is because any value higher than the threshold value underestimates and any value lower than the threshold overestimates the number of rejections. Thus, the following will be used instead of (4).

$$rejection_{est} = \left(tps - \frac{1}{avg} \times threshold\right) \times T \quad (5)$$

Threshold is computed from the experimental analyses. Theoretically, when the configuration parameters *maxThread* and *acceptCount* in *server.xml* file are set to 250 and 100 respectively, it is expe cted that JBoss handle 350 requests (250 in the thread pool and 100 in the queuing system). However, in real world experiments, there are many factors such as m emory, processor and type of operating system that may affect the actual number of threads that can be devoted to requests. More importantly, the response time evaluated by SoapUI includes the time spent in the queuing system and the time actually spent on servicing the request. With this experiment setup in the lab and by running different tests, *threshold* is estimated to be around 315. This is the threshold number of threads in JBoss that leads to performance results from the test cases performed on SoapUI.

It is expected that the response time, *avg*, to increase as the number of requests increases. Fig. 2 shows the *avg* numbers for different virtual users in the real experiments, where each user sends 10 requests. The figure exhibits that the response time increases up to a point, levels-off for a moderate range of requests, and then increases again.

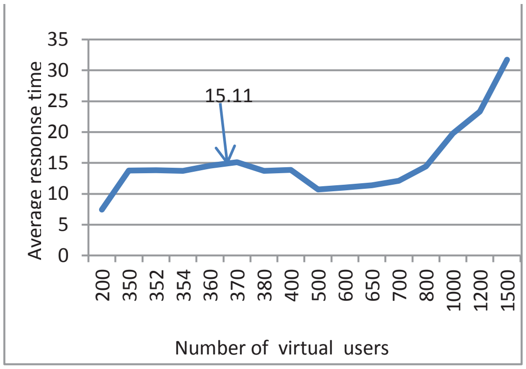


Figure 2. Average response time based on number of virtual users.

The reduction in response time beyond 370 up to around 500 users is counterintuitive because as there are more requests the response time ought to increase instead. During this period, Fig. 3 indicates that the rejection rate is increased. Toward the end, Fig. 2 further shows sharp increase in the average response time while the rejection rate on average in Fig. 3 stays the same. This paradoxical behavior has two reasons. The first reason is that SoapUI uses the total requests in calculating the average response time, regardless of whether a request is successful or rejected. It is f or this reason that the average response time of 15.11 in Fig. 2 can be interpreted as the true response time because at about 370 users the number of rejections is low or almost nonexistent. The second reason is contributed to JBoss, which can be explained as follows. From 370 to around 500 users, the server is using the resources that are already setup and activated, so the system can reallocate them to other requests, such as the thread resources. Beyond 500 users, the server overhead, such as the time taken to reject the requests, accumulates as the rejection rate increases.

Since SoapUI includes the rejection count in the evaluation of average response time, one way to f ind a good estimate of the actual response time is t o use a packet capture tool such as Wireshark and evaluate the response time for each successful request from the time the request is received by the server until the response is sent back to SoapUI. This requires filtering the blocked requests and evaluating the response time for each individual successful request, which seems to be infeasible. The other approach is to use 15.11 as the estimate of the actual response time for loads more than 370, where the rejections really start. This value shows the peak of response time when the system utilizes all the resources for high level loads without being biased by the rejected requests. Therefore, in this study *avg* = 15.11 is used as the closest approximation of average response time for high level loads.

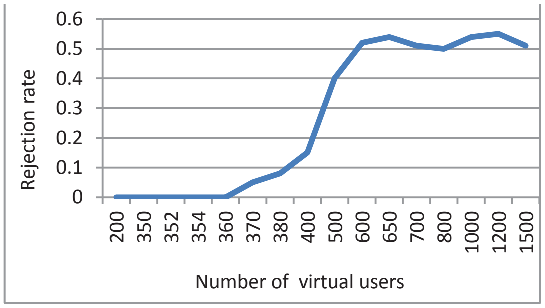


Figure 3. Request rejection rate based on number of virtual users.

C. The Stochastic Activity Network Model

Petri nets are a g raphical model for the formal specification and analysis of concurrent and discrete-event systems. Since the introduction of classic Petri nets, a number of variants have been introduced. Stochastic Petri Nets (SPN) is a subsidiary of timed Petri nets that adds non-deterministic time through randomness of transitions. Generalized Stochastic Petri Nets (GSPN) is a SPN performance analysis tool that uses the exponential random distribution, and thus conversion to Markov Chains is automated. Stochastic Activity Network (SAN) [21] is a structurally extended version of GSPN with many features such as the ability of creating complex enabling functions, probabilistic choices upon completion of activities, and reward functions. In the world of SANs, transitions (actions) are referred to as activities, which can be of two kinds: *timed* and *instantaneous*. The firing (activation) time of timed activities are exponentially distributed. Once enabled, instantaneous activities complete in zero-time, and thus have priority over timed activities. Reward functions are used to measure performance or dependability related issues.

The SAN model describing the behavior of the web service system is constructed using Mobius [22]. Mobius can solve SAN models, either mathematically or by simulation. Because of the types of reward rates used we have found it easier and less time consuming to work with the simulation solver. Fig. 4 provides a SAN model for the service requests that arrive at the server (JBoss) side. Recall that Duke's Bank web service is used in this experiment. This web service receives customer ID and returns all account numbers of the customer.

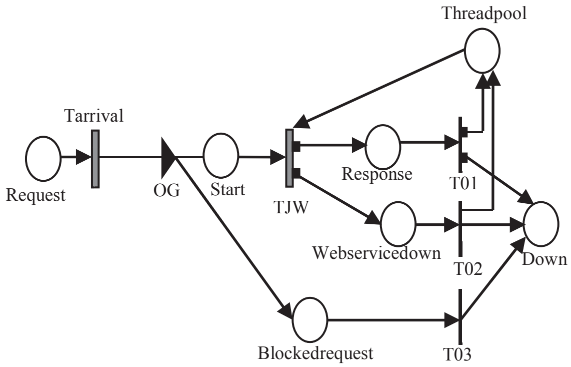


Figure 4. The Petri net model of JBoss serving the requests.

In the figure, the timed and instantaneous activities are shown by thick and thin bars, respectively. Each flat dot represents a probabilistic choice that leads to taking a different path in the model once the activity completes. The place called *Request* is initialized to *cnt*, the number of total HTTP requests for each test case. The rate of the *Tarrival* activity gives the rate of arrivals per unit of time, which is equal to *tps*. The HTTP thread instance pool in JBoss is represented by the *Threadpool* place, which is initialized to *maxThread* extracted from the configuration file named *server.xml*. Once *Tarrival* completes, the token generated, through the output gate *OG*, shown as a solid triangle, will be deposited in either *Start* or *Blockedrequest. OG* represents the conditions for rejecting HTTP requests. For instance, if *Start* has reached its maximum capacity, *OG* will redirect the token to *Blockedrequest*; otherwise the token is added to *Start*. The rejected requests are accumulated in *Down* via *T03* activity. Activity *TJW,* which represents the service rate of the application server, is enabled only if *Threadpool* and *Start* are not empty. When *TJW* is activated, a token in *Threadpool* representing an available thread in JBoss is allocated to a request in *Start*. Once the request is serviced, the thread, through *T01*, is released to *Threadpool* to be used by a next request. In case the server fails to service the request (lower case of *TJW*), the allocated thread is returned to *Threadpool* via *T02*. The failure probability (lower case) of *T01* is the probability that the service request is failed by means other than failures caused by the application server, such as a failure in the network or unregistered service. These failure probabilities are set to zero but can be set to non-zero values. Table I shows the parameters and their values used in the SAN model. The impulse reward functions count the number of activities that complete. The number of activations at *Tarrival* shows the total number of requests that has entered the Petri net model. Similarly, the number of activation completions at *T03* represents the number of requests rejected. Let $rejection_{SRN}$ be the number of these rejections, i.e. the number of *T03* activations.

TABLE I. SAN MODEL PARAMETERS

Parameter name	Parameter value
Requests	Initialized to *Cnt* (from SoapUI)
Tarrival rate	Initialized to *Tps* (from SoapUI)
Threadpool	Initialized to *maxThread* (from *server.xml* in JBoss AS)
OG	// *threadpool* = *maxThread* *// queue* = *acceptCount* from *server.xml* in // JBoss AS // *maxThread* default: 250 // *acceptCount* default: 100 If (Start →Mark() < (*threadpool* + *queue*)) Start →Mark() = Start →Mark()+1; Else Blockedrequest → Mark() = Blockedrequest → Mark()+1;
TJW rate	// *avg* from SoapUI // threshold = 315 If (Start →Mark() < Threadpool →Mark()) Return ((1/*avg*) * (Start→Mark()); Else Return ((1/*avg*) * *threshold*);
T01 probability case1	1
T01 probability case2	0
TJW probability case1	1
TJW probability case2	0
Impulse Reward Functions	Total number of requests: If Tarrival fires then return 1; Number of requests rejected: If T03 fires then return 1;

Since the threads in JBoss are executed in parallel, the if-clause of the *TJW* rate in the table ensures that there are enough threads to be allocated to the requests. The else-clause, however, is not intuitive enough. The else-clause should be looked at with the *OG* condition in mind. JBoss allows a maximum of *queue* requests to be queued. In other words, if *Threadpool* is empty, there can be at most *queue* tokens in *Start*. Consequently, if there are *x* < *threadpool* tokens in *Threadpool*, the maximum number of tokens in *Start* is (*queue* + *x*). Since the value of *x* changes depending on the available threads, the maximum tokens in *Start*, i.e. (*queue* + *x*), continuously changes as well. This causes *threshold* that represents the speed at which the SAN model services the requests to be dynamic. In other words, *threshold* needs to be throttled each time *x* changes. This makes it difficult to predict an appropriate *threshold* value that meets the rejection rate observed by the experiments performed using SoapUI. Thus, the maximum value of *Start* is set at the fixed value (*threadpool* + *queue*). This in turn makes *threshold* to be a fixed value. As it will be shown shortly, this approach has shown that performance of the SAN model is very close to that of the rejection rate reported by SoapUI.

III. Results

Sixteen tests have been conducted on the banking web service. Each test is run five times and the average of extracted data is considered as experimental data. Table II shows a sample data extracted from SoapUI. For each of the 16 different loads, *avg* and *tps* values returned by SoapUI are used in the SAN model (see Table I) and (5). Table III shows the results for the 16 tests obtained from SoapUI, the theoretical equation (5), and from running the SAN model. As it is shown in the table, the worst rejection rate happens when there are 380 users, which is 0.13 – 0.08 = 0.05, and 0.14 – 0.08 = 0.06 for the theoretical and the SAN model, respectively.

TABLE II. Sample Data Extracted From SoapUI

Users (threads)	Runs per thread	Cnt	Avg (sec)	Tps	Total time (sec)	Number of requests rejected	Request rejection rate
200	10	2000	7.46	24.68	82.55	0	
350	10	3500	13.76	22.92	153.25	0	0
352	10	3520	13.83	22.25	158.21	0.5	0
354	10	3540	13.69	22.61	156.58	5	0.001
360	10	3600	14.54	22.15	162.67	24.2	0.006
370	10	3700	15.11	21.78	170.46	185.4	0.05
380	10	3800	13.71	24.13	157.64	332	0.08

TABLE III. Request Rejection Rate for All Sixteen Tests

Number of simultaneous users	Request rejection rate (SoapUI)	Request rejection rate ($rejection_{est}$ /cnt)	Request rejection rate ($rejection_{SRN}$ /cnt)
200	0	0	0.01
350	0	0	0.04
352	0	0	0.03
354	0.001	0	0.03
360	0.006	0.02	0.04
370	0.05	0.04	0.05
380	0.08	0.13	0.14
400	0.15	0.17	0.17
500	0.4	0.4	0.4
600	0.52	0.5	0.49
650	0.54	0.52	0.52
700	0.51	0.52	0.52
800	0.5	0.54	0.53
1000	0.54	0.51	0.51
1200	0.55	0.52	0.52
1500	0.51	0.5	0.49

Fig. 5 displays graphically the request rejection rate for the three different models shown in Table III. The figure shows that the rejection rate of the analytical and the SAN models closely match the ones provided by SoapUI. Another interesting result from the experimental analysis is the throughput. Throughput is computed by:

$$throughput = \frac{cnt - rejection_{est}}{T} \qquad (6)$$

Fig. 6 shows the throughput as a function of users. As expected, the throughput decreases as the users are increased. The *tps* range reported by SoapUI is from 21 to 45 for an average of about 33, with a standard deviation of about 10. The average throughput is about 22.5, which means on average, 30% of the requests are rejected.

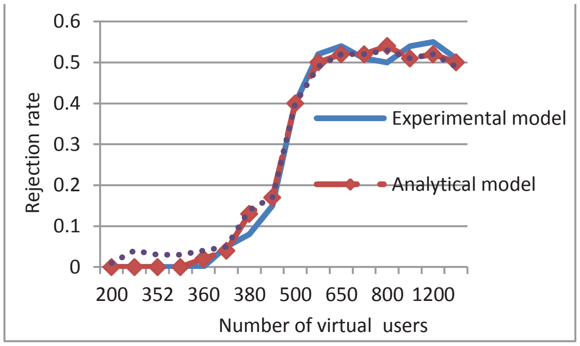


Figure 5. HTTP rejection rate.

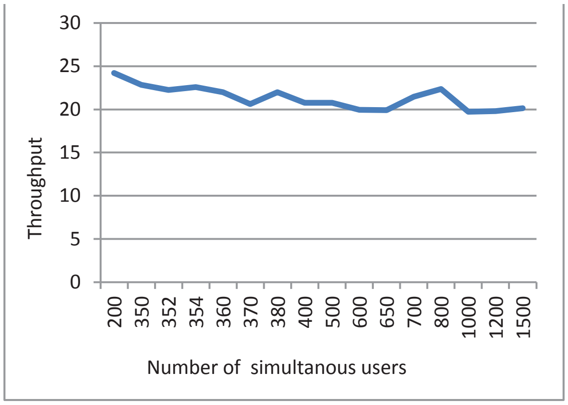


Figure 6. Throughput based on virtual users (extracted from soapUI).

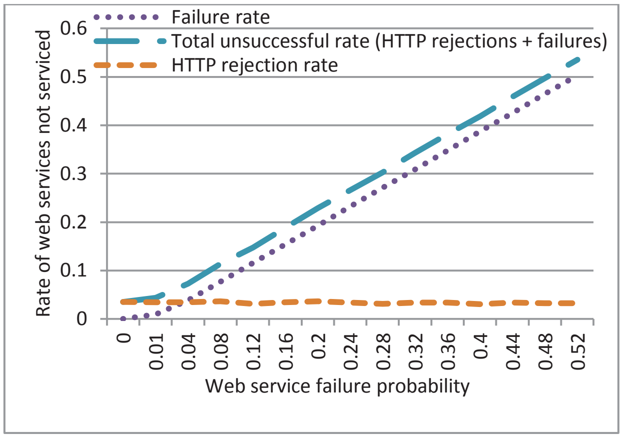


Figure 7. Web services rate not serviced using SAN model.

The close approximate behavior of the SAN model, as shown in Fig. 5, allows for cases that are more difficult to replicate in real world, e.g. inducing failures in the application server (*TJW* activity in Fig. 4). As an example, consider Fig. 7 that shows the rate of failures due to HTTP rejections and web service failures. This figure assumes the experiment with 354 simultaneous users with the same arrival rate and average response time shown in Table II. Rrunning the SAN model with different web service failure probabilities produces the rejection rate of about 3%. But the rate of successful web services is greatly impacted if the web service failure probability is greater than 4%. These probabilities are set by changing case 2 of *TJW* activity (see Table 1). Although *TJW* is the activity rate of the web services deployed, we are assuming the application server might fail to service some accepted requests but those web services do not cause the server to fail. Otherwise a failure in the server would require the removal of all requests from *Start* and restart of the server.

IV. Future Work

This study has concentrated on estimation of HTTP rejections. The research treated the application server software as a black box and showed that the SAN simulation model closely matches the experiment results. The current research is investigating a white-box approach (Fig. 8) in that a hierarchical SAN model is being created with submodels for the web server (Tomcat), the application server (JBoss), the database, and the web service itself. Since the correctness of the black-box approach has been verified through real test cases, the black-box results can be used to validate the hierarchical approach in that the white-box strategy ought to produce performance similar to those shown in this research. The white-box approach allows for finer granularity of analysis such as the impact of sensitive components on the overall reliability of the system that is otherwise not possible in the black-box approach.

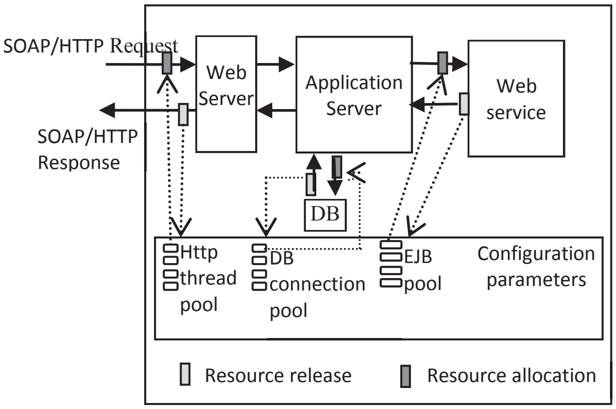


Figure 8. Architecture of JBoss, showing the interaction between layers and shared resource.

V. Conclusion

This study has emphasized on the combination of load testing and simulation modeling to predict rejections of HTTP requests in a banking web service deployed in JBoss. The analytical and the SAN models developed utilize the parameters extracted from the load tests, such as average response time and arrival rate in order to predict the rejection rate of HTTP requests. The accuracy of the models is validated by comparing the results of the models against those of the experimental ones.

Acknowledgment

This research is funded in part by Department of Defense (DoD)/Air Force Office of Scientific Research (AFOSR), NSF Award Number FA9550-07-1-0499, under the title “High Assurance Software”.

References

[1] M.N. Huhns, “Service-oriented computing: Key concepts and principles”, IEEE Internet Computing, pp. 75-81, Jan-Feb 2005.

[2] World Wide Web Consortium, http://www.w3.org/.

[3] S. S. Gokhale, P. J. Vandal, J. Lu, “Performance and reliability analysis of web server software architectures”, 12th Pacific Rim Int’l Symp on Dependable Computing , 2006.

[4] R. C. Cheung, “A user-oriented software reliability model”, *IEEE Trans on Soft Eng*, vol. 6, no. 2, pp. 118–125, 1980.

[5] K. Goˇseva–Popstojanova, K. Trivedi, “Architecture–based approach to reliability assessment of software systems”, *Performance Evaluation*, vol. 45, pp.179-204, 2001.

[6] M. Rahmani, A. Azadmanesh, H. Siy, “Architecture-based reliability analysis of web services in multilayer environment”, PESOS workshop, 33rd Int’l Conf on Soft Eng, 2011.

[7] B. Zhou, K. Yin, S. Zhang, H. Jian, A.J. Kavs, “A tree-based reliability model for composite web service with common-cause failures”, R.S. Chang et. al (Eds.), LNCS 6104, pp. 418-429, 2010.

[8] D. Zhong and Zhichang Qi, “A Petri net based approach for reliability prediction of web services”, LNCS 4277, pp. 116-125, 2006.

[9] F. Souza, R. Arteiror, N. Rosa, P. Maciel, “Using stochastic Petri nets for performance modelling of application servers”, Proc of 5th Int’l Work. on Performance Modeling, Evaluation and Optimization of Parallel and Distributed Systems, IEEE IPDPS, pp. 1-8, 2006.

[10] H. Singh, V. Cortellessa, B. Cukic, E. Guntel, and V. Bharadwaj, “A bayesian approach to reliability prediction and assessment of component based systems”, Proc. 12th Int’l Symp on Software Reliability Engineering, pp. 12–21, 2001.

[11] A. Filieri, C. Ghezzi, G. Tamburrelli, “Run-time efficient probabilistic model checking”, Int’l Conf Soft Eng, pp. 21–28, 2011.

[12] X. Xiao, T. Dohi, “Estimating the error rate in an Apache web server system”, Int’l J of Soft Eng and Its Applications, vol. 4, no. 3, 2010.

[13] R. D. van der Mei, R. Hariharan, P. K. Reeser, “Web server performance modeling,” Telecommunication Systems, vol. 16, no. 3/4, pp. 361-378, 2001.

[14] J. Cao, M. Andersson, C. Nyberg, M. Kihl, “Web server performance modeling using an M/G/1/K*PS queue”, Proc of 10th Int’l Conf on Telecommunications , pp. 1501- 1506, 2003.

[15] Z. Zheng, M. R. Lyu, “Collaborative reliability prediction of service-oriented systems”, Proc of 32rd Int’l Conf in Soft Eng, 2010.

[16] K. Goseva-Popstojanova, M. Hamill, and R. Perugupalli, “Large empirical case study of architecture-based software reliability,” Proc Int’l Symp Software Reliability Eng, pp. 43-52, 2005.

[17] JBoss Application Server, http://www.jboss.org/jbossas/.

[18] The Duke’s Bank Application, http://download.oracle.com/javaee/1.4/tutorial/doc/Ebank.html.

[19] Wireshark, http://www.wireshark.org/.

[20] SoapUI, http://www.soapui.org/.

[21] W.H. Sanders, J.F. Meyer, “Stochastic activity network: Formal definition and concept”, LNCS, vol. 2090, pp. 314-343, 2001.

[22] Mobius, https://www.mobius.illinois.edu/.

Implementing Web Applications as Social Machines Composition: a Case Study

Kellyton dos Santos Brito[1,2], Lenin Ernesto Abadie Otero[2], Patrícia Fontinele Muniz[2], Leandro Marques Nascimento[1,2]
[1]DEINFO – Federal Rural University of Pernambuco
Recife, Brazil
[ksb, leao, pfm, lmn]@cin.ufpe.br

Vanilson André de Arruda Burégio[2], Vinicius Cardoso Garcia[2], Silvio Romero de Lemos Meira[2,3]
[2]Informatics Center – Federal University of Pernambuco
[3]C.E.S.A.R - Recife, Brazil
[vaab, vcg, srlm]@cin.ufpe.br

Abstract: **With the evolution of the web and the concepts of web 3.0 as known as programmable web, several issues need to be studied in order to develop, deploy and use this new kind of application in a more effective way, such as communication between systems, unstructured data and non-scalable protocols, among others issues. In this regard, a new concept – named Social Machines – emerged to describe web based information systems that interact for a common purpose. In order to apply and validate in practice this new model, in this paper we describe a case study which implements a web application that is a composition of several public and well-known services from different application domains, such as Wikipedia, Flickr, Twitter, Google Places and Google Maps, following the Social Machines' model. In the end, we present the results and some improvement suggestions for the model.**

Keywords: social machines, web development, sociable applications, programmable web, case study.

I. Introduction

In web 3.0, the web as a programming platform [1], software is developed for the web, through the web, and in the web, using the web both as programming platform and deployment and execution environments. Thus, nowadays computing means connecting [2], due to the fact that developing software is almost the same as connecting services [3]. Examples of this scenario are the development of Facebook, Twitter, Yahoo!, Salesforce, Google, Amazon and many others, that makes their APIs available for anyone to develop applications that interact with their services.

In addition to those popular APIs, there are several public APIs and several applications that use them. For example, the ProgrammableWeb website[1] reached 5000 APIs in February 2012, and in the same month it listed more than 6500 mashups using them. Although there have been many studies about the future of the internet and concepts such as web 3.0, programmable web [1, 4], linked data [5] and semantic web [6, 7], the segmentation of data and the issues regarding the communication among systems, unstructured data, unreliable parts and non-scalable protocols are all native characteristics of the internet that needs a unifying view and explanations in order to be developed, deployed and used in a more efficient and effective way.

Furthermore, the read/write and programmable web are recent enough to represent very serious difficulties in understanding their basic elements and how they can be efficiently combined to develop real, practical systems in either personal, social or enterprise contexts. So, Meira et al. [8, 9] defined the concept of a web of Social Machines, in order to provide a common and coherent conceptual basis for understanding this still immature, upcoming and possibly highly innovative phase of software development.

In this context, in order to validate the Social Machines model and to test solutions for open issues, in this paper we revisited and applied the concepts of social machines, by performing a case study of a new application, according to following structure: in Section II we briefly present the Social Machines concept; in Section III we present a case study with the implementation of an application using the concept and guidelines; Section IV discusses the benefits, difficulties and challenges; and Section V concludes the paper and presents future works.

II. The Web of Social Machines

The concept of Social Machines overlaps other research fields and issues currently well studied such as SaaS, Cloud Computing, SOA and Social Networks, but we have not found works that directly deals with the concept adopted by this study. Roush [2] proposed Social Machine representing human operated machines responsible for socializing information among communities; Patton [10] defines them as virtual machines operating in given social fields; Fuglsang [11] discussed them as systems that (in society) consume, produce and record information and are connected at large; and Hendler [12] discusses social machines based on Berners-Lee [13] who defines social machines as machines that do the system and social administration while the people do the creative work. In addition, the robotics view of a social machine is that of one that can relate to people [14].

The social machine concept adopted by this work is an abstract model to describe web based information systems that

[1] www.programmableweb.com

could be a practical way of dealing with the complexity of the emerging programmable web. The concept starts from Kevin Kelly, of Wired fame, that is quoted as having said once: "*The internet is the most reliable machine ever made. It's made from imperfect, unreliable parts, connected together, to make the most reliable thing we have*". Thus, the social machines are these parts, which connected together compose a functional web system.

A Social Machine (**SM**) receives *requests* (**Req**) from other SM's and returns *responses* (**Resp**). The requests are converted to *inputs* (**I**) for a processing unit (**P**), which has *states* (**S**) and produces *outputs* (**O**). In addition, there are rules that define *relationships* (**R**) with other SMs, under a specific set of *constraints* (**Const**).

Formally, a SM can be defined as the tuple ***SM = <Rel, WI, Req, Resp, S, Const, I, P, O>***. More detailed description of elements can be found at [8, 9].

In addition, SMs are sociable stuff and, in nearly all cases, each one should provide means to interact with one another, giving the developer the liberty to connect any number of SMs through the Web in order to form different networks and implement new services from the ones that already exist.

The idea behind SMs is to take advantage of the networked environment they are in to make easier to combine and reuse existing services from different SMs and use them to implement new ones. In this context, SMs can be classified as: (**i**) Isolated: SMs that have no interaction with other SMs; (**ii**) Provider: SMs that provide services for other SMs; (**iii**) Consumer: SMs that consume services from other SMs; and (**iv**) Prosumer: SMs that both provide and consume services.

III. Implementing Web Applications as Social Machines Composition

Although the initial paper of SMs presented one application as a case study, more applications must be developed to maturate this concept, in special, applications that uses other API's and services instead of social networks services. Then we performed a new empirical study and implemented a new application using the model. Thus, as the objective of the study is to verify the feasibility and main benefits and challenges of software development using a this approach in a real-life context, we performed a case study, which was divided into five activities: (**a**) Definition, (**b**) Planning, (**c**) Operation, (**d**) Analysis and Interpretation, and (**e**) Presentation and Package.

A. Definition

In order to verify the feasibility to implement an application that uses several web API's of several application domains in according to SMs model, and to identify and analyze the main benefits, difficulties and challenges of this approach in these context, the study has two research questions:

Q_1. It is possible to implement this kind of application according to SMs model?

Q_2. What are the main benefits, difficulties and challenges of implementing this kind of application according to SMs model?

B. Planning

The planning follows the evaluation plan proposed by Basili et al. [15] and will be described in future tense, showing the logic sequence between the planning and operation.

Context: The study consists in an analysis and implementation of a software project. It will be conducted as a commercial project, by M.Sc. and D.Sc software engineering students in a postgraduate class of the Informatics Center of Federal University of Pernambuco (UFPE), Brazil.

Subjects: The subjects of the study are 1 D.Sc and 2 M.Sc software engineering students of UFPE. All of them are software developers and have previous experiences with industrial projects, technologies and tools.

Training: The subjects of the study will receive education about SMs and related technologies during the course. The training includes the concepts of SMs, DSL's, software architecture, public API's and cloud computing, and will be performed in the first half of the class.

Instrumentation: The subjects will be encouraged to use an agile methodology to project development, but at least a triplet of documents must be created: software requirements, architecture using SMs model, and a lessons learned document.

Null Hypothesis, H_0: This is the hypothesis that the experimenter wants to reject with a high significance as possible. In this study, the following hypothesis can be defined:

H_0': it is not possible to design and implement the application according to Social Machines model;

H_0'': there are no benefits in implementing the application according to Social Machines model;

H_0''': there are difficulties and challenges in implementing the application according to Social Machines model;

Alternative Hypothesis: This is the hypothesis in favor of which the null hypothesis is rejected. In this study, the following hypothesis can be defined:

H_1: it is possible to design and implement the application according to Social Machines model;

H_2: there are benefits in implementing the application according to Social Machines model;

H_3: there are no difficulties and challenges in implementing the application according to Social Machines.

Criteria: The evaluation criteria will be performed by a qualitative analysis of application documentation, in addition to public discussions involving the subjects, the professors and cloud computing and application development specialists from both UFPE and local companies.

Costs: Since the subjects of the study are postgraduate students of UFPE and the environment of execution is the university infrastructure and free tools, the execution of the project is free of costs.

Variables: In this study, the independent variables are the SMs model and the documentation required. We considered the dependent variables the feasibility of application implementation using the model, the benefits, difficulties and challenges of implementation. These variables will be measured by document analysis as well as promoted discussions. Finally, we consider control variable the comparison with previous project using SM model, the *Futweet*, that was implemented by a similar team in a similar context.

C. Operation

The operation was performed according to the planning. The project development was conducted as follows:

Requirements: Application goal is to help people not familiarized with one city or one city area, such as tourists, business people or citizens, to gather information about places nearby him. Using a smartphone or a traditional web browser, the application allows user to see nearby places (registered on google places or foursquare services) and to browse useful information about it, such as: wikipedia and google search information, photos from Flickr and real time comments from twitter, from or about the place.

To provide this information, the application flows must be in this sequence:

(**i**) User (in a computer or cell phone) opens the application, which automatically detects his location, searches for nearby places and shows these places in a map;

(**ii**) User selects one place in the map and gets the name, some initial information and a link for "more information";

(**iii**) User selects "more information" option and is redirected to a page containing the additional information related above: textual information, photos and comments by other people.

Architecture: To satisfy these requirements, the project team selected some initial APIs to provide needed information: Google Maps for map visualization; Foursquare and/or Google Places for places search, Wikipedia and/or Google Search for textual information, Flickr for photos, and Twitter for real-time comments.

According to SMs model, the application architecture is illustrated in Figure 1.

In the figure, each service is considered as a SM, and SMs with similar functionalities are grouped inside the same circle. The application is named *WhatHere*, and is mainly the "glue SM", which includes the application business rules. Clients are separated: Mobile and Web.

Each SM was described according to the previously mentioned tuple: ***<Rel, WI, Req, Resp, S, Const, I, P, O>***, and Figure 2 shows the specification of WhatHere social machine.

Due to its relevance, Analysis and Interpretation will be made in the next section.

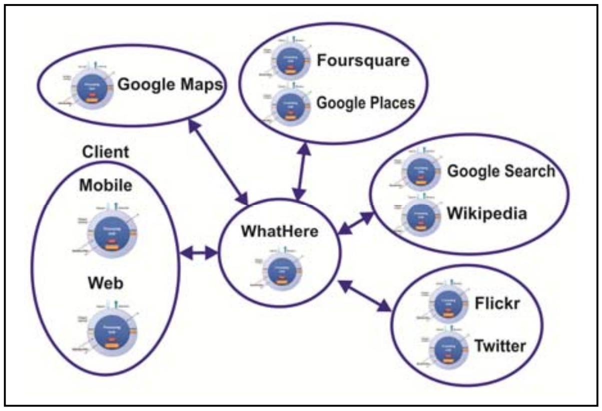
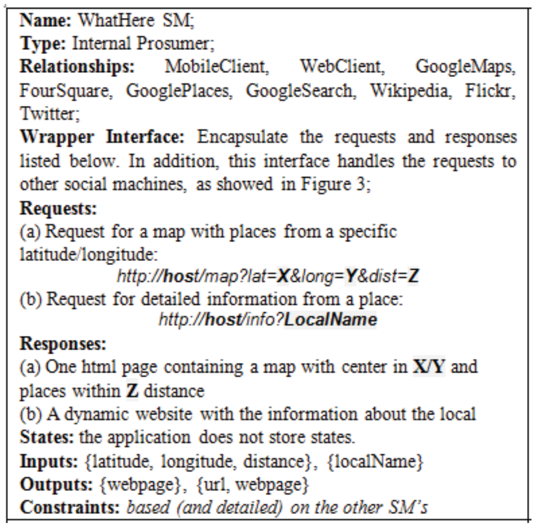


Figure 1: Application Architecture

IV. DEVELOPMENT ANALYSIS AND INTERPRETATION

According to the planning and in order to reject the null hypothesis and satisfy the alternative hypothesis, we analyzed qualitatively the application design, code and documentation, involving the subjects, the professors and some guests specialists in web API's, Cloud Computing and Software Atchitecture. First, the guests analyzed application design and performed a detailed code review. Then, documentation was analyzed and several discussions were performed between the subjects, professors and guests.



Name: WhatHere SM;
Type: Internal Prosumer;
Relationships: MobileClient, WebClient, GoogleMaps, FourSquare, GooglePlaces, GoogleSearch, Wikipedia, Flickr, Twitter;
Wrapper Interface: Encapsulate the requests and responses listed below. In addition, this interface handles the requests to other social machines, as showed in Figure 3;
Requests:
(a) Request for a map with places from a specific latitude/longitude:
*http://**host**/map?lat=**X**&long=**Y**&dist=**Z***
(b) Request for detailed information from a place:
*http://**host**/info?**LocalName***
Responses:
(a) One html page containing a map with center in **X/Y** and places within **Z** distance
(b) A dynamic website with the information about the local
States: the application does not store states.
Inputs: {latitude, longitude, distance}, {localName}
Outputs: {webpage}, {url, webpage}
Constraints: *based (and detailed) on the other SM's*

Figure 2: Internal SM definition

The application was designed and implemented in full compliance with its requirements and with the SMs model, which rejects the null hypothesis *H_0': it is not possible to design and implement the application according to Social Machines model*, which validates the alternative hypothesis $H_{1.}$

In addition, the subjects reported several benefits from applying this approach, mainly: (**i**) good system modularity and maintainability, because of separation of application rules and the services used; (**ii**) facilitation of the abstraction and use of external services and API's, centralized in the wrapper interface, instead of merged in application logic; (**iii**) reuse of SMs; due to the fact that the SMs are accessed by well defined requests/responses, they can be reused by several applications, and (**iv**) the reduced amount of lines of code implemented. These benefits reject the null hypothesis H_0*'': there are no benefits in implementing the application according to Social Machines model,* which validates the alternative hypothesis H_2.

However, the subjects reported some questions which not reject the null hypothesis H_0*''': there are difficulties and challenges in implementing the application according to Social Machines model.* Initially, they reported the difficulty to define some SM's, in special the SMs that are not developed by the team, such as Twitter or Wikipedia SMs, mainly because developers do not have full access to information about them. Thus, we concluded that can be more effective to create two classes of SMs – Internal and External – and reduce the tuple of external SMs to ***SM = <Rel, Req, Resp, S, Const>.***

Moreover, they also reported the difficulty to apply the proposal of SM Architecture Description Language (SMADL) presented in [8, 9], and prefer to represent items graphically or descriptively, such as in the Figure 2.

Finally, they related the need to create SMs controllers in order to manage groups of SMs. For example, instead of using only the Flickr to get photos, the application could use other image services, such as Google Images or Bing Images. Thereby, some mechanism is necessary to manage where to get the photos, based on heuristics, such as: service constraints, QoS or availability, and should change the policy dynamically, for example, if the preferred service became unstable. After several discussions, the subjects and specialists concludes that one possible approach is to create federations of SMs. The federations would be a group of SMs with similar services, and would have a manager, which implements the selection policies. In this way the application wrapper interface would only communicate and call the services exposed by federation manager. This approach have the potential of increase the application modularity and abstraction, because the final application developer won't need to know services API's, but only the federation manager API.

V. Conclusions and Future Works

In this paper, we revisited the concept of the emerging web of social machines [8, 9], a model to provide a common and coherent conceptual basis for the understanding of the young, upcoming and possibly highly innovative phase of software development, the "programmable web". Due to the fact that this is an initial concept and need to be more tested, we performed a case study by developing an application that uses several web API's of several application domains in according to Social Machines model.

The results of the case study showed that it is possible to implement web applications which interact with several web services of several domains. Some benefits were listed, such as the increase of modularity and maintainability, abstraction, reuse and the facility of use third part services and API's. Some improvements of the concept of SMs were suggested, such as the creation of Internal/External SMs, and the differentiation of the tuple for external SMs.

Moreover, we discuss a possible solution to some challenges presented in initial SMs preliminary definition, such as the creation of federation of SMs and a new type of SM, Federation Manager, which can be used to address these challenges.

For future developments we updated the preliminary agenda of social machines research, in special: (i) to define, implements and test a Social Machine' federation model; (ii) to investigate self-awareness characteristics of SMs and the possibility to, in conjunction with federations, allow the autonomous lifecycle management of SMs. In addition, the preliminary agenda are been performed, including an architectural framework for defining and developing SM-based Systems, security, billing, monitoring and fault tolerance questions, among others. Finally, more case studies are being developed to continue to validate and investigate the model.

Acknowledgment

This work was partially supported by the National Institute of Science and Technology for Software Engineering (INES[2]), funded by CNPq and FACEPE, grants 573964/2008-4, APQ-1037- 1.03/08 and APQ-1044-1.03/10 and Brazilian Agency (CNPq processes number 475743/2007-5 and 140060/2008-1).

References

[1] S. Yu and C. J. Woodard, "Innovation in the programmable web: Characterizing the mashup ecosystem," in *Service Oriented Computing - ICSOC 2008 Workshops*, 2009, pp. 136-147.

[2] W. Roush, "Social Machines - Computing means connecting," in *MIT Technology Review, August 2005*, 2005.

[3] M. Turner*, et al.*, "Turning Software into a Service," *Computer,* vol. 36, pp. 38-44, 2003.

[4] J. Hwang*, et al.*, "The structural evolution of the Web 2.0 service network," *Online Information Review,* vol. 33, pp. 1040-1057, 2009.

[5] C. Bizer*, et al.*, "Linked Data - The Story So Far," *International Journal on Semantic Web and Information Systems,* vol. 5, pp. 1-22, 2009.

[6] P. Hitzler*, et al.*, *Foundations of Semantic Web Technologies*: Chapman and Hall, 2009.

[7] T. Berners-Lee*, et al.*, "The Semantic Web," *Scientific American,* pp. 28-37, 2001.

[8] S. R. L. Meira*, et al.*, "The emerging web of social machines," *CoRR,* vol. abs/1010.3045, 2010.

[9] S. R. L. Meira*, et al.*, "The Emerging Web of Social Machines," in *Computer Software and Applications Conference (COMPSAC)*, Munich, 2011, pp. 26-27.

[10] P. Patton, *Deleuze and the Political*: Routledge, 2000.

[11] M. Fuglsang and B. M. Sorensen, *Deleuze and the social*: Edinburgh University Press, 2006.

[12] J. Hendler and T. Berners-Lee, "From the Semantic Web to social machines: A research challenge for AI on the World Wide Web," *Artificial Intelligence,* vol. 174, pp. 156-161, 2010.

[13] T. Berners-Lee and M. Fischetti, *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web*. New York: Harper Collins, 1999.

[14] T. N. Hornyak, *Loving the machine: the art and science of Japanese robots*: Kodansha International, 2006.

[15] V. R. Basili*, et al.*, "Experimentation in Software Engineering," *IEEE Transactions on Software Engineering,* vol. 12, pp. 733-743, July, 1986 1986.

[2] www.ines.org.br

Interactive Business Rules Framework for Knowledge Based Service Oriented Architecture

Debasis Chanda
Associate Professor, IIM Shillong
Shillong 793014
cdebasis04@yahoo.co.in

Dwijesh Dutta Majumder
Professor Emeritus, Indian Statistical Institute
203 BT Road, Kolkata 700035
ddmdr@hotmail.com

Swapan Bhattacharya
Professor & Head, Department of Computer Science & Engineering
Jadavpur University
Kolkata 700032
bswapan2000@yahoo.co.in

***Abstract* - In our paper we propose a SOA Framework based on Knowledge Bases, where Business Rules can be defined using Production Rule based Expert System. The same expert system would be instrumental for achieving user authorization functions.**

In our paper we consider an example scenario from the Banking domain for carrying out our discussions.

Keywords - Knowledge Base, Predicate Calculus, Service Oriented Architecture, Rule Based Expert System, IVRS

I. INTRODUCTION

The goal of this paper is to present an Interactive Business Rules Framework for Knowledge based Service Oriented Architecture, which uses Rules Based Expert System. With the provision of IVRS (Interactive Voice Response System), the proposed system will be of great help for visibility impaired people, since the system provides a simple step-by-step interactive user friendly process.

The rest of this paper is organized as follows: In Section II we furnish literature review of articles published in the relevant areas. In Section III, we introduce our modeling framework and an example scenario to facilitate understanding. In Sections IV we furnish benefits our proposed Framework. Finally, Section V provides some conclusions.

II. RELATED WORK

The Paper by Chanda et al. [1] proposes a SOA (Service Oriented Architecture) Framework based on Knowledge Bases, which serves the purpose of full-fledged modeling of both normal structural knowledge and dynamic behavior of the individual organizations participating in M&A/JV, as well as the consolidated organization. This paper is a continuation of the same.

The paper by El-Gayar et al. [2] reviews related work on service-oriented architecture (SOA), distributed infrastructure and business process management (BPM). In our Paper we focus on business rules in the Banking domain. This is relevant from the BPM context.

The paper by Fang et al. [3] suggests a framework for designing agile and interoperable Enterprises. Our Paper proposes a framework/architecture for business rules that may be adopted on enterprise-wide basis.

With the aim of facilitating the development of service oriented solutions, Marcos Lopez-Sanz et al [4] propose the specification of an architecture centric model driven development method. Our Paper adopts a Knowledge Based & Business Rule based approach for arriving at a service oriented model.

Service-based software architectures benefit in particular from semantic, ontology-based modeling. Claus Pahl's paper [5] presents ontology-based transformation and reasoning techniques for layered semantic service architecture modeling. Our paper provides model for configuring business rules.

Arroyo et al. [6] describe the practical application of a semantic web service-based choreography framework. We propose a business rules framework which is very much essential for choreography.

David Chen et al. [7] define and clarify basic concepts of enterprise architectures. We adopt an Expert System based Business Rules framework / architecture for services, which can be adopted on organization-wide basis.

The book by Luger [8] captures the essence of artificial intelligence. We apply artificial intelligence concepts for developing knowledge bases and business rules towards modeling business processes in our work.

III.BUSINESS RULES FRAMEWORK FOR KNOWLEDGE BASED SERVICE ORIENTED ARCHITECTURE

A. *PROPOSED APPROACH*

Our Paper proposes an Expert System based Business Rules Framework for Knowledge based Service Oriented Architecture to realize services and business processes (composition of services realized through their orchestration).

For our purpose, the representative domain of discourse considered is the deposit function of two banks (viz. Bank1 & Bank2). The realization of Business Rules Framework is based on Expert Systems [8].

B. *PREDICATE CALCULUS KNOWLEDGE BASE*

For our purpose, the representative domain of discourse considered is the deposit function of two banks, for which the knowledge can be represented as a set of Predicate Calculus expressions.

C. *PROPOSED MULTIPURPOSE SERVICE*

The realization of the Multi-purpose service is achieved through a Production Rule based Expert System. As an example of goal-driven problem with user queries, we consider an example of a simple expert system.

Let us consider the following four rules:

Rule 1

If

Name matches with Name in Bank1 Knowledge Base, and
Address Information matches with Address Information in Bank1 Knowledge Base

Then

Customer belongs to Bank1

Rule 2

If

Name matches with Name in Bank2 Knowledge Base, and
Address Information matches with Address Information in Bank2 Knowledge Base

Then

Customer belongs to Bank2

Rule 3

If

Name matches with Name in Bank1 & Bank2 Knowledge Bases, and
Address Information matches with Address Information in Bank1 & Bank2 Knowledge Bases

Then

Customer belongs to both Bank1 & Bank2

Rule 4

If

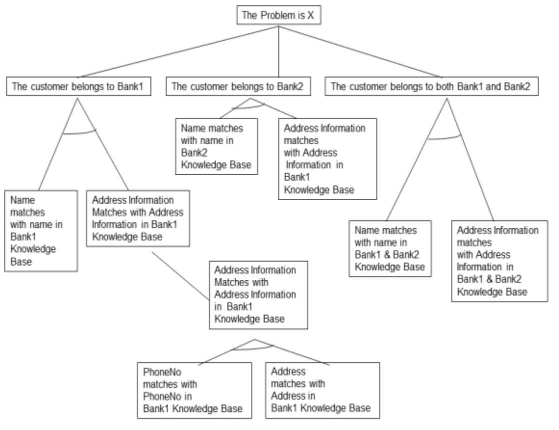
PhoneNo matches with PhoneNo in Bank1 Knowledge Base, and
Address matches with Address in Bank1 Knowledge Base

Then

Address Information matches with Address Information in Bank1 Knowledge Base.

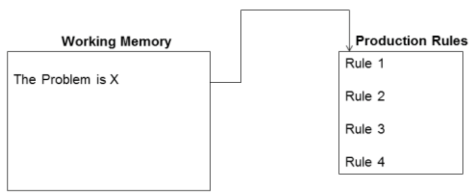
The graphical representation of the problem is furnished in Fig 1.

Fig 1: Graphical Representation of the Problem


The Problem is X
The customer belongs to Bank1
The customer belongs to Bank2
The customer belongs to both Bank1 and Bank2
Name matches with name in Bank2 Knowledge Base
Address Information matches with Address Information in Bank1 Knowledge Base
Name matches with name in Bank1 Knowledge Base
Address Information Matches with Address Information in Bank1 Knowledge Base
Name matches with name in Bank1 & Bank2 Knowledge Base
Address Information matches with Address Information in Bank1 & Bank2 Knowledge Base
Address Information Matches with Address Information in Bank1 Knowledge Base
PhoneNo matches with PhoneNo in Bank1 Knowledge Base
Address matches with Address in Bank1 Knowledge Base

To run this Knowledge Base under a goal-directed control regime, the top-level goal, 'The Problem is X' is placed in working memory as shown in Fig 2. X is a variable that can match with any phrase, e.g. 'The customer belongs to both Bank1 and Bank2'; it will become bound to the solution when the problem is solved.

Fig 2: The Production System at the start of a consultation



Three rules match with the expression in working memory: Rule 1, Rule 2 and Rule 3. If we resolve conflicts in favour of the lowest-numbered rule, then Rule 1 will fire. This causes X to be bound to the value ‘Customer belongs to Bank1’ and the premise of Rule 1 to be placed in working memory (Fig 3). The system has thus chosen to explore the possible hypothesis that the customer belongs to Bank 1. Another way to look at this is that the system has selected an ‘Or’ branch in the And / Or graph in Fig 1.

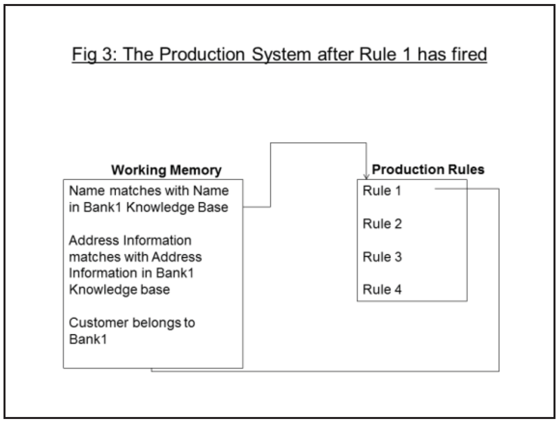


Fig 3: The Production System after Rule 1 has fired
Working Memory
Name matches with Name in Bank1 Knowledge Base
Address Information matches with Address Information in Bank1 Knowledge base
Customer belongs to Bank1
Production Rules
Rule 1
Rule 2
Rule 3
Rule 4

We also note that there are two premises to Rule 1, both of which must be satisfied to prove the conclusion true. These are ‘And’ branches of the search graph representing a decomposition of the problem (finding whether the customer belongs to Bank 1) into two sub problems (finding whether the PhoneNo matches with PhoneNo in Bank1 Knowledge Base, and Address Information matches with Address in Bank1 Knowledge Base). We may then fire Rule 4, whose conclusion matches with ‘Address Information matches with Address Information in Bank1 Knowledge Base’ causing its premises to be placed in working memory as in Fig 4.

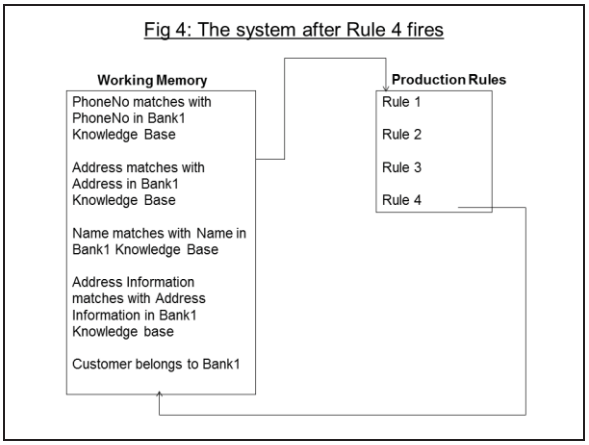


Fig 4: The system after Rule 4 fires
Working Memory
PhoneNo matches with PhoneNo in Bank1 Knowledge Base
Address matches with Address in Bank1 Knowledge Base
Name matches with Name in Bank1 Knowledge Base
Address Information matches with Address Information in Bank1 Knowledge base
Customer belongs to Bank1
Production Rules
Rule 1
Rule 2
Rule 3
Rule 4

At this point, there are three entries in working memory that do not match with any rule conclusions. Our expert system will, in this situation, query the user customer directly about these sub goals. If the customer confirms all three of these as true, the expert system will have successfully determined that the customer belongs to Bank1. In this case, the use case will return the balance of the customer from Bank1. In our example, had we failed to determine that the customer belongs to Bank1, we would have needed to back up to the top level and try Rule 2 instead.

Let us now consider a trace of the reasoning process. Suppose our expert system has an explanation facility, and let us assume that the consultation has reached the stage represented by the working memory configuration in Fig 4.

The following dialogue begins with the computer asking the customer about the goals present in the working memory.

Is your Phone Number 12345678?
Yes
Is your Address po_box_19_AB_Avenue_Kolkata?
Yes
Is your Name prakash_kumar?
Why

Here the system responds with a literal presentation of the rule under question.

It has been established that:
The Address Information matches with that in Bank1 Knowledge base,
Therefore if
The Name matches with that in Bank1 Knowledge base
Then you are a customer of Bank1.

Following the above trace, the deposit of customer would be displayed.

Because the customer answered yes to the first two questions, Rule 4 was able to fire, verifying the second premise of Rule 1. When the ‘why’ query was made, the system was trying to solve the first premise of Rule 1. It is to be noted that the explanation for the systems behavior is just a restatement of the current rule under consideration. However, this provides an adequate explanation for why the program wants to know whether the name is prakash_kumar. In the response to why query, the system states that it h as concluded that the Address Information matches with that in Bank1 Knowledge Base. Suppose the customer then asks:

How the Address Information matches with that in Bank1 Knowledge Base

The system will respond to how it concluded that the Address Information matches with that in Bank1 Knowledge base. The response is a tr ace of the reasoning that led to this conclusion, working back from the goal along the rules that support it to the user responses.
This follows from Rule 4:

If: PhoneNo matches with PhoneNo in Bank1 Knowledge Base, and
Address matches with Address in Bank1 Knowledge Base
Then: Address Information matches with Address Information in Bank1 Knowledge Base.
PhoneNo provided by Customer
Address provided by Customer

The production system architecture provides an essential basis for these explanations. Each cycle of the control loop selects and fires another rule. The program may be stopped after each cycle and inspected. Because each rule represents a c omplete chunk of problem-solving knowledge, the current rule provides a context for the explanation. It can also be seen from the above how authentication of the user/customer is achieved my means of the Production Rule based expert system. The expert system can similarly be used for implementing role based authorization by defining appropriate production rules e.g.

Rule 5
If
Role matches with Role in Bank1 Knowledge Base, and
Address Information matches with Address Information in Bank1 Knowledge Base
Then
Customer is authorized to access Bank1Knowledge Base

The graphical representation uses predicate calculus expressions in list syntax.

IV.BENEFITS OF USING BUSINESS RULES FRAMEWORK FOR KNOWLEDGE BASED SERVICE ORIENTED ARCHITECTURE

The following benefits are realized when an Expert System based Business Rules Framework is used vis-à-vis other approaches.

- Flexible configuration of Business Rules
- Highly Interactive Approach
- Explanation to the User regarding the conclusions reached
- **With the provision of IVRS (Interactive Voice Response System), the proposed system will be of great help for visibility impaired people, since the system provides a simple step-by-step interactive user friendly process**

V.CONCLUSION

Our proposed Business Rules Framework using Rule Based Expert System facilitates easy configuration of Business Rules for Business Process Modeling. The proposed Business Rules Framework is highly interactive in nature, and with the provision of IVRS (Interactive Voice Response System***) will be of gr eat help for visibility impaired users***, since the system provides a simple step-by-step interactive user friendly process.

REFERENCES

[1] Debasis Chanda, D Dutta Majumder, Swapan Bhattacharya, "Knowledge Based Service Oriented Architecture for M&A", Accepted for Publication in SEKE 2010 (The 22nd Conference on Software Engineering and Knowledge Engineering), San Francisco Bay, USA, July 1-3, 2010

[2] Omar El-Gayar, Kanchana Tandekar, An XML-based schema definition for model sharing and reuse in a distributed environment, Decision Support Systems 43 (2007) 791–808

[3] Chua Fang Fang, Lee Chien Sing, Collaborative learning using service-oriented architecture: A framework design, Knowledge-Based Systems 22 (2009) 271–274

[4] Marcos L´opez-Sanz, Cesar J. Acuna, Carlos E. Cuesta, Esperanza Marcos, Modelling of Service-Oriented Architectures with UML, Electronic Notes in Theoretical Computer Science 194 (2008) 23–37

[5] Claus Pahl, Semantic model-driven architecting of service-based software systems, Information and Software Technology 49 (2007) 838–850

[6] Sinuhe Arroyo, Miguel-Angel Sicilia, Juan-Manuel Dodero, Choreography frameworks for business integration: Addressing heterogeneous semantics, Computers in Industry 58 (2007) 487–503

[7] David Chen, Guy Doumeingts, Francois Vernadat, Architectures for enterprise integration and interoperability: Past, present and future, Computers in Industry 59 (2008) 647–659

[8] George F Luger, AI Structures and Strategies for Complex Problem Solving, Pearson Education, Fourth Edition, 2006

Defining RESTful Web Services Test Cases from UML Models

Alexandre Luis Correa
Departamento de Informática Aplicada
UNIRIO
Rio de Janeiro - Brazil
alexandre.correa@uniriotec.br

Thiago Silva-de-Souza
Escola de Ciência e Tecnologia
Universidade do Grande Rio (UNIGRANRIO)
Duque de Caxias - Brazil
thiagoein@gmail.com

Eber Assis Schmitz,
Antonio Juarez Alencar
Universidade Federal do Rio de Janeiro (UFRJ)
Rio de Janeiro - Brazil
eber@nce.ufrj.br,
juarezalencar@dcc.ufrj.br

Abstract— **This paper presents an approach for RESTful Web Services test case generation. RESTful Web Services have features that are not fully covered by traditional software testing techniques. The proposed approach uses model transformation techniques to generate platform independent test cases from UML class models enriched with Object Constraint Language (OCL) constraints. These test cases are then transformed into platform specific test cases that can be used to verify the implementation of CRUD RESTful Web Services.**

Keywords-software testing; RESTful web services; UML; OCL.

I. Introduction

Recently, a new category of web services, called RESTful Web Services, has emerged as the predominant Web service design model. RESTful Web Services follow the REpresentational State Transfer (REST) architectural style and explicitly use HTTP methods in a way that follows the protocol as defined by RFC 2616 [7].

RESTful Web Services manipulate resources. A resource is any information item accessible through a Universal Resource Identifier (URI). Resources are described by data and metadata in a MIME (Multipurpose Internet Mail Extensions) compliant representation [7]. Resources are manipulated by transferring representations between clients and servers using the operations specified in the HTTP protocol: POST creates a new resource; GET retrieves the current state of a resource in some representation; PUT replaces the current state of a resource; DELETE removes a resource [18]. Therefore, RESTful Web Services bind each operation of a resource to a specific HTTP method.

In service-oriented architecture environments, it is imperative that each service correctly performs its functions. In most software development endeavors, testing is an important quality control technique. The main purpose of software testing is to detect software failures [14]. Functional testing is a type of testing where test cases are defined from the functional specification of the software under test. Therefore, the effectiveness of the test cases is directly related to quality of the specification.

In general, RESTful web services are described in a service contract using the Web Application Description Language (WADL) format [15]. Therefore, a WADL descriptor is a service contract, represented in XML, which describes all the operations allowed on resources, the URI templates and the supported representation formats [18]. A WADL document, however, does not specify any business constraints that should be satisfied by service operations, since it focus on the specification of technical aspects. In data-oriented services, such as CRUD (Create, Retrieve, Update and Delete) services, a WADL document does not specifies neither the invariants of the domain, nor the pre and post-conditions of each operation. Therefore, in order to allow the generation of more effective test case suites, services must be described by richer semantic specifications.

Resources handled by RESTful web services can be represented in several formats. Thus, RESTful web services testing should take into account not only the information used in each operation, but also the representation format of that information. For example, an operation that retrieves a *Company* resource can return the result in several possible formats, e.g., JSON, XHTML. Therefore, the data formats used in the inputs and outputs of each service operation is an additional variable that must be considered in the design of test cases for RESTful web services.

In RESTful web services there is a natural mapping between the HTTP methods (POST, GET, PUT and DELETE) and most CRUD business operations exposed by a service. A problem that arises when the implementation details of a CRUD data service are not available is related to the precedence relations between CRUD operations, e.g., a resource can only be retrieved if it has already been created. Therefore, the generation of test cases for CRUD data services should take such precedence relations into account.

This paper proposes a specification-based approach for CRUD RESTful Web Services test case generation. Test cases are derived from services specified using UML and OCL (Object Constraint Language [11]. This paper is structured in four sections. Section 2 discusses the related work. Section 3 presents the proposed approach and Section 4 draws some conclusions.

II. Related Work

Several approaches for web services testing have been proposed. In [3], [2] and [6], the most relevant approaches are described and compared. Most of the existing web services testing approaches aim at deriving testing cases from a service specification. These approaches can be classified into two categories: a) testing based on standard specification of web services and b) testing based on semantic specification of services. In the former category, test cases are generated by parsing WSDL [17] and XML Schema documents, whilst in the latter category, test cases are generated from richer semantic specifications produced using a more expressive language such as the Web Ontology Language (OWL) [2].

Two recent studies have proposed approaches to test case generation based on contracts specified in Web Service Semantics (WSDL-S) [16] and OCL. Both approaches use methods based on combinatorial analysis to reduce the number of generated test cases providing satisfactory coverage. The approach described in [9] uses the Pair-Wise Testing (PWT) method, whilst the Askaruinisa and Abirami's approach [1] uses the Orthogonal Array Testing (OAT) method. However, such approaches are not directly applicable to CRUD RESTful Web Services Testing since they can only generate test cases for web services that: i) do not change the system state, ii) handle simple data types and iii) are based on WSDL-S format, which are specific for WS-* services.

RESTful Web Services testing is still an emerging area. Chakrabarti and Rodriquez [4] and Van Gilst and Reza [12] are two of the few published work in this area. The former describes an algorithm to automatically test the connectivity of RESTful web services via their URI address. The latter proposes a framework for RESTful web services testing based on the generation of perturbation data on input parameters present in the operation specification. This work, however, uses proprietary XML formats for the representation of the input parameters, which can hinder their acceptance.

III. Proposed Approach

The proposed approach aims at producing test cases for CRUD RESTful Web Services that handle resources based on complex data types. It follows Model-Driven Development approaches and, in particular, the PIM (Platform Independent Model) - PSM (Platform Dependent Model) structure defined in the Model-Driven Architecture (MDA) specification [10]. Test cases are derived from two models: a PIM model that specifies the service independent from the implementation technology, and a PSM model that specifies the aspects related to its implementation as a RESTful Web Service.

Sections A, B, C and D describes these approach using a simple example of CRUD services that maintain instances of *Course* and *Student* entities in a school domain.

A. Platform-Independent Model

The first activity of the approach corresponds to the elaboration of a Platform Independent Model (PIM). The first element of the PIM of a service is the class model. This model is composed by three main categories of classes: service classes; domain classes and representation classes.

A service class defines the CRUD operations that manipulate instances of a domain class. Each service class is annotated with the `<<crud>>` stereotype, defining at least four basic operations on the target entity (createXXX, retrieveXXX, updateXXX, deleteXXX, where XXX is the name of the target entity). Each operation is associated with a specific stereotype, i.e., `<<create>>`, `<<retrieve>>`, `<<update>>`, `<<delete>>`. Each domain class represents a business entity whose instances are handled by the operations of a service class, being annotated with the `<<entity>>` stereotype.

Representation classes are used to define input and output data structures used in service operations, similar to the Data Transfer Object pattern (DTO) [5]. For example, the *createStudent* operation receives an instance of the *StudentRep* class containing the input information used by the implementation to create the instance of the *Student* entity to be added to the service memory as a result. A representation usually contains a structure similar to its corresponding entity class, i.e., it is described by almost the same attributes. Representation classes are annotated with the `<<representation>>` stereotype. Figure 1 illustrates an excerpt of the model produced for two CRUD services of the example domain.

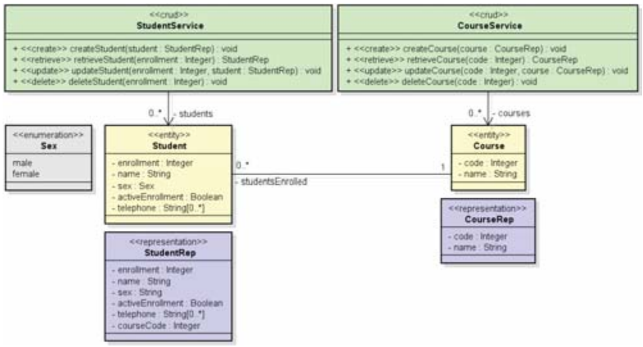

<<crud>>
StudentService
+ <<create>> createStudent(student : StudentRep) : void
+ <<retrieve>> retrieveStudent(enrollment : Integer) : StudentRep
+ <<update>> updateStudent(enrollment : Integer, student : StudentRep) : void
+ <<delete>> deleteStudent(enrollment : Integer) : void
<<crud>>
CourseService
+ <<create>> createCourse(course : CourseRep) : void
+ <<retrieve>> retrieveCourse(code : Integer) : CourseRep
+ <<update>> updateCourse(code : Integer, course : CourseRep) : void
+ <<delete>> deleteCourse(code : Integer) : void
0..*
- students
0..*
- courses
<<enumeration>>
Sex
male
female
<<entity>>
Student
- enrollment : Integer
- name : String
- sex : Sex
- activeEnrollment : Boolean
- telephone : String[0..*]
0..*
- studentsEnrolled
1
<<entity>>
Course
- code : Integer
- name : String
<<representation>>
StudentRep
- enrollment : Integer
- name : String
- sex : String
- activeEnrollment : Boolean
- telephone : String[0..*]
- courseCode : Integer
<<representation>>
CourseRep
- code : Integer
- name : String

Figure 1: Class model of School domain.

Rules constraining the values of the attributes of entity classes should be defined using invariants. These rules should be taken into account in the specification of the service operations. Figure 2 illustrates an example of this situation. In lines 1-2, an invariant defines the 1-99 range as the allowed values for the attribute *code* defined in the *Course* entity. The operation *createCourse* defined in the *CourseService* class, in turn, receives a data structure called *CourseRep*. As the invariant constrains the valid values for the *code* attribute, a pre-condition on the received value must be defined in the operation specification (lines 4-6).

```
1 context Course
2 inv: code >= 1 and code <= 99
3
4 context CourseService::createCourse(course :
5                         CourseRep)
6 pre: course.code >= 1 and course.code <= 99
```

Figure 2: Constraints on course code.

To avoid redundancy in the specification of constraints and preconditions of service operations, we suggest that constraints on attribute values should be specified in service classes as auxiliary operations. These operations can be referenced both in invariant and preconditions. Figure 3 shows an example of a refactored version of the constraints presented in Figure 2. The auxiliary operation *validCode* defined in the *CourseService* class (lines 1-3) is used in the specification of an invariant (lines 5-6) and a precondition (lines 8-10).

```
1 context CourseService::validCode(code : Integer) :
2                                  Boolean
3 body: code >= 1 and code <= 99
4
5 context Course
6 inv: courseService.validCode(code)
7
8 context CourseService::createCourse(course :
9                                     CourseRep)
10 pre: self.validCode(course.code)
```

Figure 3: Auxiliary operation used by an invariant and a pre-condition.

The PIM model should also specify the contract of each service operation using OCL pre and post-conditions. Since the operations of different CRUD services have similar behavior, e.g., create operations of *Student* and *Course* CRUD services are similar, we use templates, described in [13], to guide the writing process of contracts for each operation type. Figure 4 shows the contract for the *createCourse* operation, in which the preconditions (lines 3-5) state that a course can only be created if the arguments contain a valid code and a valid name (associated rules are defined as operations of the *CourseService* class) and if there is no course with the same code received as parameter (unique constraint on the course code). The post-condition (lines 8-14) ensures that, after the operation execution, a new instance will be added to the set of instances of *Course* existing at the invocation of that operation.

```
1 context CourseService::createCourse(cr :
2                                     CourseRep)
3   pre validCode: self.validCode(cr.code)
4   pre validName: self.validName(cr.name)
5   pre inexistentCourse: not courses->exists(c :
6                         Course| c.code = cr.code)
7
8   post newCourseCreated:
9    let newCourse : Course =
10     courses->select(c : Course | c.oclIsNew() and
11                    c.code = cr.code and
12                    c.name = cr.name)->asSequence()
13                                     ->first() in
14     courses = courses@pre->including(newCourse)
```

Figure 4: Contract of *createCourse*() operation.

B. *Platform-Specific Model*

The elaboration of the Platform Specific Model (PSM) consists in extending the PIM model by redefining stereotypes and adding tagged values to the service classes. The `<<crud>>` stereotype associated to classes in the PIM model is replaced by the `<<restservice>>` stereotype to indicate classes that implement RESTful web services. Stereotypes associated with operations in the PIM model are replaced by the `<<POST>>`, `<<GET>>`, `<<PUT>>` and `<<DELETE>>` stereotypes according to the operation type (create, retrieve, update, delete).

Tagged values may be used in two ways: (i) class scope and (ii) operation scope. In class scope, the tagged values “Base”, “Path”, “Consumes” and “Produces” are defined. “Base” indicates the base URI of the application, “Path” adds a relative path to base URI, “Consumes” and “Produces” respectively indicate the default representation formats consumed and produced by the service in all operations. In operation scope, the last three tagged values can be also used to define information specific to each operation. Figure 5 shows an example of stereotypes and tagged values defined in the *CourseService* class.

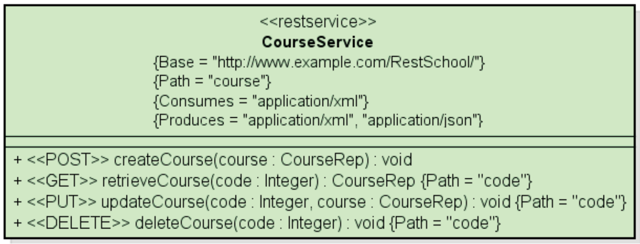

<<restservice>>
CourseService
{Base = "http://www.example.com/RestSchool/"}
{Path = "course"}
{Consumes = "application/xml"}
{Produces = "application/xml", "application/json"}
+ <<POST>> createCourse(course : CourseRep) : void
+ <<GET>> retrieveCourse(code : Integer) : CourseRep {Path = "code"}
+ <<PUT>> updateCourse(code : Integer, course : CourseRep) : void {Path = "code"}
+ <<DELETE>> deleteCourse(code : Integer) : void {Path = "code"}

Figure 5: Example of a class in the PSM model

C. *Platform-Independent Test*

The definition of PIT test cases is based on the constraints represented in the class model, e.g., attribute and association multiplicities, and on OCL constraints added to the PIM model. The technique is divided in two phases. In the first phase, a decision table derived from the constraints is created according to the following steps:

- Apply the boundary value analysis technique to the range of values defined by the invariants, enumerations and association multiplicities related to each entity class, separating them into valid and invalid values. Example for the *Course* entity: attribute code (null, 0, 1, 99, 100); attribute name (null, size 1, size 20, size 21).

- Generate entries in the decision table for valid input and invalid input combinations. Each row should correspond to the combination of at most one invalid value with valid values for the remaining input columns. This avoids combinations with more than one invalid value, since the expected behavior for one invalid input is the same as for more than one invalid input.

- Complete the decision table with the expected result for each entry. The expected result for combinations of valid values is success, whilst the expected result for rows containing an invalid input is error.

Table 1 shows the decision table generated to test the constraints on attribute values of the *Course* class.

Table 1: PIT decision table.

#	Input		Expected Result	
	code	name	Success	Error
1	null	size 1		X
2	0	size 1		X
3	1	null		X
4	1	size 1	X	
5	1	size 20	X	
6	1	size 21		X
7	99	size 1	X	
8	99	size 20	X	
9	100	size 1		X

In the second phase, test case procedures as defined as sequences of steps using the operations available in the interface of the service. Since test cases for CRUD services depend on the state of the service, defined by the existing instances and links between them, it is important to define a strategy to address the precedence relationships that exist in CRUD operations, i.e., in order to retrieve a course, we must ensure that this course has already been created. However, in black-box testing, we should rely only on the operations provided by the service.

One strategy used to minimize the number of steps per test case is the Chained Tests pattern described by Meszaros [8]. In this pattern tests are designed sequentially, so that the final state of a test is used as an initial state of the next test. However, the use of this pattern is not recommended since it violates the principle of independence between tests.

Our approach considers that each test case should create and clean up the state necessary for its execution. However, we only use the operations provided on the interface services, instead of using auxiliary test operations to control the state system, as suggested by *Back Door Manipulation* pattern [8]. Thus, one should consider that the service under test has an empty state and that each test case may use service operations to arrange the initial state (setup) and to clean up all state changes after test execution (tear down).

Table 2 shows a PIT test case for the *createCourse* operation whose steps are defined as calls to service operations. This example corresponds to the entry defined in line 4 of the decision table shown in Table 1.

Table 2: Example of PIT test case for *createCourse* operation.

Step Sequence	Expected Result	Step Function
1. retrieveCourse(1)	Error	Check pre-condition
2. createCourse({1,a})	Success	Execute main operation
3. retrieveCourse(1)	Success	Check post-condition
4. deleteCourse(1)	Success	Teardown

D. *Platform-Specific Test*

The PST test case generation translates the steps of the PIT test cases to HTTP requests, according to the appropriate method. Create operations are replaced by POST requests; read operations are transformed into GET requests; update operations are mapped into PUT requests; removal operations are represented by DELETE requests.

The verification of the expected results is performed by evaluating assertions on the response codes and headers of HTTP requests and responses. One or more assertion is done for each request, and the expected result is achieved when all assertions are true.

The technique considers in asserting the client response code for success and error (2xx and 4xx categories, respectively). Every HTTP response has a response code (status code), often used for error control. Table 3 shows the common response codes for success and error.

Table 3: Common response codes for HTTP verbs.

HTTP Verb	Success Code	Error Code
POST	201	400
GET	200	404
PUT	201	400
DELETE	200	400 or 404

The 201 status code used in POST and PUT methods indicates that the resource was successfully created or changed and that it can be referenced by a URI. The 200 status code, shared by GET and DELETE methods in successful HTTP responses indicates that the response contains the resource representation or that the resource was found and properly removed. The 400 status code used by POST, PUT and DELETE indicates that the operation was not performed because the client request contains a set of data that violates a rule operation. Finally, the 404 code used in GET and DELETE methods indicates that the resource was not found in the specified URI.

Assertions on error situations are based only in the status code. On the other hand, assertions of successful results to POST, GET and PUT methods also use some headers of the protocol itself. These headers are used to verify possible violations of technical constraints defined in the PSM.

POST requests use the Location header to indicate the URI of the resource just created. Thus, the verification of resource creation is not restricted to the status code. POST and PUT methods also uses the Content-Type header of the response to compare it with the Content-Type header of the request. This header indicates the representation formats supported by each operation (MIME types). So it is possible to check if the service implements correctly the configuration defined in the "Produces" and "Consumes" tagged values in PSM. These rules are shown in Table 4 which defines as a general definition of assertions associated to each HTTP method.

Table 4: General assertion model for PST test cases.

HTTP Verb	HTTP Success Response	HTTP Error Response
POST	*Status Code* = 201	*Status Code* = 400
	Location <> null	
	Content-Type = *Content-Type* of request	
GET	*Status Code* = 200	*Status Code* = 404
	Content-Type ⊂ *Accept* of request	
PUT	*Status Code* = 201	*Status Code* = 400
	Content-Type = *Content-Type* of request	
DELETE	*Status Code* = 200	*Status Code* = 400 or 404

From this model of assertions it is possible to derive test case templates for each operation type. The operations defined in the PIT test cases should be transformed in order to use their corresponding HTTP methods and the corresponding assertions should be defined according to Table 4.

Table 5 shows the positive PST test case model for create operations. Line 1 performs a GET request, equivalent to a PIT retrieve operation, in order to ensure that the resource to be created does not exist in the service memory. Line 2 performs the main test case request, requesting the creation of a resource using valid data. Line 3 performs a new GET query to check if the resource has been created. The DELETE request in line 4 removes the resource created by the request of line 2, cleaning up the service state.

Table 5: Positive PST test case model for creating operations.

Request Sequence	Expected Result
1. GET	*Status Code* = 404
2. POST	*Status Code* = 201
	Location <> null
	Content-Type = *Content-Type* of request
3. GET	*Status Code* = 200
	Content-Type ⊂ *Accept* of request
4. DELETE	*Status Code* = 200

Based on the PIT defined in Table 2 and on the template shown in Table 5, Table 6 shows a positive test case implementation for *createCourse* operation. The URI is derived by concatenating the tagged values "Base" of *CourseService* class and "Path" of *createCourse* operation shown in Figure 5.

Table 6: Positive PST test case for *createCourse*() operation.

Request Sequence	Expected Result in the Response
GET http://www.example.com/RestSchool/course/1 Accept: application/xml, application/json	*Status Code* = 404
POST http://www.example.com/RestSchool/course Accept: application/xml, application/json Content-Type: application/xml <course> <code>1</code><name>a</name> </course>	*Status Code* = 201 *Location* <> null *Content-Type* = *Content-Type* of request
GET http://www.example.com/RestSchool/course/1 Accept: application/xml, application/json	*Status Code* = 200 *Content-Type* ⊂ *Accept* of request
DELETE http://www.example.com/RestSchool/course/1 Accept: application/xml, application/json	*Status Code* = 200

IV. CONCLUSION

This work presented a systematic approach for generating test cases for CRUD RESTful Web Services. The approach blends techniques focused on the specification of such services and techniques focused on the test case generation based on those specifications. The main contributions of this paper are related to the systematization of the production of test cases production for services implemented following the REST architecture.

Results collected by experimental studies showed that the quality of the test cases generated with the proposed approach is significantly better than those usually produced by students and test practitioners in industry. The proposed approach does not impose a specific tool for implementing the test cases, i.e., any tool or HTTP library can be used.

REFERENCES

[1] A. Askaruinisa and A. M. Abirami. Test Case Reduction Technique for Semantic Based Web Services. International Journal on Computer Science and Engineering (IJCSE) , 02 (03), pp. 566-576, 2010.

[2] M. Bozkurt, M. Harman, and Y. Hassoun. Testing Web Services: A Survey. King's College London, Department of Computer Science, Londres, 2010.

[3] G. Canfora and M. Penta. Service-Oriented Architectures Testing: A Survey. Software Engineering: International Summer Schools, ISSSE 2006-2008 , pp. 78-105, 2009.

[4] S. K. Chakrabarti and R. Rodriquez. Connectedness testing of RESTful web-services. Proceedings of the 3rd India Software Engineering Conference (ISEC '10) (pp. 143-152). New York, NY: ACM, 2010.

[5] R. Daigneau. Service Design Patterns: fundamental design solutions for SOAP/WSDL and RESTful Web Services. Upper Saddle River: Pearson, 2012.

[6] A. T. Endo and A. D. Simão. Formal Testing Approaches for Service-Oriented Architectures and Web Services: A Systematic Review. Universidade de São Paulo, São Carlos, 2010.

[7] R. T. Fielding. Architectural styles and the design of network-based software architectures. PhD Thesis, Univ. of California, Irvine, 2000.

[8] G, Meszaros. xUnit Test Patterns: refactoring test code. Boston: Pearson, 2007.

[9] S. Noikajana and 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 (COMPSAC '09) (pp. 115-123). Washington, DC: IEEE Computer Society, 2009.

[10] OMG, Object Management Group. MDA guide version 1.0.1. OMG, 2003.

[11] OMG, Object Management Group. Object Constraint Language, 2.2. 2010. http://www.omg.org/spec/OCL/2.2/PDF

[12] H. Reza and D. van Gilst. A Framework for Testing RESTful Web Services. Proceedings of the Seventh International Conference on Information Technology (pp. 216-221). IEEE Computer Society, 2010.

[13] T. Silva-de-Souza. Uma Abordagem Baseada em Especificação para Testes de Web Services RESTful. Master Thesis, PPGI, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2012.

[14] M. Utting and B Legeard. Practical Model-Based Testing: A Tools Approach. San Francisco: Morgan Kaufmann Publishers Inc, 2007.

[15] W3C, World Wide Web Consortium. Web Application Description Language. 2009. http://www.w3.org/Submission/wadl/.

[16] W3C, World Wide Web Consortium. Web Service Semantics - WSDL-S. 2005. http://www.w3.org/Submission/WSDL-S/.

[17] W3C, World Wide Web Consortium. Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language. 2007. http://www.w3.org/TR/wsdl20/.

[18] J. Webber, S. Parastatidis, and I. Robinson. REST in Practice. Sebastopol: O'Reilly, 2010.

A model introducing SOAs quality attributes decomposition

Riad Belkhatir, Mourad Oussalah
Department of Computing
University of Nantes
Nantes, France
{riad.belkhatir, mourad.oussalah}@univ-nantes.fr

Arnaud Viguier
Department Research and Development
BeOtic
Rezé, France
arnaud.viguier@beotic.com

Abstract—**Recently, service oriented architecture (*SOA*) has been popularized with the emergence of standards like Web services. Nevertheless, the shift to this architectural paradigm could potentially involve significant risks including projects abandonments. With this in mind, the question of evaluating SOA quality arose. The appearance of methods like ATAM or SAAM propelled software architecture evaluation to a standard stage for any paradigm. However, there still are a number of concerns that have been raised with these methods; in particular their cost in terms of time and money, essentially because of the hand-operated nature of the evaluations conducted. The model proposed in this paper for evaluating SOAs takes as a starting point the McCall model; it allows the whole architecture to be decomposed in three types of quality attributes (factor, criterion and metric).**

Keywords- SOA; factor; criterion; metric

I. INTRODUCTION

Architectural paradigms are design patterns for the structure and the interconnection between software systems [1]. Their evolution is generally linked to the evolution of the technology.

An architectural paradigm defines groups of systems in terms of:

- Model of structure.
- Component and connector vocabularies.
- Rules or constraints on relations between systems [2].

We can distinguish a few architectural paradigms for distributed systems, and, among the most noteworthy ones, three have contributed to the evolution of the concerns. These are chronologically, object oriented architectures (OOA), component based architectures (CBA) and s**ervice oriented architectures (SOA).**

First developers were quickly aware of code repetitions in applications and sought to define mechanisms limiting these repetitions. **OOA** is focused on this concern and its development is one of the achievements of this research. OOA provides great control of the **reusability** (*reusing a system the same way or through a certain number of modifications*) which paved the way to applications more and more complex and consequently to the identification of new limits in terms of granularity. These limits have led to the shift of the concerns towards the **composability** (*combining in a sure way its architectural elements in order to build new systems or composite architectural elements*). Correlatively, the software engineering community developed and introduced **CBA** to overcome this new challenge and thus, the CBA reinforces control of the composability and clearly formalizes the associated processes. By extension, this formalization establishes the base necessary to automation possibilities. At the same time, a part of the software community took the research in a new direction: the **dynamism** concern (*developing applications able to adapt in a dynamic, automatic and autonomous ways their behaviors to answer the changing needs of requirements and contexts as well as possibilities of errors*) as the predominant aspect. In short, **SOA** has been developed on the basis of the experience gained by objects and components, with a focalization from the outset on ways of improving the **dynamism.**

Service oriented architecture is a popular architectural paradigm aiming to model and to design distributed systems [3]. SOA solutions were created to satisfy commercial objectives. This refers to a successful integration of existing systems, the creation of innovating services for customers and cost cutting while remaining competitive. For the purpose of making a system robust, it is necessary that its architecture can meet the functional requirements ("*what a system is supposed to do*"; defining specific behaviors or functions) and the non-functional ones ("*what a system is supposed to be*"; in other words, the quality attributes) [4]. Furthermore, developing an SOA involves many risks, so much the complexity of this technology is notable (*particularly for services orchestration*). First and foremost among these, is the risk of not being able to answer favorably to expectations in terms of quality of services because quality attributes directly derive from business objectives. Multi-million dollar projects, undertaken by major enterprises (Ford, GSA) failed and were abandoned. As these risks are distributed through all the services, the question of evaluating SOA has recently arisen. It is essential to carry out the evaluation of the architecture relatively early in the software lifecycle to save time and money [5]. This is to identify and correct remaining errors that might have occurred after the software design stage and, implicitly, to reduce subsequent risks. Lots of tools have been created to evaluate SOAs but none of them clearly demonstrated its effectiveness

[6]. The model presented in this paper allows evaluating SOAs by combining the computerized approach and the human intervention. We first relate in the section 2 the state of the art and we present the m odel in the section 3. We relate the experimentation led by the lab team in t he section 4 and we conclude the paper with a discussion comparing the past works and our model in the last section.

II. State of the Art

A. *Related works on SOA Evaluation*

There is something far more important with the SOA evaluation as it is the bond between business objectives and the system, insofar as evaluation makes it possible to assess quality attributes of services composing the system [4].

The evaluation relates to:

- Qualitative and quantitative approaches.
- Load prediction associated with evolutions.
- Theoretical limits of a given architecture.

From this perspective, tools and existing approaches have shown their limitations for SOA [6]. We are currently attending the development of a new generation of tools developed by industrialists in a hand-operated way. The scale of the task has brought the academic world to tackle these issues and to try to develop a more formal and generic approach than different existing methods (ATAM, SAAM [6]) to evaluate SOAs.

B. *Evaluation Results*

In concrete terms, SOA evaluations product a report which form and content vary depending on the method used. But, in general terms, the evaluation generates textual information and answers two types of questions [6].

1) Is the architecture adapted to the system for which it has been conceived?

2) Is there any other architecture more adapted to the system in question?

1) It could be said that the architecture is adapted if it favorably responds to the three following points:

a) The system is predictable and could answer to the quality requirements and to the security constraints of the specification

b) Not all the quality properties of the system result directly from the architecture but a lot do; and for those that do, the architecture is deemed suitable if it makes it possible to instantiate the model taking into account these properties.

c) The system could be established using the current resources: the staff, the budget, and the given time before the delivery. In other terms, the architecture is buildable.

This definition will open the way for all future systems and has obviously major consequences. If the sponsor of a system is not able to tell us which are the quality attributes to manage first, well, any architecture will give us the answer [6].

2) A part of SOA evaluation consists in capturing the quality attributes the architecture must handle and to prioritize the control of these attributes. If the list of the quality attributes (*each of which is related to specific business objectives*) is suitable in the sense that at least all the business objectives are indirectly considered, then, we can keep working with the same architecture. Otherwise, it is time to restart from the beginning and to work with a new architecture, more suitable for the system.

C. *Measuring the Quality.*

It has been suggested that software production is out of control because we cannot quantitatively measure it. As a matter of fact, Tom DeMarco (1986) stated that "you cannot control what you cannot measure" [7]. The measurement activity must have clear objectives and a whole set of sectors need to be measured separately to ensure the right management of the software.

1) McCall model

One of the models that have been published is the McCall model in 1977 decomposing quality attributes in three stages. This model led to the IEEE standard: ISO/IEC 9126. A certain number of attributes, called external (applicable to running software), are considered as key attributes for quality. We call them quality factors [6]. These attributes are decomposed in lower level attributes, the internal attributes (which do not rely on software execution), called quality criteria and each criterion is associated to a set of attributes directly measurable and which are called quality metrics.

D. *From past works to our model.*

Current methods of evaluation stop the quality attributes decomposition at the “quality factors step” and remain too vague when it c omes to giving accurate measures to quality. These methods are not precise because they cannot go further in the decomposition and consequently they cannot be automated to th e point of defining a finite value f or each attribute. Our work differs from those existing insofar as we wish to obtain a precise quantitative measurement for each quality factor with our model.

III. The Model Proposed

A. *The model in more details.*

The main idea of the process is to evaluate in three steps the whole architecture from every metric to the set of quality factors obtained after having previously identified the business objectives. Our work is based on the architect point of view and the attributes selected are the ones considered as the most relevant among all existing. The process consists in three principal stages corresponding each to a decomposition step of our quality attributes.

1) We first identified relevant quality factors for our architecture:

a) the CBA is defined with reusability and composability [8]. *Basing on previous analysis, we define the SOA with the reusability, the composability and the dynamism. Moreover,* *there exist a hierarchical ranking propelling "dynamism" on top of SOA concerns, and this is precisely why we chose to especially focus deeply on this quality factor.*

2) Then, we isolate the quality criteria defining them:

*a) We concentrated our work on technical criteria because we adopted the point of view of an architect that is itself a technical stakeholder. In this light, we identified six criteria common to each of our three factors. These technical criteria gather elements having significant impacts on global quality, from the development process to the system produced: the **loose coupling** (potential of dependences reduction between services), the **explicit architecture** (paradigm ability to define clear architectural application views), the **expressive power** (potential of paradigm expression in terms of creation capacity and optionalities), the **communication abstractions** (paradigm capacity to abstract services functions communications), the **upgradability** (paradigm ability to make evolve its services), and the **owner's responsibility** (corresponds to the responsibilities sharing out between services providers and consumers).*

3) And finally we define quality metrics composing each criterion in order to quantify them numerically:

*a) Our previous work allowed to conclude that the "**loose coupling**" criterion is of biggest importance for the quality factor "dynamism"* [9]. *We found three quality metrics for the latter which must be considered for the last stage of our model (the semantic coupling: {high, low or non-predominant} based on the high-level description of a service defined by the architect, the syntactic coupling: {high, low} measures dependencies in terms of realization between abstract services and concrete services and the physical coupling: {γ, β and α with $0 \leq \gamma \leq \beta \leq \alpha$} focusing on the implementation of the service). These metrics shall make it possible to identify physical dependencies between concrete services.*

B. Coefficients

Coefficients assigned to the factors will depend on the company needs. Our works led us to conclude that for SOA and the three factors we worked with, we would allocate a coefficient of '3' for the "dynamism" whereas we would affect the value '2' for the "reusability" and the "composability". With regards to the second step, our works led to list the six technical quality criteria chosen under three distinct levels of acceptance, α, β and γ at which we assign respectively the values '3, 2 and 1'; We allocated the "loose coupling", the "upgradability" and the "communication abstraction" with the value '3'. The coefficient '2' goes for the "owner's responsibility" and "explicit architecture" criteria and '1' for the "expressive power". And finally, the three metrics studied may be all assigned to the value '1' meaning that they are equally important for calculating the global coupling of SOAs. These coefficients will be used as a basis for the following section. They have been affected to quality attributes as an example; however, these latter have been chosen according to the principle of proportionality validated by the lab-team. We can select other impact coefficients providing that we keep the same proportionality between the quality-attributes considered.

IV. EXPERIMENTATION

For the experimentation, we tempted to quantitatively measure the key quality attributes discussed in the previous sections of this paper; notably, the quality factor "dynamism", the "loose coupling" criteria and the "physical, syntactic and semantic coupling" metrics. That being said, it is important to note that the SOAQE method must be reproduced for every quality factor identified after having analyzed the objectives of the company and the set of criteria and metrics belonging to that quality factor. Taking as a starting point an existing formula of the field of "Preliminary analysis of risks" (see formula 1.1) [10] our works led to the identification of a mathematical formula (see formula 1.2) combining the three couplings studied: semantic, syntactic and physical.

NB: The simplified formula (see formula 1.1) usually used in the automotive industry, makes it possible to measure the default risk of a car component A is the Criticality of the car component, B is the Probability of occurrence of a failure on this component and C is the Probability of non-detection of this failure.

We associate this concept of risk with our vision of the coupling. Correlatively, the quintessence of the coupling is the expression of the dependences which can exist between two elements and the principle of dependence defines that one element cannot be used without the other. Reducing the risk that the role defined by a service cannot be assured anymore is decreasing the dependence of the application in relation to this service and thus reducing its coupling. The calculation of this risk takes into account all the characteristics influencing the coupling by redefining the three variables A, B and C according to the semantic, syntactic and physical couplings. The global coupling corresponds to the sum of the three couplings calculated individually beforehand. The lower this result is, the more the coupling is weak.

NB: The criticality $A \in [(a),(b),(c)]$ is affiliated to the semantic coupling. 'a' if the service is only associated to non predominant couplings, 'b' for non predominants and low couplings and 'c' for non predominants, low and high couplings, while 'Ps' is the probability of failure of a service.

$$R = A * B * C \qquad (1.1)$$

$$Coupling = \left\{ \{(a),(b),(c)\} \right\} * \left\{ \left(\left(\prod_{k=1}^{\lambda} \prod_{i=1}^{N} \sum_{j=1}^{i} ((P_s)_{kij})^{\alpha_{kij}} \right) * C_{phys} \right) \right\} * \left\{ P_{ndetect} \right\} \qquad (1.2)$$

Figure 1: Defaut risk of a car component (1.1) and global coupling of an architecture (1.2) formulas.

This generic coupling formula can directly be used to quantify the quality of the architecture by weighting up each of the attributes concerned by means of the coefficients isolated after having organized the attributes according to their importance. Indeed, as we already specified in section 2, we cannot automate this operation and define continuously the same coefficients for all the architectures considered because this operation is specific to the business objectives of the company. By applying to known quality attributes the coefficients determined in the section III.*B*, we obtain the following tree:

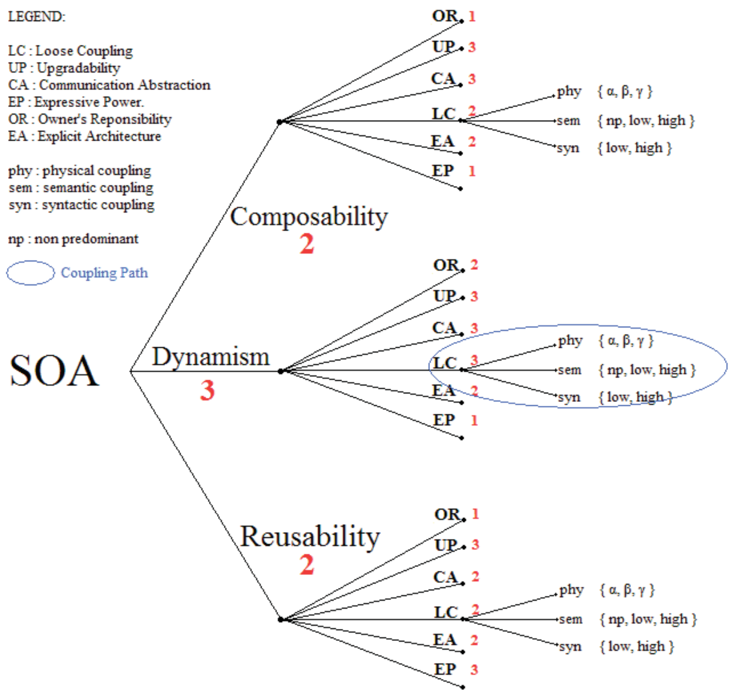


Figure 2: SOA attributes tree weighted with means of coefficients

Thus, according to the previous figure 2, we can establish that the quantitative measure of the quality of an SOA corresponds to the sum of the quality factors dynamism, reusability and composability, all three affected by their respective coefficients:

The following formula allows calculating the whole quality of an SOA.

$$SOA = 3Dynamism * 2Reusability * 2Composability$$

$$SOA = 3(2OR + 3UP + 3CA + 3(1 - LC) + 2EA + EP) * 2(OR + 3UP + 2CA + 2(1 - LC) + 2EA + 3EP) * 2(OR + 3UP + 3CA + 2(1 - LC) + 2EA\ EP)$$

$$\boxed{\begin{aligned} SOA = {} & 3(2OR + 3UP + 3CA + 3\,[1\text{-}(\ \{(a),(b),(c)\} * \left(\left(\prod_{k=1}^{\lambda}\prod_{i=1}^{N}\sum_{j=1}^{i}((P_s)_{kij})^{\alpha_{kij}}\right) * C_{phys}\right) * P_{ndetect}\ )\,] + 2EA + EP) * \\ & 2(OR + 3UP + 2CA + 2\,[1\text{-}(\ \{(a),(b),(c)\} * \left(\left(\prod_{k=1}^{\lambda}\prod_{i=1}^{N}\sum_{i=1}^{i}((P_s)_{kij})^{\alpha_{kij}}\right) * C_{phys}\right) * P_{ndetect}\ )\,] + 2EA + 3EP) * \\ & 2(OR + 3UP + 3CA + 2\,[1\text{-}(\ \{(a),(b),(c)\} * \left(\left(\prod_{k=1}^{\lambda}\prod_{i=1}^{N}\sum_{j=1}^{i}((P_s)_{kij})^{\alpha_{kij}}\right) * C_{phys}\right) * P_{ndetect}\ )\,] + 2EA + EP) \end{aligned}}$$

NB: the lower the loose coupling result is, the more the coupling is weak. Conversely, the higher the architecture quality result is, the more the quality is good; the result of each criterion is expressed in percentage, this is why we subtract to 1 the result found.

For any architecture considered, we are able to determine a finite value for the loose coupling criteria, the remaining work consists in defining a way to calculate the five others criteria in order to isolate a finite value for the quality.

V. Discussion

Because SOA implies the connectivity between several systems, commercial entities and technologies: some compromises regarding the architecture must be undertaken, and this, much more than for systems with a single application where technical difficulties prevail. Forasmuch as the decisions about SOA tend to be pervasive and, consequently, have a significant impact on the company; setting an evaluation of the architecture early in the li fe of the software is particularly crucial. During software architecture evaluations, we weigh the relevance of each problematic associated to the d esign after having evaluated the importance of each quality attribute requirement. The results obtained when e valuating software architectures with existing methods (ATAM, SAAM) are often very different and none of these latter carries out it accurately (for example, SAAM does not provide any clear quality metric for the architectural attributes analyzed [11]). We know the causes of this problem: most methods of analysis and automatic quality evaluation of software systems are carried out from the source code; whereas, with regard to evaluation cases of architectural models, the analysis is conducted based on the code generated from the model. From this code, there exist calculated metrics, more or less complex, associated with algorithms, methods, objects or relations between objects. From an architectural point of view, these techniques can be indicated of low level, and can be found out of step with projects based on new complex architectures. The finality of our work is to design a conceptual framework and, in fine, a semi-automated prototype called SOAQE *(taking as a starting point, past methods such as ATAM or SAAM)* which could quantify with an accurate value the quality of the whole service oriented architecture.

VI. References

[1] P.S.C. Alencar, D.D. Cowan, T. Kunz and C.J.P. Lucena, "A formal architectural design patterns-based approach to software understanding," Proc. Fourth workshop on program comprehension, 2002.

[2] D. Garlan and M. Shaw, An introduction to software architecture, CMU/SEI-94-TR-21, ESC-TR-94-21, 1994.

[3] H.K. Kim, "Modeling of distributes systems with SOA and MDA," IAENG, International Journal of Computer Science, ISSN 1819-656X, Volumr: 35; Issue: 4; Start page: 509, 2008.

[4] O. Lero, P. Merson and L. Bass, "Quality attributes and service oriented architectures" SDSOA 2007, 2007.

[5] P. Bianco, R. Kotermanski and O. Merson, "Evaluating a service oriented architecture" CMU/SEI-2007-TR-015, Carnegie Mellon University, Software Engineering Institute, 2007.

[6] P. Clements, R. Kazman and M. Klein, Evaluating Software Architectures: Methods and case studied, published by Addison-Wesley Professional, 2001.

[7] T. Demarco, Controlling software projects: management, measurement and estimates, Prentice Hall, 296 pages, 1986.

[8] I. Crnkovic, M. Chaudron and S. Larsson, "Component-based development process and component lifecycle" ICSEA'06, International Conference on Software Engineering Advances, 2006.

[9] A. Hock-Koon, "Contribution à la compréhension et à la modélisation de la composition et du couplage faible de services dans les architectures orientées services" Thesis (PhD). University of Nantes, 2011.

[10] Y. Mortureux, Preliminary risk analysis. Techniques de l'ingénieur. Sécurité et gestion des risques, SE2(SE4010):SE4010.1–SE4010.10, 2002.

[11] M.T. Ionita, D.K. Hammer and H. Obbink, "Scenario-based software architecture evaluation methods: an overview", ICSE 2002, 2002.

Software as a Service: Undo

Hernán Merlino
Information Systems Research Group. National University of Lanús, Buenos Aires, Argentine.
hmerlino@gmail.com

Oscar Dieste
Empirical Software Research Group (GrISE). School of Computer Science. Madrid Politechnic University. Spain.
odieste@fi.upm.es

Patricia Pesado
PhD Program on Computer Science. Computer Science School. National University of La Plata. Argentine.
ppesado@lidi.info.unlp.edu.ar

Ramon García-Martínez
Information Systems Research Group. National University of Lanús, Buenos Aires, Argentine.
rgarcia@unla.edu.ar

Abstract— **This paper proposes a highly automated mechanism to build an undo facility into a new or existing system easily. Our proposal is based on the observation that for a large set of operators it is not necessary to store in-memory object states or executed system commands to undo an action; the storage of input data is instead enough. This strategy simplifies greatly the design of the undo process and encapsulates most of the functionalities required in a framework structure similar to the many object-oriented programming frameworks.**

Keywords- Undo Framework; Software as a Services; and Usability component.

I. INTRODUCTION

It is hard to build usability into a system. One of the main reasons is that this is usually done at an advanced stage of system development [1], when there is little time left and the key designed decisions have already been taken. Usability patterns were conceived with the aim of making usable software development simpler and more predictable [2]. Usability patterns can be defined as mechanisms that could be used during system design to provide the software with a specific usability feature [1]. Some usability patterns defined in the literature are: Feedback, Undo/Cancel, Form/Field Validation, Wizard, User profile and Help [3]. The main stumbling block for applying these patterns is that there are no frameworks or even architectural or designed patterns associated with the usability patterns. This means that the pattern has to be implemented ad hoc in each system. Ultimately, this implies that (1) either the cost of system development will increase as a result of the heavier workload caused by the design and implementation of the usability features or, more likely; (2) many of these usability features (Undo, Wizard, etc.) will be left out in an attempt to reduce the development effort.

The goal of this paper is to develop a framework for one of the above usability patterns, namely, the undo pa ttern. The undo pattern provides the functionality necessary to undo actions taken by system users. Undo is a common usability features in the literature [4]. This is m ore than enough justification for dealing with this pattern first. There are other, more technical grounds to support the decision to tackle undo in first place. One of the most important is undoubtedly that undo shares much of its infrastructure (design, code) with other patterns. Redo and cancel are obvious cases, but it also applies to apparently unrelated patterns, like feedback and wizard.

Several authors have proposed alternatives of undo pattern, these alternatives focus on particular applications, notably document editors [5][6] although the underlying concepts are easily exportable to other domains. However, these proposals are defined at high level, without an implementation (or design) reusable in different types of systems. These proposals therefore do not solve the problem of introduction of usability features in software

In this paper, we present a new approach for the implementation of Undo pattern. Our proposal solves a subset of cases (stateless operations) in a highly efficiently manner. The importance of having an automated solution of those is that they are the most frequents operations occur in information systems.

We have implemented the framework using Software as a Service (SaaS). For this class of development we have developed a framework similar to other such as Spring [7] or Hibernante [8] that allows to build the undo easy into a system (that we term "host application"). Furthermore, in host application, it's only need to i nclude a few modifications in code, and this creates a lower propensity to introduce bugs in the code and allows inclusion of it in a more simple developed system.

This article is structured as follows. Section 2 describes the state of the art regarding the implementation of undo. Section 3 presents the undo infrastructure, whereas Section 4 describes undo infrastructure. Section 5 shows a proof of concept of the proposed framework. Finally, Section 6 briefly discusses and presents the main contributions of our work.

II. BACKGROUND

Undo is a very widespread feature, and is prominent across the whole range of graphical or textual editors, like, for example, word processors, spreadsheets, graphics editors, etc. Not unnaturally a lot of the undo-related work to date has focused on one or other of the above applications. For example, [6] and Baker and Storisteanu [9] have patented two methods for implementing undo in document editors within single-user environments.

There are specific solutions for group text editors that support undo functionality such as in Sun [10] y Chen and Sun [11] and Yang [12]. The most likely reason for the boom of work on undo in the context of document editors is its relative simplicity. Conceptually speaking, an editor is a container accommodating objects with certain properties (shape, position, etc.). Consequently, undo is relatively easy to implement, as basically it involves storing the state of the container in time units i, i+1, …, i + n. Then when the undo command is received, the container runs in reverse i + n, i + n-1, i.

A derivation of the proposed solutions for text editors is an alternative implementation of undo for email systems like Brown and David [13], these solutions are only for text editors and email systems and applications that are built considering undo functionality from the design.

The problems of undo in multi-user environments have also attracted significant attention. Both Qin [15] and Abrams and Oppenheim [14] have proposed mechanisms for using undo in distributed environments, and Abowd and Dix [4] proposed a formal framework for this field.

In distributed environments, the solution has to deal with the complexity of updates to shared data (basically, a history file of changes) [15].

Several papers have provided insight on the internal aspects of undo, such [16], who attempted to describe the undo process features. Likewise, Berlage [17] proposed the construction of an undo method in command-based graphical environments, Burke [18] created an undo infrastructure, and Korenshtein [19] defined a selective undo.

There has been work done on multi-level models for Undo where each action for a system is defined as a discreet group of commands performed, where each command represents a requested action by the user, this is a really valid approximation because defined as a discreet group of commands, the system could be reverted to any previous stage, only performing the actions the other way round; here a difference can be found between the theory and the practice, regarding the first one it is true that is possible to go back to any previous stage of the system if there is the necessary infrastructure for the Undo, but actually the combination of certain procedures performed by the user or a group of them could be impossible to be solved related to expected response time. For this reason the implementation of the Undo process must complete these possible alternatives with regards to the command combinations performed by the user or users.

Another important aspect which has been worked out is the method of representation of the actions performed by the users in Washizaki and Fukazawa [20], a dynamic structure of commands is presented and it represents the history of commands implemented.

The Undo model representation through graphs has been widely developed in Berlage [17] present a distinction between the linear and nonlinear undo, the nonlinear approach is represented by a tree graph, where you can open different branches according to user actions. Edwards [21] also presented a graph structure where unlike Berlage [17] these branches can be back together as the actions taken. Dix [22] showed a cube-shaped graph to represent history of actions taken. Edwards [23] actions are represented in parallel. It has also used the concept of Milestoning and Rollback [24] to manage the log where actions temporarily stored. Milestoning is a logical process which makes a particular state of the artifacts stored in the log; and rollback is process of returning back the log to one of the points of Milestoning. All these alternative representation of the commands executed by users are valid, but this implementation is not a simple task, because create a new branch and join two existing branches is not a trivial action, because you must know all possible ways that users can take; by this it may be more advisable to generate a linear structure, that can be shared by several users, ordered by time, this structure can be a queue, which is easy to deploy and manage.

Historically frameworks that have been used to represent the Undo only have used the pattern Command Processor [25], Fayard, Shumidt [26] and Meshorer [27]. This serves to keep a list of commands executed by the user, but it is not enough to create a framework that is easy to add to existing systems, As detailed below using service model allows greater flexibility for the undo process integration in an application, this approach allow a greater degree of complexity in the process of allowing Undo handle different configurations.

Undo processes has been associated to exception mechanisms to reverse the function failed [28] these are only invoked before the request fails and the user, these are associated with a particular set of applications.

Patents, like the method for building an undo and redo process into a system, have been registered [29]. Interestingly, this paper presents the opposite of an undo process, namely redo, which does again what the undo previously reverted. Other authors address the complexities of undo/redo as well. Thus, for example, Nakajima and Wash [30] define a mechanism for managing a multi-level undo/redo system, Li [31] describes an undo and redo algorithm and Martinez and Rhan [32] present a method for graphically administering undo and redo, based primarily on the undo method graphical interface.

The biggest problem with the above works is that, again, they are hard to adopt in software development processes outside the document editor domain. The only noteworthy exception to this is a design-level mechanism called Memento [33]. This pattern restores an object to a previous state and provides an implementation-independent mechanism that can be easily integrated into a system. The downside is that this pattern is not easy to build into an existing system. Additionally, Memento only restores an object to a previous state; it does not consider any of the other options that an undo pattern should include.

The solutions presented are optimized for particular cases and are difficult to apply to other domains; on the other hand, it is necessary to include a lot of code associated with Undo in host application.

III. THEORETICAL JUSTIFICATION OF UNDO FRAMEWORK

Before describing proposed Undo Framework, and its implementation as SaaS, theoretical foundations that demonstrate the correctness of our approach. This will be done in two steps; first we will describe how to undo operations that do not depend on its state, the procedure to undo these operations consist in reinjection input data at time t-1, second we prove that reinjection input always produces correct results.

A. *Initial Description*

The most commonly used option for developing an undo process is to save the states of objects that are liable to undergo an undo process before they are put through any operation; this is the command that changes the value of any of their attributes. This method has an evident advantage; the system can revert without having to enact a special-purpose process; it is only necessary to remove and replace the current in-memory objects with objects saved previously.

This approach is a simple mechanism for implementing the undo process, although it has some weaknesses. On one hand, saving all the objects generates quite a heavy system workload. On the other hand, developer's need to create explicitly commands for all operations systems. Finally, the system interfaces (mainly the user interface) have to be synchronized with the application objects to enact an undo process. This is by no means easy to do in monolithic systems, but, in modern distributed computer systems, where applications are composed of multiple components all running in parallel (for example, J2EE technology-based EJB), the complications increase exponentially.

There is a second option for implementing an undo process. This is to store the operations performed by the system instead of the changes made to the objects by these operations. In this case, the undo would execute the inverse operations in reverse order. However, this strategy is seldom used for two reasons. On one hand, except for a few exceptions like the above word processing or spreadsheet software, applications are seldom designed as a set of operations. On the other hand, some operations do not have a well-defined inverse (imagine calculating the square of a table cell; the inverse square could be both a positive and a negative number).

The approach that we propose is based on this last strategy, albeit with a simplified complexity. The key is that, in any software system whatsoever, the only commands processed that are relevant to the undo process are the ones that update the model data (for example, a data entry in a field of a form that updates an object attribute, the entry of a backspace character that deletes a letter of a document object, etc.). In most cases, such updates are idempotent, that is, the effects of the entry do not depend on the state history. This applies to the form in the above example (but not, for example, to the word processor). When the updates are idempotent, neither states of the objects in the model nor the executed operations has to be stored, and the list of system inputs is only required. In other words, executing an undo at time t is equivalent to entering via the respective interface (usually the user interface) the data item entered in the system at time t-2. Figure 1 shows an example of this approach. At time t, the user realizes that he has made a mistake updating the name field in the form, which should contain the value John not Sam. As a result, he wants to revert to the value of the field that the form had at time t-1. To do this, it is necessary (and enough) to re-enter the value previously entered at time t-2 in the name field.

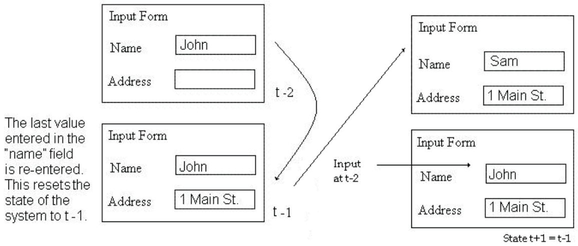


Figure 1. Undo sequence.

Unless the updates are idempotent, this strategy is not valid (as in the case of the word processor, for example), and the original strategy has to be used (that is, store the command and apply its inverse to execute the undo). However, the overwhelming majority of cases executed by a system are idempotent, whereas the others are more of an exception.

Consequently, the approach that we propose has several benefits: (1) the actual data inputs can be processed fully automatically and transparently of the host application; (2) it avoids having to deal with the complexity of in-memory objects; (3) the required knowledge of system logic is confined to commands, and (4), finally, through this approach, it is possible to design an undo framework that is independent of the application and, therefore, highly reusable.

B. *Formal Description*

The following definitions and propositions are used to proof (in an algebraic way) that UNDO process (UNDO transformation) may be built under certain process (transformation) domain constrains.

Definition 1. Let $E = \{\varepsilon_j^i / \varepsilon_j$ is a data structure$\}$ be the set of all data structures.

Definition 2. Let ε_j^i be the instance *i* of data structure ε_j belonging to E.

Definition 3. Let $\varepsilon_j^C = \{ \varepsilon_j^i /\ \varepsilon_j^i$ is an instance *i* of the structure $\varepsilon_j\}$ be the set of all the possible instances of data structure ε_j.

Definition 4. Let $o_\tau^{\varepsilon j}$ be a transformation which verifies $o_\tau^{\varepsilon j} : \varepsilon_j^C \rightarrow \varepsilon_j^C$ and $o_\tau^{\varepsilon j}(\varepsilon_j^i) = \varepsilon_j^{i+1}$.

Definition 5. Let ε_j^{Cr} be a constrain of ε_j^C defined as $\varepsilon_j^{Cr} = \{ \varepsilon_j^i /\ \varepsilon_j^i$ is an instance *i* of the data structure ε_j which verifies $o_\tau^{\varepsilon j}(\varepsilon_j^{i-1}) = \varepsilon_j^i\}$

Proposition 1. If $o_\tau^{\varepsilon j} : \varepsilon_j^C \rightarrow \varepsilon_j^{Cr}$ then $o_\tau^{\varepsilon j}$ is bijective. Proof: $o_\tau^{\varepsilon j}$ es injective by definition 4, $o_\tau^{\varepsilon j}$ is surjective by definition 5, then $o_\tau^{\varepsilon j}$ is bijective for being injective and surjective. QED.

Proposition 2. If $o_\tau^{\varepsilon j} : \varepsilon_j^C \rightarrow \varepsilon_j^{Cr}$ then has inverse. Proof: Let $o_\tau^{\varepsilon j}$ be bijective by proposition 1, then by usual algebraic properties $o_\tau^{\varepsilon j}$ has inverse. QED.

Definition 6. Let o_τ be the set of al transformations $o_\tau^{\varepsilon j}$.

Definition 7. Let Φ be the operation of composition defined as usual composition of algebraic transformations.

Definition 8. Let Σ be the service defined by structure < E^{Σ} , o_τ^{Σ} , Φ > where $E^{\Sigma} \subseteq E$ and $o_\tau^{\Sigma} \subseteq o_\tau$.

Definition 9. Let X = $o_\tau^{\varepsilon j1}$ Φ $o_\tau^{\varepsilon j2}$ Φ ... Φ $o_\tau^{\varepsilon jn}$ be a composition of transformations which verifies $o_\tau^{\varepsilon ji}$: $\varepsilon_j^{C} \rightarrow \varepsilon_j^{Cr}$ for all i :1...n. By algebraic construction X : $\varepsilon_j^{C} \rightarrow \varepsilon_j^{Cr}$.

Proposition 3. The composition of transformations X ha s inverse and is bijective. Proof: Let be X = $o_\tau^{\varepsilon j1}$ Φ $o_\tau^{\varepsilon j2}$ Φ ... Φ $o_\tau^{\varepsilon jn}$. For all i:1...n verifies $o_\tau^{\varepsilon ji}$ has inverse by proposition 2. Let $[o_\tau^{\varepsilon ji}]^{-1}$ be the inverse transformation of $o_\tau^{\varepsilon ji}$, by usual algebraic properties $[o_\tau^{\varepsilon ji}]^{-1}$ is bijective. Then it is possible to compose a transformation X^{-1} = $[o_\tau^{\varepsilon jn}]^{-1}$ Φ $[o_\tau^{\varepsilon jn-1}]^{-1}$ Φ ... Φ $[o_\tau^{\varepsilon j1}]^{-1}$. The transformation X^{-1} is bijective by being composition of bijective transformations. Then transformation X^{-1} : $\varepsilon_j^{Cr} \rightarrow \varepsilon_j^{C}$ exists and is the inverse of X. QED.

Definition 10. Let UNDO be the X^{-1} transformation of X.

IV. STRUCTURE OF UNDO FRAMEWORK

In this section, we will describe our proposal for designing the undo pattern using SaaS to implement the replay of data.

A. Undo Service Architecture

Figure 2 represents the service Undo infrastructure, a high-level abstraction of the architecture. Undo service has 3 modules, (a) Undo Business Layer, (b) Undo Application Layer (c) Undo Technology Layer.

Undo Business Layer is responsible for creating, maintaining and deleting applications that will access the undo service. An a pplication that could access to se rvice must execute following steps: (a) creating application unique identifier, this should be attached to each message that is sent to the service, (b) creation of user profile identifier, this must be attached to each message that is sent to the service, once defined two identifiers, host application may immediately use undo service

All these added to the header data set that can be invoked by the user for later retrieval, enable the service to h andle different applications at the same time, within an application users can manage their own recovery without interfering lists, plus each user can manage their own separate lists per interface; the service giving maximum flexibility for every application.

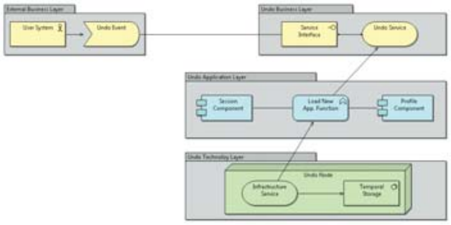


Figure 2. Undo infrastructure

B. Operation of Undo Framework

Fig. 3 details process of send and get data to service, first we described undo receives data service from external system, at this point is where you start the process that ended with the injection of data re be invoked by the external system. In the External Layer, the user application generates an event that triggers an action likely to be overturned, this creates an Undo Service invocation, this is received by the service interface that is plotted on the Undo Abstract Layer, this action fires a set of processes:

(i) Check current user session, this start with Validate Session and Profile, this process communicates with the Undo Application Layer, with f unction that processes Validate Undo Service Session and Profile. This service is based on two components responsible for validation and maintenance of active user sessions and profile´s user, Session component is responsible for validating whether the session with which you access the service is active, component Profile is responsible for validating invocation of the temporary storage. Both components communicate with lower-level layer called Layer Undo Technology, this is basic infrastructure for Undo service, which consists of a processing unit and data storage.

(ii) After that, the validation process begins to check if host application has access to temporary storage, this process communicates with Validate Undo Data process, and it is responsible for validating the data to be stored, first validates that host application is active, if so, host application obtains credentials to use. If the process is successful the user is returned a successful update code, if an error occurs, it returns an error code also asynchronously, and with external system code decides if it generates exception or continues with the normal flow.

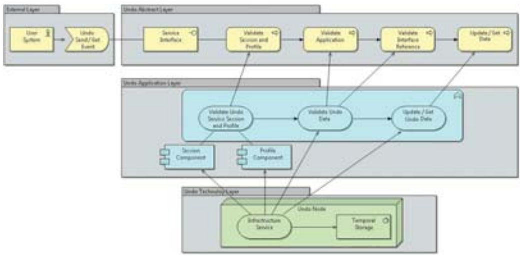


Figure 3. Undo receives data service

At last we described process which undo data service return stored temporarily to external system, this is where it describes the beginning of re injection data by the external system for the service provided. In the same way explained above, Layer External triggers an event that generates a r equest for data stored, this process is divided into two stages:

(i) Charge of the validation of the application, which has the same activities as described in the process of Undo Send Data,

(ii) Retrieve all values that have been stored for the tuple, application and interface. Return of this process to External Layer is the list sorted in reverse with all values stored, service provide option to request only the last value stored. If event failure, external application receives an asynchronously error code.

V. Conclusions

In this paper we have proposed the design of an undo framework to build the undo functionality into any software application whatsoever. The most salient feature of this framework is the type of information it stores to be able to undo the user operations: input data instead of in-memory object states or commands executed by the system. This lessens the impact of building the framework into the target application a great deal.

Building an Undo Service has some significant advantages with respect to Undo models presented, first of all the simplicity of inclusion in a host application under construction or existing, you can see in the proof of concept. Second the independence of service in relation to the host application allows the same architectural model to provide answers to different applications in different domains. Construction of a service allows to Undo be a complex application, with possibility of include analysis for process improvement, as described in the next paragraph it is possible to detect patterns of invocation of Undo in different applications.

Further work is going to bring: (a) creation of a pre-compiler, (8) automatic detection of fields to store, (c) extend the framework to other platforms.

VI. References

1. Ferre, X., Juristo, N., Moreno, A., Sanchez, I. 2003. A Software Architectural View of Usability Patterns. 2nd Workshop on Software and Usability Cross-Pollination (at INTERACT'03) Zurich (Switzerland)
2. Ferre, X; Juristo, N; and Moreno, A. 2004. Framework for Integrating Usability Practices into the Software Process. Madrid Polit. University.
3. Juristo, N; Moreno, A; Sanchez-Segura, M; Davis, A. 2005. Gathering Usability Information through Elicitation Patterns.
4. Abowd, G.; Dix, A. 1991. Giving UNDO attention. University of York.
5. Qin, X. y Sun, C. 2001. Efficient Recovery algorithm in Real-Time and Fault-Tolerant Collaborative Editing Systems. School of computing and Information Technology Griffith University Australia.
6. Bates, C. and Ryan, M. 2000. Method and system for UNDOing edits with selected portion of electronic documents. PN: 6.108.668 US.
7. Spring framework. http://www.springsource.org/.
8. Hibernate framework. http://www.hibernate.org/.
9. Baker, B. and Storisteanu, A. 2001. Text edit system with enhanced UNDO user interface. PN: 6.185.591 US.
10. Sun, C. 2000. Undo any operation at time in group editors. School of Computing and Information Technology, Griffith University Australia.
11. Chen, D; Sun, C. 2001. Undoing Any Operation in Collaborative Graphics Editing Systems. School of Computing and Information Technology, Griffith University Australia.
12. Yang, J; Gu, N; Wu, X. 2004. A Documento mark Based Method Supporting Group Undo. Department of Computing and Information Technology. Fudan University.
13. Brown, A; Patterson, D, 2003. Undo for Operators: Building an Undoable E-mail Store. University of California, Berkeley. EECS Computer Science Division.
14. Abrams, S. and Oppenheim, D. 2001. Method and apparatus for combining UNDO and redo contexts in a distributed access environment. PN: 6.192.378 US.
15. Berlage, T; Genau, A. 1993. From Undo to Multi-User Applications. German National Research Center for Computer Science.
16. Mancini, R., Dix, A., Levialdi, S. 1996. Reflections on UNDO. University of Rome.
17. Berlage. T. 1994. A selective UNDO Mechanism for Graphical User Interfaces Based On command Objects. German National Research Center for Computer Sc.
18. Burke, S. 2007. UNDO infrastructure. PN: 7.207.034 US.
19. Korenshtein, R. 2003. Selective UNDO. PN: 6.523.134 US.
20. Washizaki, H; Fukazawa, Y. 2002. Dynamic Hierarchical Undo Facility in a Fine-Grained Component Environment. Department of InformaTION AND Computer Science, Waswda University. Japan.
21. Edwards, W; Mynatt, E. 1998. Timewarp: Techniques for Autonomous Collaboration. Xerox Palo Alto Research Center.
22. Dix, A; Mancini, R; Levialdi, S. 1997. The cube – extending systems for undo. School of Computing, Staffordshire University. UK.
23. Edwards, W: Igarashi, T; La Marca, Anthony; Mynatt, E. 2000. A Temporal Model for Multi-Level Undo and Redo.
24. O´Brain, J; Shapiro, M. 2004. Undo for anyone, anywhere, anytime. Microsoft Res..
25. Buschmann, F; Meunier, R; Rohnert, H; Sommerlad, P; Stal, M. 1996. Pattern-Oriented Software Architecture: A System Of Patterns. John Wiley & Sons.
26. Fayad, M.; Shumidt, D. 1997. Object Oriented Application Frameworks. Comunications of the ACM, 40(10) pp 32-38.
27. Meshorer, T. 1998. Add an undo/redo function to you Java app with Swing. JavaWord, June, IDG Communications.
28. Shinnar, A; Tarditi, D; P lesko, M; Steensgaard, B. 2004. Integrating support for undo with exception handling. Microsoft Research.
29. Keane, P. and Mitchell, K. 1996. Method of and system for providing application programs with an UNDO/redo function. PN:5.481.710 US.
30. Nakajima, S., Wash, B. 1997. Multiple level UNDO/redo mechanism. PN: 5.659.747 US.
31. Li, C. 2006. UNDO/redo algorithm for a computer program. PN: 7.003.695 US.
32. Martinez, A. and Rhan, M. 2000. Figureical UNDO/redo manager and method. PN: 6.111.575 US.
33. Gamma, E., R. Helm, R. Johnson, and J. Vlissides. 1994. Design Patterns: Elements of Reusable Object-Oriented Software, Addison- Wesley.

A Petri Net Model for Secure and Fault-Tolerant Cloud-Based Information Storage

Daniel F. Fitch and Haiping Xu
Computer and Information Science Department
University of Massachusetts Dartmouth, North Dartmouth, MA 02747, USA
{daniel.fitch, hxu}@umassd.edu

***Abstract*—Cloud computing provides a promising opportunity for both small and large organizations to transition from traditional data centers to cloud services, where the organizations can be more concerned with their applications, services, and data rather than the underlying network infrastructures and their associated cost. There are major concerns, however, with data security, reliability, and availability in the cloud. In this paper, we address these concerns by proposing a novel security mechanism for secure and fault-tolerant cloud-based information storage. We present a formal model of the security mechanism using colored Petri nets (CPN). The model utilizes multiple cloud service providers as a cloud cluster for information storage, and a service directory for management of the cloud clusters including service query, key management, and cluster restoration. Our approach not only supports maintaining the confidentiality of the stored data, but also ensures that the failure or compromise of an individual cloud provider in a cloud cluster will not result in a compromise of the overall data set.**

Keywords-Cloud computing; information storage; data security; fault tolerant; colored Petri nets; formal modeling and verification.

I. INTRODUCTION

As the Internet continues to evolve, service-oriented systems are becoming more widely adopted by large companies and government into their computing platforms. Cloud computing extends this concept, allowing for access to powerful, ubiquitous and reliable computing to the general public through the use of web services and application programming interfaces (API). Although there is a large push towards cloud computing, there is a lack of work having been done in regards to data security, ownership and privacy in cloud computing. A survey conducted by the US Government Accountability Office (GAO) states that "22 of 24 major federal agencies reported that they were either concerned or very concerned about the potential information security risks associated with cloud computing" [1]. Due to the infancy of cloud computing, there are not many standards or best practices in terms of securing data in the clouds. Besides a few major companies investing into the cloud, there are also numerous startups and smaller companies attempting to become cloud providers. For these smaller entities, there are no guarantees that they are following or have the resources available to follow best practices for securing their data centers. In addition, services change frequently as product offerings are developed and discontinued, leaving users of the service scrambling to find alternatives, forced to take a migration path etched by the provider, or stuck using a service that is no longer being developed, refined, and patched. In an enterprise environment, the issues that are plaguing cloud computing would be considered unacceptable. If the corporate world is to adopt cloud computing, it must guarantee that the stored data is secure, stable, and available in the cloud.

Personal Information (PI), such as credit card information, that falls under Payment Card Industry (PCI) data security standards legislation, or medical records that falls under the Health Insurance Portability and Accountability Act (HIPAA), are especially at question regarding if and how exactly these pieces of data can be stored and managed utilizing cloud computing. Although there are some attempts to address HIPAA compliance in the cloud [2], there exist no widely accepted best practices or clear recommendations as to how this data can be stored in the cloud. Currently, users are advised to seek their own legal counsel on this matter, with the provider offering no liability for misguiding or incorrect advice. In addition, legislative acts, such as HIPAA, were developed with traditional network architectures in mind, with numerous regulations regarding physical facilities, employee best practices, and operating system best practices. In a cloud environment, all or at least most of these implementation details are hidden from the cloud consumers, so companies that fall under HIPAA regulations do not have direct influence or authority over these compliance details. Procedures and requests can be specified during the contract creation time between a cloud provider and an enterprise, but this would require complex negations and audit procedures that the cloud provider may not be equipped for or willing to follow. There are efforts in the security industry to certify certain cloud providers as being compliant with legislative mandates for handling PI, but we can find no established practice at this time. In this paper, we develop a security model that addresses these issues of cloud computing, easing concerns of legislatures and enterprise of storing data in the cloud. The proposed model can be adopted to serve as an equivalent alternative to enterprise controlled facility, personnel and infrastructure mandates.

Although cloud computing is still in its infancy, there has been a considerable amount of work on data security and federation for distributed data, in which this work is related to. Goodrich *et al.* explored efficient authentication mechanisms for web services with unknown providers [3]. In their approach, they utilized a third party notary service that could ensure users the trustworthiness of the service providers. Weaver studied the topic of exploring data security when using web services [4]. In his approach, he placed a layer of authentication and authorization services between the clients and the web services they were trying to access in order to authorize users. He also explored the issue of federation to manage trust relationships among services, which could be extended towards securing cloud computing. Santos *et al.* provided efforts to establish a secure and trusted cloud computing environment [5]. They, however, assumed that providers could prevent physical attacks to their servers, which might not be true in the case of a poorly vetted employee or a poorly designed facility. In our approach, we focus on data security, redundancy, and privacy issues in cloud computing. We develop a formal model of information storage that utilizes a cloud cluster with multiple cloud providers to leverage the benefits of storing data in the cloud while minimizing the risks associated with storing data in an offsite, insecure, unregulated and possibly noncompliant atmosphere.

II. Secure and Fault-Tolerant Cloud-Based Information Storage

A. A Motivating Example

Consider a scenario where a medical company wishes to have all medical records of its patients available to its trusted partners, as well as to doctors who may be off site. First, the medical data is required to be highly fault tolerant, as losing patient records is not an option. Secondly, the data must be secure, as the company has an obligation to its patients to protect their personal information. Thirdly, the medical records must be guaranteed to be available, as it may become a matter of life or death if the data cannot be accessed quickly. The company realizes that storing the data on site would require a complex setup to make the data widely available to its central location and branch offices, with a large cost to purchase servers and storage devices. Furthermore, storing the data on site also requires a robust mechanism to ensure that the data is redundant and available in case of disaster, as well as a scalable infrastructure in case of growth. The company is attracted by the benefits of cloud computing, namely the availability of the data over the Internet for its remote offices and doctors, not having to invest a large amount of money to establish the infrastructure, the scalability, and the promise of resiliency and redundancy. Therefore, the company wishes to explore the option of using cloud computing for information storage and archiving of its data. It is, however, very concerned with moving its data into the cloud since losing physical control of its data be of high risk. Although the company can choose reputable cloud providers to host its data, there is no way to vet individual employees who are hired by the cloud provider to prevent insider attacks, whereas the medical company is required to do full background checks and audits on employees who are allowed to handle its data. The company is also concerned with the physical locations of its data. With a cloud provider, the medical company does not even know where its data resides in the cloud, let alone what safeguards are at place at the physical facility. The company has also seen through the media the amount of damage that can be caused by its data being compromised by a third party. It must assume that by storing its data in the cloud, it can be compromised, so it needs to ensure that the cloud providers and their employees absolutely do not have access to the underlying information. The company is finally concerned with the availability of its data, as although it sees cloud computing as mostly reliable, it needs to make sure that its data is available and that there are no extended length of outages. When the medical company is treating a patient, for example, it is critical to know if the patient is allergic to any medication. If this information became unavailable, there may be dire consequences. If the company chooses to build the data center by itself, it would take into account all of the above concerns. In the cloud, however, the environment is ever-changing with providers having the ability to decide critical implementation details, where data resides, and can at whim discontinue or radically change a service offering. Given the current state of cloud computing, the healthcare provider would have some very serious and legitimate concerns that need to be addressed.

In order to mitigate the major concerns that the medical company faces, we design a reliable, fault-tolerant, and secure architecture for cloud computing. Our approach can assist in bridging a major issue that resides in cloud computing yet to be solved, namely how to securely store personal information in the clouds. Thus, our approach can mitigate concerns from companies that are trying to adopt cloud computing and regulators as well as the general public who are concerned for the security and confidentiality of the stored information.

B. An Architectural Design

Our proposed information storage model for cloud computing can be deployed on multiple cloud service providers. As shown in Fig. 1, the model consists of users, a service directory, and a collective group of cloud storage providers. The users are cloud clients who wish to store and access data in the clouds. A user can first interact with the service directory, which acts as a coordinator that can set up a cloud cluster with multiple cloud service providers and assign it to the user, and also store information regarding the addresses of all service providers in the a cloud cluster.

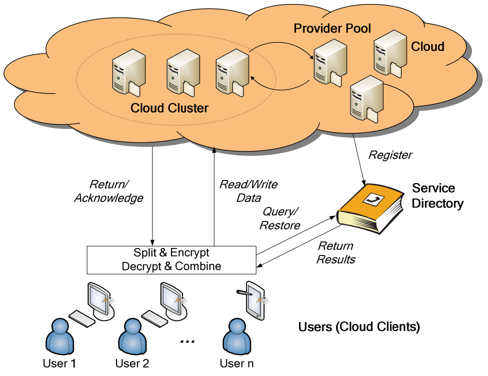


Figure 1. Architectural design of the information storage model

Once the client obtains the address information of the cloud providers from the service directory, it interacts with the cloud cluster, namely a collection of available service providers that can store and send data using predefined protocols. Each set of data, composed of the user's sensitive information can be split into multiple pieces using a predefined security mechanism, and are stored into the cloud cluster after they are encrypted. The cloud providers in the cloud cluster have no knowledge of which cluster they belong to as well as which are the other members in the cloud cluster they are servicing. This means the cloud clusters are *virtual* clusters in the clouds, which are generated and exclusively managed by the service directory. Furthermore, the service directory has the capability to restore data when needed. When a service provider in a cloud cluster fails, the service directory can automatically restore the data using a predefined restoration algorithm, and replace the failed service provider with a new one from the *Provider Pool*.

The security mechanism defined in our model consists of two levels. In the first level, the information to be stored is split into multiple pieces using the *RAID 5* techniques with distributed parity [6] so that if a provider fails, the data stored collectively in the cluster would be recoverable. Note that the *RAID* technique uses block-level striping with distributed parity in a cluster of disk drives [7]. Due to data redundancy, when a disk drive fails, any subsequent reads can be calculated from the distributed parity, and the data in the failed drive can be restored. In our approach, we consider each cloud provider in a cloud cluster as a virtual disk drive; thus, our information storage model is fault tolerant upon the failure of any cloud provider in the cloud cluster, and the missing piece of data can be recovered from the distributed parity stored with the other cloud providers in the cloud cluster. Another advantage of our approach is, due to the distribution of data over multiple cloud providers, no cloud provider is able to calculate the original data because the providers have no knowledge of which the other members are in the cloud cluster.

In the second level of our security mechanism, encryption plays an important role. To ensure that providers do not have access to the underlying data that is being stored, symmetric key encryption can be used. A symmetric key is an encryption key that is used for both encryption and decryption of data and should be kept secret from all entities that do not have access to the data. A user with the needed access permission can utilize a symmetric key to encrypt a piece of data to be stored prior to sending it out to the cloud and to decrypt the data after it is retrieved from a cloud provider. When a read operation is performed, all pieces of information need to be decrypted after retrieved from the cloud providers in the cloud cluster, and then they are combined into the original information.

In order to correctly design the security mechanism, we develop a formal model of the secure and fault-tolerant information storage system in cloud computing, and verify some key properties of the model. We adopt colored Petri net formalism because it is a well-founded process modeling technique that has formal semantics to allow specification, design, verification, and simulation of a complex concurrent software system [8]. A Petri net is a directed, connected, and bipartite graph, in which each node is either a place or a transition. In a Petri net model, tokens are used to specify information or conditions in the places, and a transition can fire when there is at least one token in every input place of the transition. Colored Petri nets (CPN or CP-net) are an extension of ordinary Petri nets, which allow different values (represented by different colors) for the tokens. Colored Petri nets have a formal syntax and semantics that leads to compact and operational models of very complex systems for modular design and analysis. The major advantage of developing a CPN model of the information storage system is to provide a precise specification, and to ensure a correct design of the information storage system; therefore, design errors, such as a deadlock, can be avoided in the implemented system.

III. Formal Modeling Secure and Fault-Tolerant Cloud-Based Information Storage Archietcture

To make the model easy to comprehend, we utilize hierarchical CPN (HCPN), which allows using substitution transitions and input/output ports to represent a secondary Petri net in the hierarchy. In our design, we first provide the high-level model with its key components. Then we utilize HCPN to refine each component into a more complete Petri net. Since the architecture we proposed is most suitable for storing personal or confidential data, In the following sections, we present the HCPN model with an example of medical record online storage system , which consists of a service directory, a cloud cluster with three cloud providers, and two users (cloud clients), namely a patient and a doctor.

A. High-Level Petri Net Model

The HCPN model can be devloped using CPN Tools [9]. In Fig. 2, we present a high-level model that defines the key components, namely the *Doctor*, the *Patient*, the *Cloud*, and the *Directory*, as well as the communications among the components. The key components are defined as substitution transitions, denoted as double rectangles in the figure. The purpose of the communications among the patient, doctor, and cloud is to transfer and access a patient's medical record. The directory acts as a data coordinator between the users and the cloud. To simulate the cloud providers that are selected as members of a cloud cluster as well as the data being transferred between the users and the cloud providers, a PROV and a MEDRECORD colored token type are defined using the ML functional language integrated in CPN Tools as follows:

```
colset PROV = record       colset MEDRECORD = record
  prID: STRING *             recID:STRING *
  ready: BOOL *              data: STRING;
  mrec: MEDRECORD;
```

where prID is a provider ID, ready is a flag of a provider indicating whether the provider is functioning or failed, mrec is a medical record, recID is a record ID, and data is the medical data stored in the record.

The directory is responsible for initializing the cloud providers in a cloud cluster assigned to a user, replying queries from a user for providers' addresses, and processing restoration request upon the failure of a cloud provider in a cloud cluster. As shown in Fig. 2, the cloud cluster (denoted as the place "*Cluster Providers*") is initialized with three providers "Pr1", "Pr2" and "Pr3" of type PROV, each of which is initialized with initrec that contains a blank medical record. Furthermore, the place "*Provider Pool*" is initialized with one spare cloud provider "Pr4", which can be used to replace a failed cloud provider in the cloud cluster.

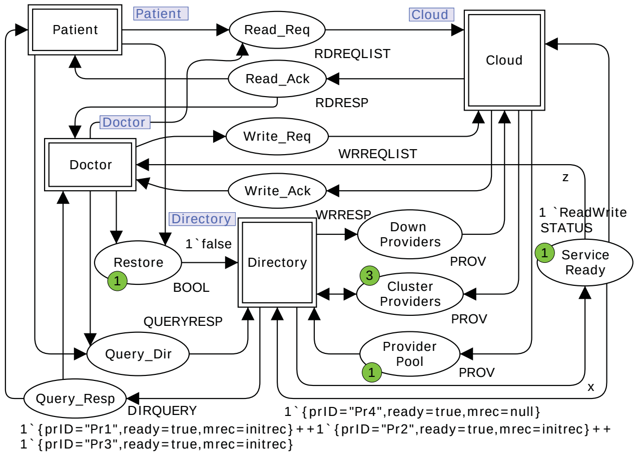


Figure 2. High-level CPN model of the cloud storage

A read request (RDREQ) and a write request (WRREQ) to a cloud provider can be defined as colored tokens as follows:

```
colset RDREQ = record      colset WRREQ = record
  clID:STRING *              clID:STRING *
  recID:STRING *             mrec:MEDRECORD *
  prID:STRING;               prID:STRING;
```

where clID is a client ID. Note that in Fig. 2, RDREQLIST and WRREQLIST are defined as a list of read requests, and a list of write requests, repectively. Thus, our model allows accessing multiple pieces of information concurrently from the cloud providers participating in a cloud cluster.

After a read (write) requst has been processed, a read (write) response will be returned to the user, simulated as a token of type RDRESP (WRRESP) being deposited in place "*Read_Ack*" ("*Write_Ack*"). The colored token types RDRESP and WRRESP are defined as follows:

```
 colset RDRESP = record    colset WRRESP = record
  clID:STRING *              clID:STRING *
  prID:STRING *              prID:STRING *
  mrec:MEDRECORD *           mrec:MEDRECORD *
  success:BOOL;              success:BOOL;
```

where the flag success indicates if a read request or a write request is successful or failed. In case a read or write request fails (i.e., a cloud provider is down), the user will change the token in place "*Restore*" from false to true, notifying the directory to start the restoration process for the cloud cluster.

B. Petri Net Model for the Directory Component

We now refine the *Directory* component (i.e., the *Directory* substitution transition in Fig. 2) into a CPN model as shown in Fig. 3. In the figure, the place "*Clust_Prov_List*" is intialized with a list of providers ["Pr1","Pr2","Pr3"] due to the initial setting of the cloud cluster in place "*Cluster Providers*." When a patient client or a doctor client starts querying the directory for the addresses of the providers in its assigned cloud cluster, a query token will be placed by the client into place "*Query_Dir*." This enables the transition "*Provider Locations*." When it fires, it creates a token of QUERYRESP type in palce "*Query_Resp*," which attaches the provider information stored in place "*Clus_Prov_List*." Note that to simplify our CPN model, the provider information only consists of the provider IDs rather than the providers' actual endpoint addresses. Therefore, a service invocation to a cloud provider could be simulated by matching the cloud provider's ID rather than calling at its endpoint address. Since the place "*Query_Resp*" is an input port of the clients, the token becomes available to the client for further processing. On the other hand, if the "*Restore*" place contains a true token due to an access error experienced by a user, the "*Check Providers*" transition becomes enabled as long as the directory is not currently restoring the cloud cluster (denoted by a false token in place "*Init_Restore*") and there is a failed provider (i.e., its ready flag is set to false) in place "*Cluster Providers*." Once the transition fires, it places a true token into the "*Init_Restore*" place, signifying that a restoration process should take place. The firing also removes the failed provider from the "*Clust_Prov_List*" place and transfers the provider from the "*Cluster Providers*" place to the "*Down Provider*" place. When the restoration process starts, the "*Restore*" transition fires, and deposits a copy of the remaining two providers into the "*Restore Gather Info*" place. This enables the "*Calculate Replacement*" transition, and its firing simulates the calculation of the missing piece of data based on the distributed parity information, and results in the restored medical record being placed in the "*Replacement Record*" place. Note that for simplicity, the detailed procedure of the parity calculation is not modeled in Fig. 3.

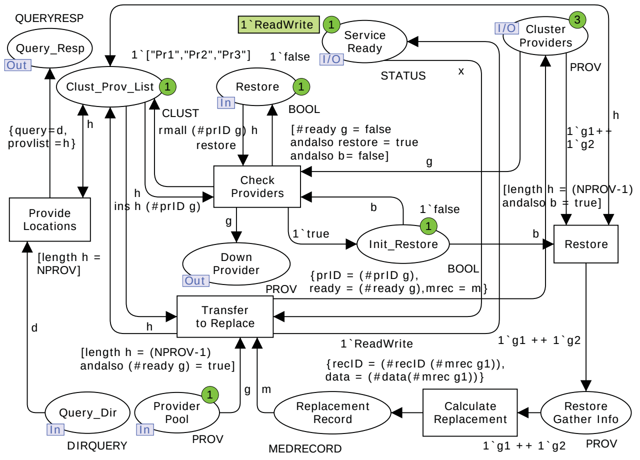


Figure 3. CPN model for the Directory component

Once the record has been restored, the "*Transfer to Replace*" transition becomes enabled, and its firing takes a provider from the "*Provider Pool*," initializes it with the restored medical record, updates provider list in the "*Clus_Prov_List*" place by adding the new provider into the list, and places the provider into the "*Cluster Providers*" place. This step completes the restoration process, with the required number of functioning providers allocated in the cloud cluster.

C. Petri Net Models for the Patient and Doctor Clients

A patient client should have the permisison to read its medical record. As shown in Fig. 4, a patient first requests the addresses of the cloud providers in the cloud cluster assigned to him, which is modeled by placing a true token in the

"*Query Directory*" place. With this token as well as the client ID of the user in the "*ClientID*" place, the "*Init_Query*" transition can fire, and its firing results in a `DIRQUERY` token to be placed in the "*Query_Dir*" output port.

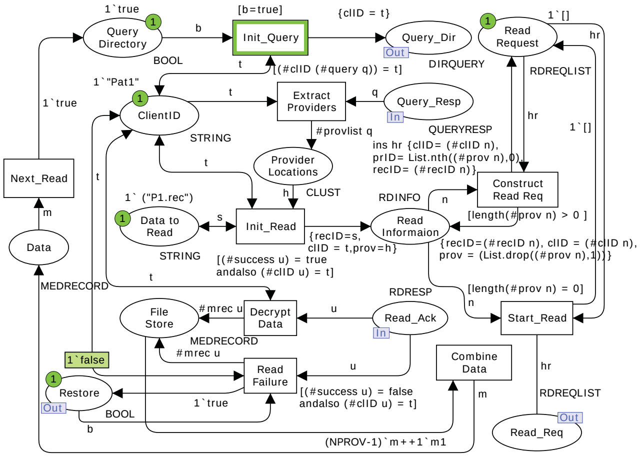


Figure 4. CPN model for the patient client

When a response from the directory is put into the "*Query_Resp*" input port, the providers' address information becomes available. This enables the "*Extract Providers*" transition, and the firing of the transition places a `CLUST` token in the "*Provider Locations*" place. The `CLUST` token type is defined as a list of providers as follows:

```
colset CLUST = list STRING with 0..3;
```

where the `with` clause specifies the minimum and maximum length of the list, and each item in the list contains the address of a provider (represented by its provider ID as a string for simplicity) that can be used by the client to communicate with the provider. To model a read operation, a token `"P1.rec"` is initialized in the "*Data to Read*" place, which is a record ID representing patient `P1`'s medical record. The firing of the "*Init_Read*" transition starts the read process, and places the record ID along with the provider information into the "*Read Information*" place. Note that in this model, we assume that there is only one record for each patient that can be matched with medical data stored on the providers. Now the "*Construct Read Req*" transition can fire once for each provider in the provider list, and creates a token of type `RDREQLIST` in the "*Read Request*" place, such that the multiple read requests in the list can be made concurrently to the cloud providers in the cloud cluster. This makes the associated providers in place "*Read Information*" being removed and enables the "*Start Read*" transition. When it fires, it transmits the `RDREQLIST` token to the "*Read_Req*" place, which is an input port to the cloud cluster. After the requests have been processed by the cloud providers, multiple tokens of type `RDRESP` will be deposited in place "*Read_Ack*." If a `RDRESP` token contains a `success` flag with a `true` value, it indicates that the read request has been completed successfully by the corresponding cloud provider. In this case, the piece of medical record is extracted from the token and placed in the file store after bieng decrypted. Once all pieces of the medical record are successfully decrypted, the "*Combine Data*" transition becomes enabled and can fire. The firing simulates the process of generating the original medical record by recombining the *RAID* data slices retrieved from the cloud providers. If one of the providers returns a token with the `success` flag set to `false`, a read failure occurs for the cloud provider. In this case, the "*Read Fail*" transition becomes enabled. Once it fires, it changes the token in place "*Restore*" from `false` to `true`, signifying the directory to initiate a "restore" operation.

The CPN model for the doctor client that replaces the `Doctor` substitution transition of the high-level model is similar to the one for the patient client, but a doctor client should also have the privilege to write data into the clouds. Due to page limits, we do not show such a CPN model here.

D. Petri Net Model for the Cloud Component

Finally, we refine the *Cloud* substitution transition of the high-level model into a CPN model as shown in Fig. 5, where the cloud providers are represented as colored tokens of type `PROV`. The clouds can accept either "read" or "write" requests from the clients, namely the patient and the doctor. Upon receiving the requests, cloud providers invoke corresponding cloud services by matching their IDs in the cloud cluster, and return responses to the clients. In addition, the cloud providers in a cloud cluster are also responsible for providing their data to the service directory on demand in a case that a restoration process is initiated when a read or write request fails due to the failure of a cloud provider in the cloud cluster.

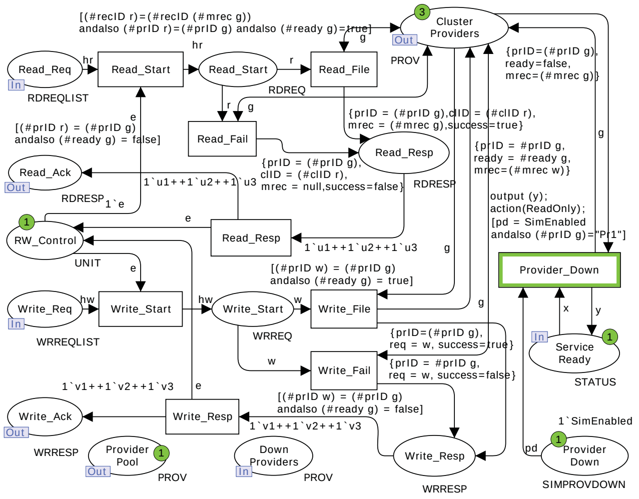


Figure 5. CPN model for the cloud component

In this model, the "*Cluster Providers*" place is shared with the directory, where the `PROV` tokens in the place represent the providers selected to constitute the cloud cluster. In addition, the "*Provider Pool*," "*Down Providers*," and "*Service Ready*" places are also shared places in the directory model. The "*Provider Pool*" acts as a holding place for available providers identified by the directory. The "*Down Providers*" place contains the providers that are down and deemed needing replacement. Finally, the "*Service Ready*" place acts as an input place to the cloud for simulation purposes only. In our current model, we only consider a maximum of one cloud provider going down at a time. This is a reasonable assumption because cloud providers should be somewhat reliable. In order to satisfy this constraint in the model, the "*Provider Down*" place is connected to the "*Provider_Down*" transition, which allows the "*Provider_Down*" transition to fire once.

When a client makes a "read" request, a `RDREQLIST` token with a list of `RDREQ` requests will be deposited into place "*Read_Req*." This enables the "*Read_Start*" transition as long as the "*RW_Control*" place contains a unit token, which ensures "read" and "write" actions are mutual exclusive. When the "*Read_Start*" transition fires, it splits the `RDREQLIST` token into singular `RDREQ` tokens, places them into place "*Read_Start*," and removes the unit token from the "*RW_Control*" place. The "*Read_File*" transition then examines each of the `RDREQ` tokens, matches it with its respective cloud provider, and fires as long as the `success` flag of the corresponding `PROV` token in place "*Cluster Provider*" is `true`. The following ML transition guard code accomplishes this task:

```
[(#recID r) = (#recID (#mrec g)) andalso
 (#prID r) = (#prID g) andalso (#ready g)=true]
```

where `g` represents a cloud provider that is a member of the cloud cluster and `r` represents a "read" request. The guard selects the correct provider by comparing the provider ID in the request `(#prID r)` with that of a provider from the cluster`(#prID g)`, matches the medical record ID, and makes sure that the cloud provider is functioning, i.e., its `ready` flag is set to `true`. If all conditions are met, the transition can fire, and the firing of the transition creates a `RDRESP` token and deposits it into the "*Read_Resp*" place. On the other hand, if a "read" request fails due to the corresponding provider being not ready (i.e., its `ready` flag is set to `false`), the "*Read_Fail*" transition can fire, and its firing sends a `RDRESP` token with a blank medical record and a `success` flag set to `false` to the "*Read_Resp*" place. Once all three tokens are in the "*Read_Resp*" place, the "*Read_Resp*" transition may fire. The firing of the transition returns a unit token to the "*RW_Control*" place and places the `RDRESP` tokens into the "*Read_Ack*" port, available for the clients to digest.

A "write" request follows an almost identical path through the model. When the doctor places a `WRREQLIST` token into the "*Write_Req*" port, the "*Write_Start*" transition becomes enabled, and the firing of the transition places the individual `WRREQ` tokens into place "*Write_Start*." With the tokens in this place, the "*Write_File*" transition can fire as long as the `ready` flag of some `PROV` token in place "*Cluster Providers*" is `true`. The firing of the transition replaces the medical record stored in the `PROV` token with the replacement record, and also constructs a `WRRESP` token and places it in the "*Write_Resp*" place. On the other hand, if the `ready` flag of a provider is set to `false`, the "*Write_Fail*" transition may fire. In this case, the medical record is not altered, and a `WRRESP` token with the `success` flag set to `false` will be deposited in place "*Write_Resp*." Once all three `WRRESP` tokens are in the "*Write_Resp*" place, the "*Write_Resp*" transition can fire, and its firing returns a unit token back to the "*RW_Control*" place and deposits the `WRRESP` tokens in the "*Write_Ack*" place, being available for the client to process.

A restoration process can be simulated in the cloud model by setting the `SIMPROVDOWN` token in place "*Provider Down*" to `SimEnabled`. When the "*Provider_Down*" transition fires, it randomly selects a provider from the place "*Cluster Providers*" and sets the `ready` flag of the provider to `false.` This step simulates the failure of a cloud provider in the cloud cluster. Furthermore, the firing of the transition also sets the `STATUS` token in place "*Service Ready*" to `ReadOnly`, which disables the transition for writing in the CPN model for the doctor patient. The doctor patient will be allowed to write again only after the `STATUS` token in place "*Service Ready*" is changed back to `ReadWrite`. Meanwhile, when either a patient or a doctor client experiences an access error to a failed cloud provider, a restoration process will be initiated by the client. Communication with the directory for a "restore" operation is done through the shared port "*Cluster Providers*." This port, containing the `PROV` tokens of the providers who make up the cluster, allows the directory direct access to the `PROV` state when required. When the restoration process completes, the failed cloud provider in place "*Cluster Providers*" will be replaced by a new one taken from the "*Cluster Pool*."

IV. Formal Analysis of the CPN-Based Model

In addition to providing an accurate model for our proposed security mechanisms for cloud-based information storage, building a formal design model also has the advantage of ensuring a correct design through state space analysis. Utilizing the CPN Tools, a formal analysis of the CPN model can be performed to verify if the model meets certain system requirements. Typically, the model we developed should be live, bounded, and deadlock-free. When we use the CPN Tools to calculate the state space and analyze its major behavioral properties, the CPN Tools produce the following results:

```
Statistics                Liveness Properties
-----------------------   ---------------------------
  State Space               Dead Markings
     Nodes:  154908            99 [44684,44683,44682,
     Arcs:   571408            44510,44509,...]
     Secs:   1832             Dead Transition Instances
     Status: Full                None
  Scc Graph                 Live Transition Instances
     Nodes:  90616               None
     Arcs:   431478
     Secs:   37
```

The analysis shows that the state space contains dead markings, thus the model we developed must contain deadlocks. By tracing the firing sequence for the deadlock states as we did in our previous work [10], we found a subtle design error. The error is due to the removal of the failed cloud provider from the place "*Cluster Provider*" in the CPN model for the *Directory* component, which occurs when the transition "*Check Providers*" fires. However, some "read" request in place "*Read_Req*" of the CPN model for the *Cloud* component would require communicating with a removed cloud provider if the "read" request was created before the cloud provider fails. Since there is no matched cloud provider in the "*Cluster Provider*" place of the *Directory* model, the system may enter a deadlock state. The easiest way to fix this problem is to allow the failed cloud provider to stay in the "*Cluster Provider*" place. This would allow the "*Read_Fail*" transition to fire, and return a "read" error to the client. After we add a new arc from the transition "*Check Providers*" to the place "*Cluster Provider*" in the *Directory* model, the CPN Tools now produce the following results:

```
Statistics                 Liveness Properties
----------------------     --------------------------
  State Space                Dead Markings
     Nodes:  204267             None
     Arcs:   880021          Dead Transition Instances
     Secs:   4599               None
     Status: Full            Live Transition Instances
  Scc Graph                     Cloud'Read_File 1
     Nodes:  133063             Cloud'Read_Resp 1
     Arcs:   726216             Cloud'Read_Start 1
     Secs:   242                ...


Boundedness Properties
-------------------------------------------------
  Place                            Upper      Lower
  Cloud'Provider_Down                1          0
  Cloud'RW_Control                   1          0
  Cloud'Read_Resp                    3          0
  Cloud'Read_Start                   3          0
  Cloud'Write_Resp                   3          0
  Cloud'Write_Start                  3          0
  Directory'Clust_Prov_List          1          1
  Directory'Init_Restore             1          0
  Directory'Replacement_Record       1          0
  Directory'Restore_Gather_Info      2          0
  High_Level'Cluster_Providers       4          3
  High_Level'Down_Providers          1          0
  High_Level'Provider_Pool           1          0
  High_Level'Query_Dir               2          0
  High_Level'Query_Resp              2          0
  High_Level'Read_Ack                6          0
  High_Level'Read_Req                2          0
  High_Level'Restore                 1          1
  High_Level'Service_Ready           1          1
  High_Level'Write_Ack               3          0
  High_Level'Write_Req               1          0
  ...
```

The analysis shows that our modified net model is deadlock free, and all transitions except those related to the restoration process are live. Note that in our simulation, we allow the "*Provider_Down*" transition in the *Cloud* model can fire only once. The analysis also shows that our net model is bounded. We notice that the upper bound of the place "*Cluster_Providers*" in the high-level model is 4 rather than 3. This is because a failed cloud provider will be kept in the cloud cluster after the service directory restores the cloud cluster by adding a replacement cloud provider into the cluster. A more sophisticated model that allows a failed cloud provider to be removed from the cloud cluster, and also allows more than one patient and more than one doctor to access cloud clusters with shared cloud providers is envisioned as a future, and more ambitious research plan.

V. Conclusions and Future Work

Cloud computing is quickly becoming a widely adopted platform to allow for complex computational nodes and storage clusters without any of the difficulties and cost associated with configuration and maintenance. There are, however, major legitimate concerns from enterprises and sensitive data holders related to offsite storage of personal or mission critical data. Studies show that given the current state of cloud computing, enterprises are very concerned with unresolved issues related to security, trust, and management in the cloud. For a majority of these enterprises, this is also the main reason why they have not yet adopted cloud computing into their infrastructure. In this paper, we introduce a cloud-based information storage model that takes into account the fact that cloud providers may experience outages, data breaches, and exploitations. We cope with these issues by developing a distributed cloud-based security mechanism. We then utilize hierarchical colored Petri nets to formally model and analyze our concurrent security model. The verification results show that the model we developed is live, bounded and deadlock-free.

For future work, we plan to develop a more sophisticated model that allows more clients to access the cloud clusters with shared cloud providers, and demonstrate how to cope with the state explosion problem using the net reduction approach. Meanwhile, we will try to implement a prototype cloud-based information storage system with an improved distributed parity algorithm that may strengthen the security mechanism by preventing potential provider collusion to obtain information stored in a cloud cluster. Finally, the model can be further improved if we allow the service directory to autonomously detect failures, drops in quality of service (QoS), and anomalies of the cloud providers, and react accordingly.

References

[1] GAO, "Information Security: Additional Guidance Needed to Address Cloud Computing Concerns," *United States Government Accountability Office (GAO)*, October 6, 2011, Retrieved on December 18, 2011 from http://www.gao.gov/new.items/d12130t.pdf

[2] AWS, "Creating HIPAA-Compliant Medical Data Applications with AWS," *Amazon Web Services (AWS)*, Amazon, April 2009, Retrieved on August 22, 20 10, from http://awsmedia.s3.amazonaws.com/AWS_HIPAA _Whitepaper_Final.pdf.

[3] M. T. Goodrich, R. Tamassia, and D. Yao, "Notarized Federated ID Management and Authentication," *Journal of Computer Security*, Vol. 16, No. 4, December 2008, pp. 399-418.

[4] A. C. Weaver, "Enforcing Distributed Data Security via Web Services," In *Proceedings of the IEEE International Workshop on Factory Communication Systems (WFCS)*, Vienna, Austria, September 22-24, 2004, pp. 397-402.

[5] N. Santos, K. Gummadi, and R. Rodrigues, "Towards Trusted Cloud Computing," In *Proceedings of the Workshop on Hot Topics in Cloud Computing (HotCloud09)*, San Diego, CA, USA, June 15, 2009.

[6] A. Thomasian and J. Menon, "RAID 5 Performance with Distributed Sparing," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 8, No. 6, June 1997, pp. 640-657.

[7] D. A. Patterson, P. Chen, G. Gibson, and R. H. Katz, "Introduction to Redundant Arrays of Inexpensive Disks (RAID)," *COMPCPN Spring'89, Thirty-Fourth IEEE Computer Society International Conference: Intellectual Leverage, Digest of Papers*, Feb. 27 - March 3, 1989, San Francisco, CA , USA, pp. 112-117.

[8] K. Jensen, *Colored Petri Nets: Basic Concepts, Analysis Methods and Practical Use*, Vol. I: Basic Concepts, EATCS Monographs on Theoretical Computer Science, New York Springer-Verlag, 1992.

[9] A. V. Ratzer, L. Wells, H. M. Lasen, M. Laursen, J. F. Qvortrup, et al., "CPN Tools for editing, simulating and analyzing colored Petri nets," In *Proceedings of the 24th International Conference on Application and Theory of Petri Nets*, Eindhoven, Netherlands, Jun. 2003, pp. 450-462.

[10] H. Xu, M. Ayachit, and A. Reddyreddy, "Formal Modeling and Analysis of XML Firewall for Service-Oriented Systems," *International Journal of Security and Networks (IJSN)*, Vol. 3, No. 3, 2008, pp. 147-160.

Decidability of Minimal Supports of S-invariants and the Computation of their Supported S-invariants of Petri Nets

Faming Lu, Qingtian Zeng*, Hao Zhang,Yunxia Bao, Jiufang An
Shandong University of Science and Technology
Qingdao, China

***Abstract*—S-invariants play an important role in the structural property analysis of Petri nets and there is no algorithm that can derive all the S-invariants of a Petri net in polynomial time. Fortunately what needed to do in some practical applications is only to decide whether or not a given place subset is a minimal support of S-invariants or to compute one of its supported S-invariants. For this reason, a sufficient and necessary condition for a place subset to be a minimal support of S-invariants is proved in this paper. After that, two polynomial algorithms for the decidability of a minimal support of S-invariants and for the computation of an S-invariant supported by a given minimal support are presented.**

Keywords: Petri nets; S -invariants; minimal supports of S-invariants

I. INTRODUCTION

Petri nets are widely applied in modeling and simulation of flexible manufacturing system, workflow management, discrete event systems and many other fields [1-4]. S-invariants are one of the basic analysis tools of Petri nets from which we can analyze such properties as conservativeness, liveness and other important properties of a system.

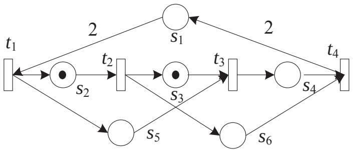
For the computation of S-invariants, reference [1] has already pointed out that there is no algorithm which can derive all the S-invariants of Petri nets in polynomial time complexity. Even so, there is still a lot of work devoted to deriving S-invariants. In [5], a linear programming based method is presented to compute part of S-invariant's supports, but integer S-invariants can't be obtained. In [6-7], a Fourier_Motzkin method is presented to compute a basis of all S-invariants, but its time complexity is exponential. Reference [8-9] put forward a STFM method which has a great improvement in efficiency compared to the above methods, but there are some kinds of Petri nets the S-invariants of which can't be obtained with STFM method and the STFM method has an exponential time complexity too in the worst case.

So the difficulty of deriving S-invariants is high. What needed to do in some practical applications is only to decide whether or not a given place subset is a minimal support of S-invariants, or to compute only one S-invariant for a given place subset. Even so, there is no efficient algorithm by now which can decide the minimal supports of S-invariants or compute an S-invariant for a given place subset. In this paper, two polynomial algorithms for the decidability of a minimal support of S-invariants and for the computation of an S-invariant supported by a given minimal support are presented.

The rest of this paper is organized as follows. In Section 2, we provide the basic concepts about Petri nets and S-invariants. In Section 3, some properties of S-invariants are presented. With these properties, a polynomial algorithm is provided to decide the minimal supports of S-invariants in Section 4. After that a polynomial algorithm to compute an S-invariant for a given place subset is presented in Section 5. Finally a case study and a conclusion are given in Section 6-7.

II. BASIC CONCEPTS ABOUT PETRI NETS

A Petri net is a 5-tuple $\Sigma=(S,T;F,W,M_0)$, where S is a finite set of places, T is a finite set of transitions, $F\subseteq(S\times T)\cup(T\times S)$ is a set of flow relation, $W:F\rightarrow\{1,2,3,...\}$ is a weight function, $M_0:S\rightarrow\{0,1,2,\cdots\}$ is the initial marking, and $S\cap T=\varnothing\wedge S\cup T\neq\varnothing$. Usually, a Petri net can be represented by a bipartite graph just like in Fig.1 where a place is presented by a circle, a transition is presented by a rectangle, a flow relation is presented by an arc and the marking is presented by those black-points in places.



$$A=\begin{bmatrix}-2 & 1 & 0 & 0 & 1 & 0\\ 0 & -1 & 1 & 0 & 0 & 1\\ 0 & 0 & -1 & 1 & -1 & 0\\ 2 & 0 & 0 & -1 & 0 & -1\end{bmatrix}$$

Fig 1. A Petri net example Σ_1

According to [3-4], the incidence matrix of Σ_1 is the above matrix A with each row corresponding to a transition and each column corresponding to a place. Place subsets $S_1=\{s_1,s_2,s_3,s_4\}$ and $S_2=\{s_1,s_4,s_5\}$ are two minimal supports of S-invariants of Σ_1. It has been proved in [4] that there is a unique minimal S-invariant supported by a given minimal support. And the minimal S-invariants supported by S_1 and S_2 are $Y_1=[1\ \ 2\ \ 2\ \ 2\ \ 0\ \ 0]^T$ and $Y_2=[1\ \ 0\ \ 0\ \ 2\ \ 2\ \ 0]^T$ respectively. $S_3=\{s_1,s_2,s_3,s_4,s_5\}$ is a support of S-invariants too and $Y_3=[1\ \ 1\ \ 1\ \ 2\ \ 1\ \ 0]^T$ is one of its supported S-invariants. But S_3 is not a minimal

This work is supported partly by the NSFC (61170079); Sci. & Tech. Development Fund of Shandong Province of China (2010GSF10811); Specialized Research Fund for the Doctoral Program of Higher Education of China (20103718110007); Sci. & Tech. Development Fund of Qingdao(10-3-3-32-nsh and 2011-2-47), Excellent Young Scientist Foundation of Shandong Province (BS2009DX004 and BS2010DX009);Natural Science Foundation for Distinguished Young Scholars of Shandong and SDUST (JQ200816 and 2010KYJQ101),Guiding project of Coal Ministry(MTKJ2011-370); Project of Shandong Province Higher Educational Sci.&Tech. Program(J12LN11); Research Project of SUST Spring Bud(2010AZZ177, 2010AZZ069).

** corresponding author: Q. Zeng, Email: qtzeng@sdust.edu.cn*

support because $S_3 = S_1 \cup S_2$ which means that S_3 is the union of other supports.

III. Properties Of Minimal Supports Of S-invariants

In this section, we will give a sufficient and necessary condition for a place subset to be a minimal support of S-invariants. Before that, some properties are given below.

***Lemma* 1**[12] If S_0 is an support of S-invariants, but not a minimal support, then there are at least two minimal supports of S-invariants $S_1, S_2, ..., S_k\,(k \ge 2)$ such that $S_0 = S_1 \cup S_2 \cup ... \cup S_k$.

***Lemma* 2** Let C be an $m*n$ ordered integer matrix and the rank of C satisfies $R(C) < n$. Then the homogeneous equation $CX = \vec{0}$ has a fundamental system of solution in which all the base solutions are integer vectors.

Proof: Because $R(C) < n$ and the coefficient matrix C is an integer matrix, using Gaussian elimination method, we can get a fundamental system of solution which is consisted of rational vectors for the homogeneous equation $CX = \vec{0}$. We denote the system by $\xi = \{\beta_1, \beta_2, ..., \beta_k\}$. Because $\beta_i\,(1 \le i \le k)$ is a rational vector, so we can suppose that $\beta_i = \begin{bmatrix} \frac{\mu_{i1}}{v_{i1}} & \frac{\mu_{i2}}{v_{i2}} & \cdots & \frac{\mu_{in}}{v_{in}} \end{bmatrix}^T$, where $\mu_{i1}, ..., \mu_{in}$ and $v_{i1}, ..., v_{in}$ are all integers and $v_{i1}, ..., v_{in}$ are nonzero.

Let $\lambda_i = \prod_{j=1}^{n} v_{ij}$ and $\beta_i\,' = \lambda_i * \beta_i = \begin{bmatrix} \mu_{i1}\prod_{\substack{j=1\\ j\neq 1}}^{n} v_{ij} & \mu_{i2}\prod_{\substack{j=1\\ j\neq 2}}^{n} v_{ij} & \cdots & \mu_{in}\prod_{\substack{j=1\\ j\neq n}}^{n} v_{ij} \end{bmatrix}^T$.

Then it is easy to see that $\beta_i^{'}$ is an integer vector. And as a constant times of the base solution vector β_i for the homogeneous equation $CX = \vec{0}$, $\beta_i^{'}$ is surely to be a base solution of $CX = \vec{0}$. As a result, $\xi\,' = \{\beta_1^{'}, \beta_2^{'}, ..., \beta_k^{'}\}$ constitute a fundamental system of integer solution of $CX = \vec{0}$.■

***Definition* 1**[4] Let A be an incidence matrix of a Petri net $\Sigma = (S, T; F, W, M_0)$, and $A_i\,(1 \le i \le |S|)$ denotes the i^{th} column vector in A. For a given place subset $S_1 = \{s_{j_1}, s_{j_2}, ..., s_{j_k}\} \subseteq S$, $A_{S_1} = [A_{j_1}, A_{j_2}, ..., A_{j_k}]$, a sub-matrix of A, is called the generated sub-matrix corresponding to S_1.

***Theorem* 1** Let $S_1 = \{s_{j_1}, s_{j_2}, ..., s_{j_k}\}$ be a place subset, A_{S_1} be the generated sub-matrix corresponding to S_1. If the rank of A_{S_1} satisfies $R(A_{S_1}) = |S_1| - 1$ and $A_{S_1} Y_{S_1} = \vec{0}$ has positive integer solutions, then S_1 is a minimal support of S-invariants.

Proof: Since $A_{S_1} Y_{S_1} = \vec{0}$ has positive integer solutions, S_1 must be a support of S-invariants. Assuming that S_1 is not a minimal support of S-invariants, then there are at least two minimal supports of S-invariants $S_{11}, S_{12}, ..., S_{1k}\,(k \ge 2)$ such that $S_1 = S_{11} \cup S_{12} \cup ... \cup S_{1k}$ according to Lemma 1. Let Y_i and Y_j ($1 \le i, j \le k \wedge i \ne j$) be the minimal S-invariants supported by S_{1i} and S_{1j} respectively. It's obviously that Y_i and Y_j are linearly independent and $AY_i = AY_j = \vec{0}$.

Denote the l^{th} element of Y_i by $Y_i(l)$. If $Y_i(l)$'s corresponding place s_l does not belong to S_1, then $Y_i(l) = 0$ because the support of S-invariant Y_i, i.e. S_{1i}, is a subset of S_1. Deleting all those elements whose corresponding places don't belong to S_1 from Y_i, and denoting the resulted vector by β_i, then $A_{S_1}\beta_i = \vec{0}$ holds because β_i is constructed by deleting some zero elements from Y_i and $AY_i = \vec{0}$ holds.Similarly, deleting all those elements whose corresponding places don't belong to S_1 from Y_j, and denoting the resulted vector by β_j, then the equation $A_{S_1}\beta_j = \vec{0}$ holds.

In addition, because Y_i and Y_j are linearly independent, β_i and β_j are obtained by deleting the same zero elements from Y_i and Y_j respectively, so β_i and β_j must be linearly independent too. Furthermore, from $A_{S_1}\beta_i = \vec{0}$ and $A_{S_1}\beta_j = \vec{0}$ we can see β_i and β_j are two linear independent solutions of $A_{S_1} Y_{S_1} = \vec{0}$. Only when $R(A_{S_1}) <= |S_1| - 2$, $A_{S_1} Y_{S_1} = \vec{0}$ can have such two independent solutions. But this is contradictory with the assumption $R(A_{S_1}) = |S_1| - 1$ in the theorem. So S_1 must be a minimal support of S-invariants. ■

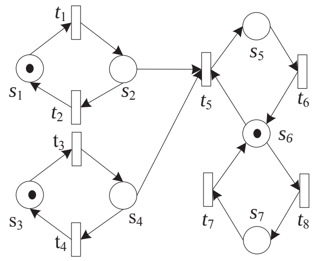


Fig 2. A Petri net example Σ_2

Now let's verify Theorem 1 with Peri net Σ_2 in Fig.2. In fact, Σ_2 has two minimal supports of S-invariants, which are $S_1 = \{s_1, s_2, s_5, s_6, s_7\}$ and $S_2 = \{s_3, s_4, s_5, s_6, s_7\}$ respectively.

The generated sub-matrix A_{S_1} corresponding to $S_1 = \{s_1, s_2, s_5, s_6, s_7\}$ is as follows.

$$A_{S_1} = \begin{bmatrix} -1 & 1 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 1 & 0 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & -1 & 1 \end{bmatrix} \xrightarrow[r8+r7]{r2+r1} \begin{bmatrix} -1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 1 & 0 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

After two elementary-row-transformations listed above, it's easy to see that the rank of A_{S_1} satisfies $R(A_{S_1}) = 4 = \left|S_1 - 1\right|$ and $Y_{S_1} = \begin{bmatrix}2 & 2 & 1 & 1 & 1\end{bmatrix}^T$ is a positive integer solution of $A_{S_1}Y_{S_1} = \vec{0}$. According to Theorem 1, S_1 is a minimal support of S-invariants. This is consistent with the facts.

$$A_{S_3} = \begin{bmatrix} -1 & 1 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 1 & -1 & 0 \\ 0 & -1 & 0 & -1 & 1 \\ 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \xrightarrow{\substack{r2+r1 \\ r2\leftrightarrow r5 \\ r4+r3 \\ r4\leftrightarrow r6}} \begin{bmatrix} -1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & -1 & 1 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

However, for $S_3 = \{s_1, s_2, s_3, s_4, s_5\}$, its corresponding generated sub-matrix A_{S_3} is given above. After four elementary-raw-transformations, it's easy to see that the last element in any solution of $A_{S_3}Y_{S_3} = \vec{0}$ must be 0 according to the 4th line in the ladder-type matrix. So $A_{S_3}Y_{S_3} = \vec{0}$ has no positive integer solutions. According to Theorem 1, S_3 isn't a minimal support of S-invariants. This is correct again.

***Theorem* 2** If S_1 is a minimal support of S-invariants, then the rank of A_{S_1} satisfies $R(A_{S_1}) = |S_1| - 1$ and $A_{S_1}Y_{S_1} = \vec{0}$ has positive integer solutions.

Proof: Let Y_0 be the minimal S-invariant supported by S_1, β be the sub-vector of Y_0 corresponding to S_1. Obviously, β is a positive solution of $A_{S_1}Y_{S_1} = \vec{0}$ according to the definition of minimal S-invariants. Next, we only need to prove $R(A_{S_1}) = |S_1| - 1$.

Assuming that $R(A_{S_1}) = |S_1|$, then A_{S_1} is a full-column-rank matrix. According to [10], the equation $A_{S_1}Y_{S_1} = \vec{0}$ has only zero solutions when A_{S_1} is full-column-rank. It is contradictory with the above conclusion that β is a positive solution of $A_{S_1}Y_{S_1} = \vec{0}$.

Assuming that $R\left(A_{S_1}\right) < |S_1| - 1$, according to Lemma 2, the equation $A_{S_1}Y_{S_1} = \vec{0}$ has a fundamental system of integer solution, denoted by $\xi = \{\beta_1, \beta_2, ..., \beta_k\}$,where $k \geq 2$.

1) On one hand, we will prove that any base solution in ξ isn't constant times of β. Otherwise, assuming that $\beta_i \in \xi$ satisfies $\beta_i = c * \beta$ ($c \in R \wedge c \neq 0$). Since the linear independence among base solutions, $\beta_j \in \xi (1 \leq j \leq k \wedge j \neq i)$ isn't constant times of β. So for any positive integers λ_1 and λ_2, $\beta' = \lambda_1 * \beta_i + \lambda_2 * \beta_j$ isn't constant times of β.

However, because β_i and β_j are both positive integer solutions of $A_{S_1}Y_{S_1} = \vec{0}$, so $\beta' = \lambda_1 * \beta_i + \lambda_2 * \beta_j$ is also a positive integer solution vector of $A_{S_1}Y_{S_1} = \vec{0}$. So β' corresponds to an S-invariant supported by S_1. By now, an S-invariant corresponding to β' is obtained, which isn't constant times of the minimal S-invariant corresponding to β. Both of them are supported by the same minimal support S_1. However, it has been proved in [4] that any S-invariant is a constant multiple of the minimal S-invariant when they are supported by one minimal support of S-invariants. This contradiction indicates that **any fundamental system of integer solution of $A_{S_1}Y_{S_1} = \vec{0}$ should contain no base solutions which are constant times of β**;

2) On the other hand, as a solution of $A_{S_1}Y_{S_1} = \vec{0}$, β should have the form that

$$\beta = \lambda_1 * \beta_1 + \lambda_2 * \beta_2 + ... + \lambda_k * \beta_k \text{ ------(1)}$$

where $\lambda_1, \lambda_2, ..., \lambda_k$ are constants.

As has been proved that each vector in $\xi = \{\beta_1, \beta_2, ..., \beta_k\}$ can't be constant times of β, so there are at least two nonzero constants in $\{\lambda_1, \lambda_2, ..., \lambda_k\}$. Assuming that $\lambda_i \neq 0$, then the following equation holds:

$$\beta_i = (\beta - \sum_{\substack{j=1 \\ j \neq i}}^{k} \lambda_j \beta_j) / \lambda_i \text{ ------(2)}$$

According to (1) and (2), $\xi' = \{\beta_1, \beta_2, ...\beta_{i-1}, \beta, \beta_{i+1}, ..., \beta_t\}$ and $\xi = \{\beta_1, \beta_2, ..., \beta_t\}$ are equivalent. So ξ' is also a fundamental system of integer solution of $A_{S_1}Y_{S_1} = \vec{0}$. However, ξ' contains vector β, which is contradictory to the conclusion obtained in the last sentence (shown in bold) of part 1) in this proof. So $R\left(A_{S_1}\right) < |S_1| - 1$ is not true.

To sum up, neither $R\left(A_{S_1}\right) = |S_1|$ nor $R\left(A_{S_1}\right) < |S_1| - 1$ holds which indicates $R(A_{S_1}) = |S_1| - 1$.■

In Fig.2, $S_1 = \{s_1, s_2, s_5, s_6, s_7\}$ is a minimal support of S-invariants, so $R(A_{S_1}) = |S_1| - 1$ holds and $A_{S_1}Y_{S_1} = \vec{0}$ has a positive integer solution $Y_{S_1} = [2 \quad 2 \quad 1 \quad 1 \quad 1]^T$, which is consistent with Theorem 2.

Now we can get a sufficient and necessary condition for a place subset to be a minimal support of S-invariants as follows.

***Theorem* 3** For a place subset S_1, it is a minimal support of S-invariants if and only if $R(A_{S_1}) = |S_1| - 1$ and $A_{S_1} Y_{S_1} = \vec{0}$ has positive integer solutions.

Proof: It can be derived from Theorem 1 and Theorem 2. ■

IV. Decidability Of A Minimal Support OF S-invariants

For the conditions of Theorem 3, whether $R(A_{S_1}) = |S_1| - 1$ is established or not can be determined through elementary transformation method of matrix. But it's difficult to decide whether $A_{S_1} Y_{S_1} = \vec{0}$ has positive integer solutions. Fortunately, the following lemma can address the problem.

***Lemma* 3** Let $R\left(A_{S_1}\right) = |S_1| - 1$ and η be an arbitrary non-trivial solutions of $A_{S_1} Y_{S_1} = \vec{0}$. The equation $A_{S_1} Y_{S_1} = \vec{0}$ has positive integer solutions if and only if η is a positive vector or a negative vector.

Proof: First, we prove the sufficiency. Assuming η is positive or negative, we only need to construct a positive integer solution of $A_{S_1} Y_{S_1} = \vec{0}$.

Since $R\left(A_{S_1}\right) < |S_1|$, $A_{S_1} Y_{S_1} = \vec{0}$ has a fundamental system of integer solution according to Lemma 2. Let β be an integer solution in such a system.

From $R\left(A_{S_1}\right) = |S_1| - 1$, we can know that β must be constant times of η. Because η is positive or negative, so is β. If β is negative, $-1 * \beta$ is a positive integer solution of $A_{S_1} Y_{S_1} = \vec{0}$. If β is positive, β itself is a positive integer solution of $A_{S_1} Y_{S_1} = \vec{0}$. In both cases, we construct a positive integer solution of $A_{S_1} Y_{S_1} = \vec{0}$ successfully.

Next, we prove the necessity. If $A_{S_1} Y_{S_1} = \vec{0}$ has positive integer solutions, let β be such a positive integer solution. Since $R\left(A_{S_1}\right) = |S_1| - 1$ there must be a nonzero constant λ such that $\eta = \lambda * \beta$. Because β is a positive integer vector, η is positive or negative. ■

***Theorem* 4** Let η be an arbitrary non-trivial solution of $A_{S_1} Y_{S_1} = \vec{0}$. S_1 is a minimal support of S-invariants if and only if $R(A_{S_1}) = |S_1| - 1$ and η is positive or negative.

Proof: It can be derived from Theorem 3 and Lemma 3. ■

According to Theorem 4, we can get the following algorithm to decide a minimal support of S-invariants.

```
Algorithm 1 Decidability of a minimal support of S-invariants
INPUT: a place subset $S_1$ and its generated sub-matrix $A_{S_1}$
OUTPUT: $S_1$ is a minimal support of S-invariants or not
PROCEDURES:
  Step1: Transform $A_{S_1}$ to a ladder-type matrix and compute the rank of $A_{S_1}$
         with the elementary row transformations method [10];
  Step2: If $R\left(A_{S_1}\right) = |S_1| - 1$ {
    Step3: Computing a non-trivial solution of $A_{S_1} Y_{S_1} = \vec{0}$ with Gaussian
           elimination method[10];
    Step4: If the solution is positive or negative, output that $S_1$ is a minimal
           support of S-invariants and exit;
  }
  Step5: Output $S_1$ isn't a minimal support of S-invariants.
```

We perform Algorithm 1 with $S_2 = \{s_3, s_4, s_5, s_6, s_7\}$ and its generated sub-matrix A_{S_2} in Fig. 2 as input.

First, with the elementary raw transformation method, A_{S_2} can be transformed to a ladder-type matrix as follows.

$$A_{S_2} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 \\ 0 & -1 & 1 & 1 & 0 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & -1 & 1 \end{bmatrix} \xrightarrow[r8+r7]{r4+r3} \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 1 & 0 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Obviously, $R(A_{S_2}) = 4 = |S_2| - 1$ holds according to the adder-type matrix above. Starting from the ladder-type matrix and using Gaussian elimination method, we can compute one solution $Y_{S_2} = [2 \quad 2 \quad 1 \quad 1 \quad 1]^T$ of $AY_{S_2} = \vec{0}$. The solution is positive, so S_2 is a minimal support of S-invariants .

In Algorithm 1, the time complexity of Step1 is $O\left(|S_1|^2 |T|\right)$ [10]. For Step 3, it is $O\left(|S_1|^2\right)$ [10]. For Step 4, it is $O\left(|S_1|\right)$. So the time complexity of Algorithm 1 is $O\left(|S_1|^2 |T|\right)$.

V. Computation Of A Minimal-supported S-Invariant

According to Theorem 4, if S_1 is a minimal support of S-invariants, then $R\left(A_{S_1}\right) = |S_1| - 1$. In this case, the great linearly independent group of A_{S_1}'s row spaces must include $|S_1| - 1$ row vectors[10]. Denote the matrix composed of these row vectors by *C*. Then *C* must be a $(|S_1| - 1) * |S_1|$ ordered full-row-rank matrix which satisfies $R(C) = R(A_{S_1}) = |S_1| - 1$. And the equation $CY_{S_1} = \vec{0}$ is equivalent to $A_{S_1} Y_{S_1} = \vec{0}$ because the relationship between *C* and A_{S_1} [10]. Since $R(C) = |S_1| - 1$ the great linearly independent group of C's column spaces must include $|S_1| - 1$ column vectors. Denote the matrix composed of these column vectors by *D*. Then *D* must be an $(|S_1| - 1) * (|S_1| - 1)$ ordered full-rank square matrix which satisfies $R(D)=R(C)=\left|S_1\right|-1$. Compared to *C*, *D* is obtained by deleting one column vector from *C*. Assume the deleted column vector is the *j*th column of *C* and denote it by C_j.

Next we will construct a non-trivial integer solution of $A_{S_1}Y_{S_1}=\vec{0}$ with *D*, C_j and *j* defined above.

***Lemma* 4** For the equation $A_{S_1}Y_{S_1}=\vec{0}$, denote the *k*th element of Y_{S_1} by $y_{S_1:k}$. Let $y_{S_1:j}=\det(D)$. And for $\forall i\in\left[1,\left|S_1\right|\right]\wedge i\neq j$, let $y_{S_1:i}=-\det(D_i(C_j))$, where $\det(D)$ is the determinant of *D*, $D_i(C_j)$ denote the resulted matrix after replace the *i*th column of *D* with vector *C**j*. Then $\widehat{Y}_{S_1}=\left[y_{S_1:1},y_{S_1:2},\ldots,y_{S_1:|S_1|}\right]^T$ is a non-trivial integer solution of $A_{S_1}Y_{S_1}=\vec{0}$.

Proof : Denote the *k*th column vector of the matrix C by C_k. Denote the resulted vector after deleting the *j*th element from Y_{S_1} by $Y_{S_1/j}$. According to the relationship between *D* and *C*, $CY_{S_1}=\vec{0}$ has the following equivalent transformations:

$$CY_{S_1}=\vec{0}\Leftrightarrow\left[C_1,C_2,\ldots,C_{|S_1|}\right]\begin{bmatrix}y_{S_1:1}\\y_{S_1:2}\\\cdots\\y_{S_1:|S_1|}\end{bmatrix}\Leftrightarrow\sum_{i=1}^{|S_1|}y_{S_1:i}*C_i=\vec{0}$$

$$\Leftrightarrow\sum_{i=1,\ldots,j-1,j+1,\ldots|S_1|}y_{S_1:i}*C_i+y_{S_1:j}*C_j=\vec{0}\Leftrightarrow DY_{S_1/j}+y_{S_1:j}*C_j=\vec{0}$$

$$\Leftrightarrow DY_{S_1/j}=-y_{S_1:j}*C_j$$

Let $y_{S_1:j}=\det(D)$, then the above equation becomes $DY_{S_1/j}=-\det(D)*C_j$. Because *D* is a full-rank matrix, the solution of $DY_{S_1/j}=-\det(D)*C_j$ is unique. According to the Cramer's rule[10], the *i*th element in the solution is as follows:

$$y_{S_1:i}=\frac{\det(D_i(-\det(D)*C_j))}{\det(D)}=-\det(D_i(C_j))$$

where $i=1,2,\ldots,j-1,j+1,\ldots\left|S_1\right|$.

Thus, $y_{S_1:i}=-\det(D_i(C_j))$ $\left(i=1,2,\ldots,j-1,j+1,\ldots\left|S_1\right|\right)$ constitutes a solution of $DY_{S_1/j}=-y_{S_1:j}*C_j$ when $y_{S_1:j}=\det(D)$. Because $DY_{S_1/j}=-y_{S_1:j}*C_j$ is equivalently transformed from $CY_{S_1}=\vec{0}$, so $y_{S_1:i}=-\det(D_i(C_j))$ $\left(i=1,2,\ldots,j-1,j+1,\ldots\left|S_1\right|\right)$ and $y_{S_1:j}=\det(D)$ constitute one solution of $CY_{S_1}=\vec{0}$ (i.e., $\widehat{Y}_{S_1}$ is a solution of $CY_{S_1}=\vec{0}$). In addition, because $A_{S_1}Y_{S_1}=\vec{0}$ and $CY_{S_1}=\vec{0}$ are equivalent, so $\widehat{Y}_{S_1}$ is also a solution of $A_{S_1}Y_{S_1}=\vec{0}$.

Furthermore, according to the construction method of D and C_j, their elements come from A_{S_1}. According to Definition 1, the elements of A_{S_1} are all integers, so D and $D_i(C_j)$ are integer matrixes. Hence $y_{S_1:j}=\det(D)$ and $y_{S_1:i}=-\det(D_i(C_j))$ are all integers. Finally, $y_{S_1:j}=\det(D)$ is nonzero because D is full-ranked. Therefore, $\widehat{Y}_{S_1}$ is a non-trivial integer solution of $A_{S_1}Y_{S_1}=\vec{0}$. ■

According to Theorem 4, the solution $\widehat{Y}_{S_1}$ constructed in Lemma 4 must be positive or negative integer vector. If $\widehat{Y}_{S_1}$ is positive let $\tilde{Y}_{S_1}=\widehat{Y}_{S_1}$, else let $\tilde{Y}_{S_1}=-1*\widehat{Y}_{S_1}$. Then $\tilde{Y}_{S_1}$ must be a positive integer solution of $A_{S_1}Y_{S_1}=\vec{0}$. Each element of $\tilde{Y}_{S_1}$ corresponds to a place in S_1. For each place in $S-S_1$, we add a zero element to $\tilde{Y}_{S_1}$ and denote the resulted vector by $\tilde{Y}$. Then it's obviously that $\tilde{Y}$ is an S-invariants supported by S_1. As a result, we can get the following algorithm.

***Algorithm* 2** Computation of a minimal-supported S-invariant

INPUT: a minimal support of S-invariants S_1 and its generated sub-matrix A_{S_1}

OUTPUT: one integer S-invariant supported by S_1

PROCEDURES:

Step1: Get the great linearly independent group of A_{S_1}'s row spaces with elementary column transformation method[10]. Denote the matrix composed of these row vectors by *C;*

Step2: Get the great linearly independent group of *C*'s column spaces with elementary row transformation method[10] . Denote the matrix composed of these column vectors by *D;*

Step3: In fact, *D* is obtained by deleting one column vector from *C*. Assume the deleted column vector is the *j*th column of *C* and denote it by C_j ;

Step4: For equation $A_{S_1}Y_{S_1}=\vec{0}$, denote the *k*th element of Y_{S_1} by $y_{S_1:k}$. Let $y_{S_1:j}=\det(D)$. And for $\forall i\in\left[1,\left|S_1\right|\right]\wedge i\neq j$, let $y_{S_1:i}=-\det(D_i(C_j))$. Construct the vector $\widehat{Y}_{S_1}=\left[y_{S_1:1},y_{S_1:2},\ldots,y_{S_1:|S_1|}\right]^T$;

Step5: If $\widehat{Y}_{S_1}$ is positive, let $\tilde{Y}_{S_1}=\widehat{Y}_{S_1}$,else let $\tilde{Y}_{S_1}=-1*\widehat{Y}_{S_1}$;

Step6: Denote the element of $\tilde{Y}_{S_1}$ corresponding to place $s_i\in S_1$ by $\tilde{y}_{S_1}(s_i)$ .Construct an |*S*|-dimensional column vector $\tilde{Y}$ as follows:

$$\tilde{y}(s_j)=\begin{cases}\tilde{y}_{S_1}(s_j) & if\ s_j\in S_1\\ 0 & if\ s_j\notin S_1\end{cases}$$

where $\tilde{y}(s_j)$ is the element of $\tilde{Y}$ which corresponds to $s_j\in S$;

Step7: Output that $\tilde{Y}$ is an S-invariants supported by S_1 .

Considering the Petri net and $S_2=\{s_3,s_4,s_5,s_6,s_7\}$ in Fig.2. In Step1, the great linearly independent group of A_{S_2}*'s* row spaces constitutes *C* as follows.

$$C=\begin{bmatrix}-1&1&0&0&0\\0&-1&1&1&0\\0&0&-1&1&0\\0&0&0&1&-1\end{bmatrix}\quad D=\begin{bmatrix}-1&1&0&0\\0&-1&1&1\\0&0&-1&1\\0&0&0&1\end{bmatrix}\quad C_5=\begin{bmatrix}0\\0\\0\\-1\end{bmatrix}$$

In Step2, the great linearly independent group of *C*'s column spaces constitute *D* shown above. In Step3, it's easy to see the 5th column of C is deleted in *D*. So *j=5* and C_5 is shown above.

In Step 4, the elements in $\widehat{Y}_{S_2}$ are as follows:

$$y_{S_2:5}=\det(D)=\begin{vmatrix}-1 & 1 & 0 & 0\\ 0 & -1 & 1 & 1\\ 0 & 0 & -1 & 1\\ 0 & 0 & 0 & 1\end{vmatrix}=-1;\quad y_{S_2:1}=-\det(D_1(C_5))=-1*\begin{vmatrix}0 & 1 & 0 & 0\\ 0 & -1 & 1 & 1\\ 0 & 0 & -1 & 1\\ -1 & 0 & 0 & 1\end{vmatrix}=-2$$

Similarly, $y_{S_2:2}=-\det(D_2(C_5))=-2$, $y_{S_2:3}=-\det(D_3(C_5))=-1$, $y_{S_2:4}=-\det(D_4(C_5))=-1$. Thus $\widehat{Y}_{S_2}=\begin{bmatrix}-2 & -2 & -1 & -1 & -1\end{bmatrix}^T$.

In step 5, $\tilde{Y}_{S_2}=-1*\widehat{Y}_{S_2}=\begin{bmatrix}2 & 2 & 1 & 1 & 1\end{bmatrix}^T$.

In step 6, the elements of $\tilde{Y}$ is as follows:

$$\tilde{y}(s_1)=0,\ \tilde{y}(s_2)=0,\ \tilde{y}(s_3)=\tilde{y}_{S_1}(s_3)=2,\ \tilde{y}(s_4)=\tilde{y}_{S_1}(s_4)=2,$$
$$\tilde{y}(s_5)=\tilde{y}_{S_1}(s_5)=1,\ \tilde{y}(s_6)=\tilde{y}_{S_1}(s_6)=1,\ \tilde{y}(s_7)=\tilde{y}_{S_1}(s_7)=1$$

Thus we get an S-invariant $\tilde{Y}=\begin{bmatrix}0 & 0 & 2 & 2 & 1 & 1 & 1\end{bmatrix}^T$ supported by $S_2=\{s_3,s_4,s_5,s_6,s_7\}$.

In Algorithm 2, th e time complexity of Step 1 is $O(|S_1|^2|T|)$ [10]. For Step 2, it is $O(|S_1|^3)$ [10]; For Step 3, it is $O(|S_1|)$; For Step 4, it is $O(|S_1|^4)$ [10]; For Step 5-7, the time complexity is all $O(|S|)$. So the time complexity of Algorithm 2 is $O(|S_1|^2|T|+|S_1|^4+|S|)$.

VI. Case Study

Taking Petri net Σ_1 and place su bset $S_2=\{s_1,s_4,s_5\}$ in Fig.1 as a case.

$$A_{S_2}=\begin{bmatrix}-2 & 0 & 1\\ 0 & 0 & 0\\ 0 & 1 & -1\\ 2 & -1 & 0\end{bmatrix}\xrightarrow[r_4+r_3]{r_4+r_1}\begin{bmatrix}-2 & 0 & 1\\ 0 & 0 & 0\\ 0 & 1 & -1\\ 0 & 0 & 0\end{bmatrix}\quad C=\begin{bmatrix}-2 & 0 & 1\\ 0 & 1 & -1\end{bmatrix}\quad D=\begin{bmatrix}-2 & 0\\ 0 & 1\end{bmatrix}$$

With Algorithm 1, A_{S_2} can be trans formed to a ladder-type matrix shown above. Obviously, $R(A_{S_2})=2=|S_2|-1$ holds. With Gaussian elimination method we get $Y_{S_2}=\begin{bmatrix}1/2 & 1 & 1\end{bmatrix}^T$ which is a solu tion of $A_{S_2}Y_{S_2}=\vec{0}$. The solution is positi ve, so S_2 is a minimal support of S-invariants.

With Algorithm 2, the parameters *C,D* are shown above too. Based on this, we get that $C_3=\begin{bmatrix}1 & -1\end{bmatrix}^T$ and the elements in $\widehat{Y}_{S_2}$ are as follows:

$$y_{S_2:3}=\det(D)=\begin{vmatrix}-2 & 0\\ 0 & 1\end{vmatrix}=-2;\quad y_{S_2:1}=-\det(D_1(C_3))=\begin{vmatrix}1 & 0\\ -1 & 1\end{vmatrix}=-1;$$

Similarly $y_{S_2:2}=-\det(D_2(C_3))=-2$ .Thus we get $\widehat{Y}_{S_2}=\begin{bmatrix}-1 & -2 & -2\end{bmatrix}^T$ and $\tilde{Y}_{S_2}=-1*\widehat{Y}_{S_2}=\begin{bmatrix}1 & 2 & 2\end{bmatrix}^T$.

Eventually we get a S-invariant $\tilde{Y}=\begin{bmatrix}1 & 0 & 0 & 2 & 2 & 0\end{bmatrix}^T$ supported by $S_2=\{s_1,s_4,s_5\}$.

VII. Conclusions

In this paper, after a deep research to th e properties of supports of S-invariants, a sufficient and necessary condition for a place s ubset to be a m inimal support of S-invariants is obtained. Based on the condition, two polynomial algorithms for the decidability of minimal supports of S-invariants and for the computation of an S-invariant supported by a given minimal support are presented.

Compared to those existed methods for the computation of S-invariants, the algorithms presented in this paper have polynomial time complexity. And based on these algorithms, we can further give some effective algorithms for the S-coverability verification of workflow nets, as well as th e decidability of the existence of S-invariants.

References

[1] Claude Girault, Rüdiger Valk. Petri Nets for Systems Engineering : A Guide to Modeling,Verification and Applications[M]. Springer-Verlag , Berlin Heidelberg, 2003.

[2] Wil van der Aalst, Christian Stahl. Modeling Business Processes: A Petri Net-Oriented Approach. MIT Press,May,2011.

[3] T.Murata,"Petri nets:properties,analysis and application", Proc.IEEE, vol.77, no.4, pp.541-579, 1989.

[4] Zhehui Wu. Introduction to Petri net. Beijing: China Machine Press.2005.

[5] Q.W.Ge, T.Tanida, and K.Onaga.Construction of a T-base and design of a periodic firint sequence of a Petri net. In Proc.8th Mathematical Programming symposium, Japan,pp.51-57,November 1987.

[6] J.Martinez and M.Silva. A Simple and Fast Algorithm to Obtain All Invariants Of a Generalized Petri Nets[J] Proceedings of S econd European Workshop on Application and Theory of Petri Nets,Informatik Fachberichte 52,Springer Publishing Company,Berlin,1982

[7] Maki Takata,Tadashi Matsumoto and Sciichiro Moro. A Direct Method to Derive All Generators of Solutions of a Matrix Equation in a Petri Net·Extended Fourier-Motzkin Method [J],the 2002 International Technical Conference on Circuits/systems,Computers and Communitions,2002.

[8] M.Yamauchi, M.wakuda, S.Taoka, and T.Watanabe. A Fast and Space-Saving Algorithm for Computing Invariants of Petri Nets. Systems, Man, and Cybernetics, 1999. IEEE SMC '99,Vol 1,pages:866-871

[9] Akihiro TAGUCHI , Atsushi IRIBOSHI, Satoshi TAOKA, and ToshimasaWATANABE. Siphon-Trap-Based Algorithms for Efficiently Computing Petri Net Invariants. IEICE Trans. Fundamentals,2005, E88–A(4):964-971

[10] David C.Lay. Linear algebra and its applications. 4th Edition[M]. Addison-Wesley , 2012:12-124.

[11] S.Tanimoto, M.Yamauchi,,T.Watanabe. Finding minimal siphons in general Petri nets[J].IEICE Trans.Fundamentals,1996,E79-A(11): 1817-1824.

[12] Faming Lu. An algebraic method for the reachability analysis of Petri nets [D].Qingdao, China: ShanDong Universitry of Science and Technology. 2006.

Automated Generation of Concurrent Test Code from Function Nets

Dianxiang Xu
National Center for the Protection of the Financial
Infrastructure, Dakota State University
Madison, SD 57042, USA
dianxiang.xu@dsu.edu

Janghwan Tae
Intelligent Computing Lab
Samsung Advanced Institute of Technology
Yongin-si, Korea
janghwan.tae@samsung.com

Abstract—**The advances in multi-core computing are promoting a paradigm shift from inherently sequential to truly concurrent applications. Testing concurrent applications, however, is more labor-intensive. In this paper, we present an approach to automated generation of concurrent test code from function nets, which are lightweight high-level Petri nets. We generate concurrent test sequences automatically from function nets. Based on a mapping of the modeling elements to implementation constructs, we further transform the test sequences into test code in C/pthread, which can be executed immediately against the system under test. We have implemented our approach based on MISTA, a framework for model-based integration and system testing. This paper also demonstrates the technical feasibility of our approach through some case studies.**

Keywords- ***software testing, model-based testing, concurrent programming, Petri nets, high-level Petri nets, test automation***

I. INTRODUCTION

The advances in multi-core computing are promoting a paradigm shift from inherently sequential to truly concurrent applications. To utilize the power of multi-core technologies, concurrent programming is critical. It is well-known, however, concurrent programming is more error-prone. As an important means for quality assurance, software testing is very labor-intensive. Various studies have reported that software testing and related activities account for more than 50% of the total development costs. It is highly desirable to automate the software testing process. Automated testing can significantly improve overall productivity and reduce costs. For example, model-based testing uses explicit behavior models of software for generating test cases. As model-based testing can automate or partially automate test generation, many test cases can be created, executed, and repeated. Although automated test generation and execution have a great potential, the extent of automation in the existing model-based testing approaches remains limited. In particular, there is little research on model-based generation of concurrent tests.

In this paper, we present a model-based approach to automated generation of concurrent test code. In this approach, function nets, which are lightweight high-level Petri nets [1], are used as the modeling notation for building test models of concurrent programs. High-level Petri nets have been widely applied to modeling and analysis of distributed systems [2]. Concurrent test sequences and multi-threaded test code are generated automatically from the function nets. We have implemented this approach based on MISTA (formerly ISTA) [3][1], a framework for model-based integration and system testing. MISTA allows executable test code to be generated automatically from a MID specification, which consists of a function net and a mapping of the modeling elements to implementation constructs. In this paper, we enhance MISTA for testing concurrent programs from the following perspectives. First, to facilitate generating concurrent tests, new notations (regions and sinks) are introduced to function nets for building test models of concurrent programs. Second, new algorithms are developed for generating concurrent test sequences from test models. Third, new algorithms are developed for generating concurrent test code in C.

The remainder of this paper is organized as follows. Section II introduces function nets as test models of concurrent programs. Sections III and IV present automated generation of test sequences and pthread test code. Section V describes case studies. Section VI reviews the related work. Section VII concludes this paper.

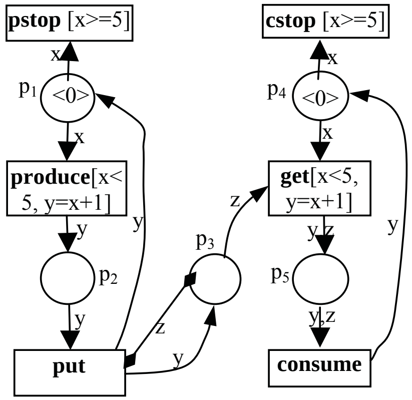
II. FUNCTION NETS AS TEST MODELS OF CONCURRENT PROGRAMS

A. Function Nets: Structure and Semantics

Function nets are lightweight high-level Petri nets. A function net consists of places (circles), transitions (rectangles), arcs, arc labels, guard conditions of transitions, and initial marking. *p* is an input place (or precondition) of transition *t* if there is an arc from *p* to *t*; *p* is an output place (or postcondition) of *t* if there is an arc from *t* to place *p*. If the arc is bi-directional, *p* is both input and output place (or precondition and postcondition) of *t*. An inhibitor arc between *p* and *t* is represented by a line segment with a little diamond on both ends (*p* is called an inhibitor place or negative precondition of *t*). The label of an arc is a tuple of variables and constants. Suppose variables start with a lower-case letter or the question mark (?). If an arc is not labeled, the default arc label is the no-argument tuple, denoted as <>. The guard

[1] The beta release of MISTA v1.0 is available at http://www.homepages.dsu.edu/dxu/research/MBT.html

condition of a transition is a logical formula built from arithmetic or relational operations. A transition may be associated with a list of variables as its formal parameters. A marking is a set of tokens distributed in all places. A token is a tuple of constants. We represent token $<V_1, \ldots, V_n>$ in p as $p(V_1, \ldots, V_n)$. No-argument token (<>, or little solid circle or dot in traditional Petri nets) in p is simply denoted as p. Figure 1 shows an example, where initial marking is $\{p_1(0), p_4(0)\}$.



$M_0 = \{p_1(0), p_4(0)\}$
region(consumer)={get,consume,cstop}
region(producer)={produce,put, pstop}
sinks = {pstop, cstop}

Figure 1. A test model for producer/consumer

A variable substitution θ is a set of variable bindings $\{x_i=V_i\}$, where variable x_i is bound to value V_i. Let l be an arc label. l/θ denotes the tuple (or token) obtained by substituting each variable in l for its bound value in θ. A transition t is said to be enabled by variable substitution θ under marking M_0 if (1) each input place p of t has a token l/θ, where l is the label of the arc from p to t; (2) each inhibitor place p of t has no token l/θ, where l is the label of the inhibitor arc between p and t; (3) t's guard condition evaluates true with respect to the substitution θ.

Firing an enabled transition t with θ removes the matched token from each input place, and adds a token l/θ to each output place p, where l is the label of the arc from t to p. This leads to a new marking. A firing sequence is denoted as $M_0[t_1\theta_1>M_1[t_2\theta_2>M_2 \ldots [t_n\theta_n>M_n$, or simply $t_1\theta_1t_2\theta_2 \ldots t_n\theta_n$, where firing transition t_i with substitution $\theta_i (1 \le i \le n)$ under M_{i-1} leads to M_i $(1 \le i \le n)$. A marking M is said to be reachable from M_0 if there is such a firing sequence that leads to M from M_0.

A function net captures system behaviors through state transitions from a given initial state (marking). An initial marking determines the entry point (s), i.e., what transitions can be fired at the initial state. We also allow a function net to have one or more "*sink*" transitions, similar to termination states or exit points in other models such as finite state machines and activity diagrams. The behaviors of a function net can be interpreted by its reachability tree (graph). To facilitate test generation from different data sets, our approach allows a function net to be associated with multiple initial markings. Therefore, the root of a reachability tree represents a dummy state, whose child nodes are built from the given initial markings. The construction of a reachability tree starts with expanding each initial marking node. Each node is expanded as follows: (1) find *all possible* firings $t_i\theta_j$ under the marking of the current node under expansion; (2) for each firing $t_i\theta_j$, create a new child node according to the new marking. The edge from the current node to the new child is labeled with $t_i\theta_j$. (3) if t_i is not a sink transition and the new marking has not yet expanded in the sub-tree of the same initial marking, then expand the new child node.

In MISTA, partial ordering of concurrent (or independent firings) and pairwise combination of input values are two techniques for reducing the complexity of reachability tree. When the partial ordering of concurrent firings is used, step (2) only selects one firing from a set of concurrent firings. When pairwise combination is used , all pairs (rather than all combinations) of input values are used to derive firings in step (1). We use a *region* to represent a collection of events in the same thread (or process) of a concurrent system. For example, *region (consumer)={get, consume, cstop}*. Building the reachability tree for a region uses the transitions listed in the given region.

B. Function Nets as Test Models

Function nets can represent various building blocks of software models, such as sequence, condition (choice), repetition (loop), concurrency, synchronization and mutual exclusion, structured data, arithmetic and relational operations, and modularization (hierarchy). These building blocks can be used to compose complex test models of concurrent programs. Consider bounded buffer as an example. A bounded buffer is shared by different producer and consumer threads. A producer thread puts data into the shared buffer, whereas a consumer thread gets data from the shared buffer. To function correctly, the implementation of bounded buffer must synchronize the concurrent "*put*" and "*get*" actions from different threads. Figure 1 shows a test model of bounded buffer, where a *producer* produces and puts five items to the buffer and *a consumer* gets and consumes five items from the buffer. The *producer* is not intended to put any item to the buffer if the buffer is not empty. The *consumer* is not intended to get an item from the buffer unless the buffer has an item . The "*produce*" and "*put*" actions in the *producer* thread are sequential; the "*get*" and "*consume*" actions in the *consumer* thread are sequential. They are represented by two sequential structures. They are included in the two loop structures, respectively. "*put*" has two postconditions: updating the loop control value and depositing the produced item into the buffer (place p_3). *Producer* and *consumer* have concurrent actions (e.g., *produce* and *consume*). They are synchronized because of the shared buffer.

Although building a test model depends on what needs to be tested against a specific system under test (SUT), there are two different perspectives: whitebox and blackbox. A white test model is similar to the design model of a SUT – what is described in the model is supposed to be implemented by the SUT. For example, Figure 1 would be a whitebox test model for the producer/consumer program if a SUT has implemented the producer and consumer threads and these threads are the target of testing. A blackbox test model does not describe how a SUT is implemented. Rather, it focuses on how a SUT will be tested from a blackbox perspective according to the interface. Figure 1 would be a blackbox test model for bounded buffer. The model describes how bounded buffer will be tested through calls to *put* and *get*. Bounder buffer is like a black box to the test model. As will be discussed in Section III, whitebox and blockbox models are used for different testing purposes.

III. Automated Generation of Test Sequences

A. *Generating Deterministic Test Sequences from WhiteBox Test Models*

A deterministic test sequence of a concurrent program (or syn-sequence [4]) is of the form, $<r_0, t_0\theta_0, M_0,>\ldots, <r_{n-1}, t_{n-1}\theta_{n-1}, M_{n-1},>$, where r_i $(0\leq i<n)$ is a region, $t_i\theta_i$ is a firing (test input) in region r_i, and M_i is the resultant marking (test oracle). Deterministic test sequences are used for deterministic testing or reachability testing of a concurrent program – forcing the execution of a concurrent program to follow the specified order of events. This can be done by using a replay tool [4]. Our approach generates deterministic test sequences from the reachability tree of a whitebox test model. To reduce the number of sequences, the partial ordering technique can be used to generate only one sequence from each set of concurrent firings. To derive test sequences, we retrieve all leaf nodes from the reachability tree , and, for each leaf node, obtain the test sequence from the initial marking node to the leaf node. For example, <producer, produce(1)>, <producer,put(1)>, <consumer, get>, <consumer, consume>, <producer, produce(2)>, <producer, put(2)>… is a deterministic sequence generated from the model in Figure 1.

B. *Generating Nondeterministic Test Sequences from Blackbox Models*

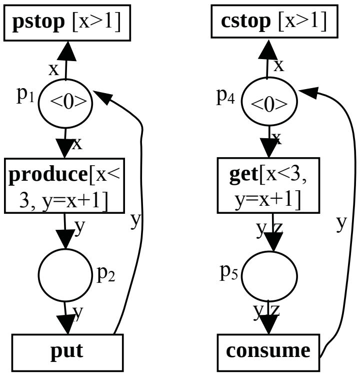
A nondeterministic test sequence is of the form , $<r_0, F_0>,\ldots, <r_{n-1}, F_{n-1}>$, where r_i $(0\leq i<n)$ is a region (thread) and F_i is a firing sequence in region r_i. It means that the transition firings in different regions are concurrent. To generate nondeterministic test sequences, we first determine whether or not the regions in a given blackbox test model are independent (this paper will not elaborate on this due to limited space). If they are independent, we generate nondeterministic sequences by parallel composition of all firing sequences from the partial-ordering-based reachability trees of individual regions; otherwise we first generate deterministic sequences from the partial-ordering-based reachability tree of the entire test model and convert them into non-deterministic ones. The partial ordering technique is very efficient because it avoids the interleaving of concurrent firings from independent regions.

Algorithm 1 describes the generation of nondeterministic test sequences from a blackbox test model with independent regions. It first creates a partial-ordering-based reachability tree for every region (lines 3 -7), which includes the firing sequences in the respective region. Then it generates concurrent test sequences for each initial marking by parallel composition of firing sequences from different regions (lines 8-28). Lines 9-16 create a new list of trees that includes only the trees that contain test sequences derived from the given initial marking. The size of this new tree list indicates the number of threads of test code. Lines 17 -20 compute (non-deterministic) test sequences of all trees in the list. Each combination is represented by the sequence indices in the respective trees.

```
Algorithm 1. Generating nondeterministic test sequences from
a test model with independent regions
Input: PrT net with independent regions
Output: nondeterministic (concurrent) test sequences
Declare: tree is a regional partially-ordered reachability tree;
   treeList, newTreeList are lists of regional reachability trees
  totalSeqsOfIndThreads is integer array
  combSeqs is a two-dimensional array
  currentSeq is a nondeterministic test sequence
  testSeqs is a list of nondeterministic test sequences:
1.  begin
2.  testSeqs ← ∅
3.  treeList ← ∅
4.  for each region do
5.      tree ← generate a reachability tree for the region
6.      add tree to treeList
7.  end for
8.  for each initial marking do
9.      newTreeList ← ∅
10.     for each tree in treeList
11.         if tree has test sequences with respect to the
    current initial marking
12.             add tree to newTreeList
13.         end if
14.     end for
15.     if newTreeList = ∅
16.         continue (to next initial marking)
17.     for i=0 to newTree.size-1 do
18.         totalSeqOfIndThreads[i] ← number of sequences in
    newTree(i)
19.     end for
20.     combSeqIndices ← all combinational sequence indices
    from totalSequencesOfIndividualThreads
21.     for i=0 to combSeqIndices.length-1 do
22.         currentSeq ← ∅
23.         for regionIndex=0 to newTree.size-1
24.             currentSeq ← currentSeq + < newTree
    (regionIndex).region, sequence from newTree(regionIndex)
    whose index is combSeqIndices[i][regionIndex] >
25.         end for
26.         add currentSeq to testSeqs
27.     end for
28. end for
29. end
```

The test model in Figure 2 yields the following nondeterministic test sequences:

1. <producer<produce(1), put(1), produce(2), put(2), pstop >>,<consumer,<get, consume, get, consume, cstop>>
2. <producer,<produce(1),put(1),produce(2),put(2),pstop>, <consumer,<get, consume, get, consume, get, consume, cstop>
3. <producer,<produce(1),put(1),produce(3), put(2), produce(3), put(3), pstop>, <consumer, <get,consume, get,consume, cstop>>
4. <producer,<produce(1),put(1), produce(2), put(2), produce(3), put(3), pstop>, <Consumer, <get, consume, get, consume, get, consume, cstop>>



M_0= {p_1(0), p_4(0)}
region(consumer)={get,consume, cstop}
region(producer)={produce,put, pstop}
sinks = {pstop, cstop}

Figure 2. A test model with independent regions

When the regions are interdependent, we first generate deterministic sequences (similar to Section III-A) and transform them into concurrent sequences. For example, deterministic sequence <producer, produce(1)>, <producer, put(1)>, <consumer, get>, <consumer, consume>, <producer, produce(2)>, <producer, put(2), <consumer, get>, <consumer, consume> > can be transformed into the following non-deterministic sequence: <producer, <produce(1),put(1), produce (2), put(2)>>, <consumer, <get, consume, get, consume>>.

IV. AUTOMATED GENERATION OF TEST CODE

We generate test code from nondeterministic sequences based on model-implementation mapping (MIM).

A. MIM

A MIM specification for a test model consists of the following components [3]:

(1) *ID* is the identity of the SUT tested against the model.
(2) f_o is the object function that maps objects in the test model to objects in the SUT.
(3) f_e is the event (also called method) mapping function that maps events in the model to operations in the SUT.
(4) f_a is the accessor function that maps predicates in the test model to accessors in the SUT.
(5) l_h is the list of hidden predicates in the test model that produce no test code.
(6) *h* is the helper code function that defines user-provided code to be included in the test code.

ID refers to the system under test. The functions f_o, f_e, f_a and f_m map objects, events, and predicates to respective counterparts in the SUT. Operation for an event or accessor for a predicate is a block of code that can be executed . The helper code function allows the user to provide additional code required to make the generated tests executable.

TABLE I. A MIM SPECIFICIFICATION FOR BOUNDED BUFFER

SYSTEM	BoundedBuffer1	
HIDDEN (l_h)	p1, p2, p3, p4, pstop, cstop	

METHOD (f_e)	produce(?y)	pch = item[?y];
		printf("Producing %c...\n", pch);

	put(?y)	put(pch);
	get()	gch=get();
	consume()	printf("Consuming %c ... \n", gch);
#INCLUDE (h(include))	#include <pthread.h> …	

Table 1 shows an example. The system name is BoundedBuffer1, after which the files of test code will be named. Predicates p_1, p_2, p_3, and p_4 and transitions *pstop* and *cstop* in the test model are listed as "*hidden*", which means they do not produce test code. Transition produce(?y) is corresponding to the following code:

```
pch = item[?y];
printf("Producing %c ...\n", pch);
```

When generating test code, a transition firing is a function call or more generally a block of code. Variables (e.g., *?y*) will be substituted for the bound values in the variable substitution of the transition firing.

B. Algorithms for Test Code Generation

The code structure of a concurrent test based on pthreads consists of the header (including setup, teardown and local code), declarations of pthreads, definitions of pthread functions, and the main function (test driver). The header is provided by tester; other code is generated automatically according to the MIM specification. Algorithm 2 below generates a pthread function from a firing sequence in a region. A pthread function is a function that is used to create a pthread.

```
Algorithm 2. Generate a pthread function from a firing sequence of a region
Input: region, firing sequence [$t_1θ_1$>$M_1$[$t_2θ_2$>$M_2$ …[$t_nθ_n$>$M_n$, MIM (ID, $f_o$, $f_e$, $f_a$, $l_h$, h)
Output: pthread function code named after region (pfcode)
Declare: testInput is a string representing test input code
         testOracle is a string representing test oracle code
         newLine starts a new line
1.  begin
2.  pfcode ← "void *"+region+"(void * parm)"
3.  pfcode ← pfcode +"{"+ newLine
4.  for i=1 to n do
5.      if $t_i$ is not a hidden event, i.e., $t_i \notin l_h$
6.          let $t_iθ_i$ = $t_i$ ($V_1$…., $V_k$),$V_j$ is an actual parameter
7.          testInput ← $f_e$($t_i$ ($f_o$($V_1$),…, $f_o$($V_k$)));
8.      endif
9.      testOracle ← "";
10.     for each p($V_1$,…, $V_k$) in $M_i$ do
11.        if $p \notin l_h$
12.          testOralce ←testOracle + $f_a$(p($f_o$($V_1$),…, $f_o$($V_k$)));
13.        endif
14.     end for
15.     pfcode ← pfcode+testInput+testOralce
16. end for
17. pfcode ← pfcode+ newLine +"pthread_exit(0)"
18. pfcode ← pfcode+ newLine +"}"
19. end
```

A firing sequence is a sequence of nodes ($[t_iθ_i>M_i)$ in a reachability tree. Line 2 creates the signature of the function. Lines 3 and 18 provide "{}" to enclose the function. Lines 4-16 generates test inputs and oracles from the given firing sequence. Each $t_i\theta_i$ in the firing sequence is a test input if t_i is not a hidden event (Lines 5-8). Each predicate in M_i is corresponding to a test oracle if p is not hidden (Lines 11-13). $f_e(t_i(f_o(V_1),\ldots, f_o(V_k)))$ means that each model-level actual parameter V_j of transition t_i is first replaced with the corresponding implementation-level parameter from object mapping f_o and then the implementation code for transition t_i is obtained from f_e. $f_a(p(f_o(V_1),\ldots, f_o(V_k)))$ means that each model-level actual parameter V_j of p is first replaced with the corresponding implementation-level parameter from object mapping f_o and then the accessor code for transition t_i is obtained from event mapping f_e.

Algorithm 3 below generates a C file, which contains multiple pthreads for a nondeterministic test sequence.

```
Algorithm 3. Generate multi-threaded test code in C from a nondeterministic test sequence
Input: test ID, nondeterministic test sequence <r0, F0>,…, <rn-1, Fn-1>, MIM(ID, fo, fe, fa, lh, h)
Output: testCode file
Declare: fi is the thread function for Fi(generated by Algorithm2)
1.  begin
2.  testCode ← h
3.  for i=1 to n do
4.       testCode ← testCode + thread declaration for ri
5.  endfor
6.  for i=1 to n do
7.       fi ← call Algorithm 2 with Fi and MIM
8.       testCode ← testCode + fi
9.  endfor
10. testCode ← testCode + signature of main function
11. testCode ← testCode + "{"
12. testCode ← testCode + setup call if setup is defined
13. for i=1 to n do
14.      testCode ← testCode + thread creation for ri
15. endfor
16. for i=1 to n do
17.      testCode ← testCode + thread join for ri
18. endfor
19. testCode ← testCode + teardown call if is defined
20. testCode ← testCode + "}"
21. save test code to a file named after system ID + test ID
22. end
```

The code snippet below shows the test code for concurrent test sequence <producer <produce(1), put(1), produce(2), put(2), pstop>>, <consumer,<get, consume, get, consume, cstop>>.

```
#include <pthread.h>
…
pthread_t tidproducer;                  // thread declaration

void *producer(void * parm) {
  pch = item[1];                        // produce(1)
  printf("Producing %c ...\n", pch);    // produce(1)
  put(pch);                             // put(1)
  pch = item[2];                        // produce(2)
  printf("Producing %c ...\n", pch);    // produce(2)
  …
  pthread_exit(0);
}
void *consumer(void * parm)  {
  gch=get();                            // get
  printf("Consuming %c ... \n", gch); // consume
  …
}
main(int argc, char *argv[]) {
  pthread_create(&tidproducer, NULL, producer, NULL);
  …
  pthread_join(tidconsumer, NULL);
}
```

Obviously, Algorithm 3 can be called to generate test code for a set of non-deterministic test sequences, e.g., produced by Algorithm 1.

V. CASE STUDIES

We have implemented our approach based on the MISTA tool. Our case studies focus on demonstrating that our approach can generate concurrent test code from test models.

A. *Bounded Buffer*

Bounded buffer is a classical example for demonstrating the synchronization problem of multi-threaded programming. It involves two types of threads, *producer* and *consumer*, that share a fixed-size buffer. A *producer* generates a piece of data, puts it into the buffer, and starts again. At the same time a *consumer* consumes the data one piece at a time. Existing work often focuses on the testing of given *producer* and *consumer* programs together with the bounder buffer (Strictly speaking, *producer* and *consumer* are only one test for bounded buffer). Our work, however, focuses on the testing of the bounded buffer. We view *producer* and *consumer* programs as test cases for the bounded buffer. As shown in Section IV, we use test models to generate different kinds of producers and consumers. For example, a concurrent test may have different number of producers and consumers. The *producer* and *consumer* in one test may have different numbers of calls to put and get.

B. *ATM*

ATM [5] is a client/server program for managing ATM accounts, e.g., querying the current balance of an account, depositing money to an account, and withdrawing money from an account. The server program can accept requests from different clients by using multi-threaded programming. The multiple threads can access account data simultaneously. For the ATM program, testing can be done from the client side. For example, a test case can first deposit money to an account and then withdraw money from the same account. Such traditional test cases can be generated by the existing MISTA tool. In this study, we focus on generation of concurrent tests with multiple threads. Each thread sends requests to the server. Different threads may operate on the same account to check if the server fails to respond correctly.

For each case study, we were able to generate executable pthread code for many concurrent tests. This has demonstrated the technical feasibility. We are currently applying the approach to an industry project at Samsung.

VI. RELATED WORK

Our approach is related to code-based and specification-based testing of concurrent programs and testing with Petri nets. Due to limited space, this paper will not review the work on code-based testing or test execution frameworks of concurrent programs.

A. Specification-based Testing of Concurrent Programs

Carver and Tai [6][7] have developed a specification-based approach to testing concurrent programs. Sequencing constraints on succeeding and preceding events (CSPE) are used to specify restrictions on the allowed sequences of synchronization events. They described how to achieve coverage and detect violations of CSPE constraints based on a combination of deterministic and nondeterministic testing. Chung et al. [8] developed a specification-based approach to testing concurrent programs against Message Sequence Charts (MSCs) with partial and nondeterministic semantics. Based on the sequencing constraints on the execution of a concurrent program captured by MSCs, this approach verifies the conformance relations between the concurrent program and the MSC specification. Seo et al. [9] presented an approach to generating representative test sequences from statecharts that model behaviors of a concurrent program. Representative test sequences are a subset of all possible interleavings of concurrent events. They are used as seeds to generate automata that accept equivalent sequences. Wong and Lei [10] have presented four methods for generating test sequences that cover all the nodes in a given reachability graph. The reachability graph is typically constructed from the design of a concurrent program. Test sequences are generated based on hot spot prioritization or topological sort . It is not concerned with automated generation of data input or mapping of the abstract SYN-sequences derived from the given reachability graph to concrete implementation sequences. Different from the above work, our approach can generate executable multi-threaded test code from test models.

B. Testing with Petri Nets

Zhu and He [11] have proposed a methodology for testing high-level Petri nets. It is not concerned with how tests can be generated to meet the criteria. Lucio et al. [12] proposed a semi-automatic approach to test case generation from CO-OPN specifications. CO-OPN is a formal object-oriented specification language based on abstract data types and Petri nets. This approach transforms a CO-OPN specification into a Prolog program for test generation purposes.

VII. Conclusions

We have presented an approach to automated testing of concurrent pthread programs. It can automatically generate (non) deterministic test sequences as well as multi-threaded test code. These functions, together with the existing MISTA tool, can provide a useful toolkit for testing concurrent programs. It can reap two important benefits. First, software testing would not be effective without a good understanding about a SUT. Using our approach, testers will improve their understanding about the SUT while documenting their thinking through the creation of test models. Test models can clearly capture test requirements for software assurance purposes. Second, automated generation of test code from test models can improve productivity and reduce cost. When concurrent software needs many tests, the tests can be generated automatically from their test models. This can better deal with the frequent changes of requirements and design features because only the test models, not test code, need to be updated. The savings on the manual development of test code also allow testers to focus on the intellectually challenging aspect of testing – understanding what needs to be tested in order to achieve high-level assurance. Our case studies based on small programs are primarily used to demonstrate the technical feasibility. We expect to apply our approach to real-world applications that require a large number of concurrent tests in order to reach a high level of quality assurance.

Our future work will adapt the approach to automated generation of concurrent code in other languages and thread libraries, e.g., Java, C#, and C++. In addition, effectiveness of the test cases generated from test models essentially depends on the test models and test generation strategies. We plan to evaluate the effectiveness of our approach by investigating what types of bugs in concurrent programs can be revealed by the test cases generated from the test models. To this end, we will first define a fault model of concurrent programs in a given target language (e.g., C/pthread), create mutants by injecting faults in the subject programs deliberately, and test the mutants with the test code generated from the test models. A mutant is said to be killed if the test code reports a failure. Usually the percentage of the mutants killed by the tests is a good indicator of the fault detection capability of the test cases.

References

[1] H.J. Genrich, “Predicate/Transition Nets,” In: Brauer, W., Reisig, W., Rozenberg, G. (eds.) Petri Nets: Central Models and Their Properties. Springer-Verlag London, 1987, pp. 207-247.

[2] T. Murata, “Petri Nets: Properties, Analysis and Applications,” Proc. of the IEEE, vol. 77, no. 4, 1989, pp. 541-580.

[3] D. Xu, “A Tool for Automated Test Code Generation from High-Level Petri Nets”, Proc. of the 32nd International Conf. on Application and Theory of Petri Nets and Concurrency (Petri Nets 2011), LNCS 6709, Newcastle, UK, June 2011, pp. 308–317.

[4] R. H. Carver and K. C. Tai, Modern Multithreading: Implementing, Testing, and Debugging Multithreaded Java and C++/Pthreads/Win32 Programs, Wiley, 2006.

[5] D. Buttlar, J. Farrell, B. Nichols, PThreads Programming: A POSIX Standard for Better Multiprocessing, O'really Media, Sept 1996.

[6] R. H. Carver and K. C. Tai, “Use of Sequencing Constraints for Specification-based Testing of Concurrent Programs,” IEEE Trans. on Software Engineering, vol. 24, no. 6, 1998, pp. 471-490.

[7] R.H. Carver, K.C. Tai, "Test Sequence Generation from Formal Specifications of Distributed Programs," Proc. 15th IEEE International Conference on Distributed Computing Systems (ICDCS'95), 1995, pp. 360-367.

[8] I. S. Chung, H. S. Kim, H. S. Bae, Y. R. Kwon, and B. S. Lee, “Testing of Concurrent Programs based on Message Sequence Charts,” Proc. the International Symposium on Software Engineering for Parallel and Distributed Systems (PDSE '99), pp. 72-82.

[9] H. S. Seo, I. S. Chung, and Y. R. Kwon. “Generating Test Sequences from Statecharts for Concurrent Program Testing,” IEICE - Trans. Inf. Syst. E89-D, 4, 2006, pp. 1459-1469.

[10] W. E. Wong and Y. Lei. “Reachability Graph-Based Test Sequence Generation for Concurrent Programs,” International Journal of Software Engineering and Knowledge Engineering, vol. 18, no. 6, 2008, pp. 803-822.

[11] H. Zhu and X. He. "A Methodology for Testing High-Level Petri Nets," Information and Software Tech., vol.44, 2002, pp. 473-489.

[12] L. Lucio, L. Pedro, and D. Buchs. “Semi-Automatic Test Case Generation from CO-OPN Specifications,” Proc. of the Workshop on Model-Based Testing and Object-Oriented Systems, 2006, pp. 19-26.

SAMAT - A Tool for Software Architecture Modeling and Analysis

Su Liu, Reng Zeng, Zhuo Sun, Xudong He
School of Computing and Information Sciences
Florida International University
Miami, Florida 33199, USA
{sliu002, zsun003, rzeng001, hex}cis.fiu.edu

***Abstract*—A software architecture specification plays a critical role in software development process. SAM is a general framework for developing and analyzing software architecture specifications. SAM supports the scalability of architectural descriptions through hierarchical decomposition and the dependability analysis of architectural descriptions using a dual formalism based on Petri nets and temporal logic. In this paper, we present SAMAT[1] (Software Architecture Modeling and Analysis Tool), a tool to support the hierarchical modeling and analyzing of software architecture specifications in SAM. SAMAT nicely integrates two external tools PIPE+ for behavioral modeling using high-level Petri nets and SPIN for model checking system properties.**

Keywords: Petri Net; Modeling and Analysis Tool; SAM

I. Introduction

Since late 1980s, software architecture has become an active research area within software engineering for studying the structure and behavior of large software systems [16]. A rigorous approach towards architecture system design can help to detect and eliminate design errors early in the development cycle, to avoid costly fixes at the implementation stage and thus to reduce overall development cost and increase the quality of the systems. SAM [17], [8], [9], [10] is a general framework for systematically modeling and analyzing software architecture specifications. Its foundation is a dual formalism combining a Petri net model for behavioral modeling and a temporal logic [9] for property specification. To support the application of SAM framework, we are developing a tool set, called SAMAT.

SAMAT has the following features:

1) supporting software architecture modeling through hierarchical decomposition;
2) modeling software component and connector behaviors using high-level Petri nets [2];
3) specifying model constraints (system properties) using first-order linear time temporal logic [15];
4) analyzing the SAM's behavior model through model translation and model checking using SPIN [11].

In the following sections, we discuss our development of SAMAT and its main features. Section 2 gives an overview of the SAM framework as well as its foundation. Section 3 presents the components and functionality of SAMAT and the design of SAMAT. Section 4 provides the model translation process from SAM to PROMELA for model checking in SPIN. Section 5 shows an example to illustrate the use of SAMAT. Section 6 compares the SAM famework and SAMAT with related software architecture framework and tools. Section 7 contains a conclusion.

II. The SAM Framework

SAM [17] is a general formal framework for specifying and analyzing software architecture. SAM supports the hierarchical modeling of software components and connectors.

The architecture in SAM is defined by a hierarchical set of compositions, in which each composition consists of a set of components (rectangles), a set of connectors (bars) and a set of constraints to be satisfied by the interacting components. The component models describe the behavior (Petri net) and communication interfaces (called ports, represented by semicircles). The connectors specify how components interact with each other . The constraints define requirements imposed on the components and connectors, and are defined by temporal logic formulas.

Figure 1 shows a hierarchical SAM specification model. The boxes, such as “A1” and “A2”, are called compositions, in which “A1” is component and “A2” is connector. Each composition may contain other compositions, for example “A1” wrap up three compositions: “B1”, “B2” and “B3”. Each bottom-level composition is either a component or a connector and has a property specification (a temporal logic formula). The behavior model of each component or connector is defined using a Petri net. Thus, composing all the bottom-level behavior models of components and connectors implicitly derives the behavior of an overall software architecture. The intersection among relevant components and connectors such as “P1” and “P2” are called ports. The ports form the interface of a behavior model and consist of a subset of Petri net places.

A. The Foundation of SAM

The foundation of SAM is based on two complementary formal notations: predicate transition nets [6], [7] (a class of high-level Petri nets) and a first-order linear-time temporal logic [15].

[1] SAMAT can be downloaded at http://users.cis.fiu.edu/~sliu002/samat

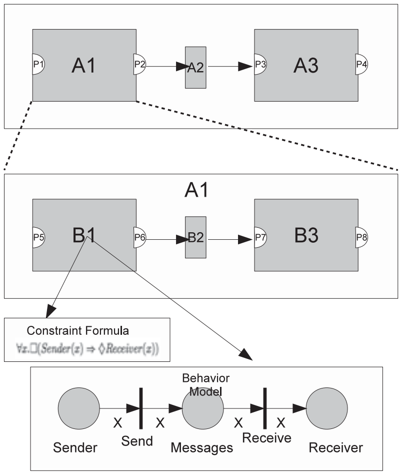


Figure 1. Hierarchical SAM Specification Model

1) Predicate transition nets (PrT nets), a class of high-level Petri nets, are used to define the behavior models of components and connectors. A PrT net comprises a net structure, an underlying specification and a net inscription [7]. A token in PrT nets contains typed data and the transitions are inscribed with expression as guard to move or stop the tokens.
2) First-order linear time temporal logic (FOLTL) is used to specify the properties (or constraints) of components and connectors. The vocabulary and models of our temporal logic are based on the PrT nets. Temporal formulae are built from elementary formulae (predicates and transitions) using logical connectives $\neg$ and $\wedge$ (and derived logical connectives $\vee$, $\Rightarrow$ and $\Leftrightarrow$), the existential quantifier $\exists$ (and derived universal quantifier $\forall$) and the temporal always operator $\Box$ (and the derived temporal sometimes operator $\Diamond$).

SAM supports the behavior modeling using PrT nets [14] and property specification using the FOLTL. SAM supports structural as well as behavioral analysis of software architectures. Structural analysis is achieved by enforcing the completeness requirement imposed on the components and connectors within the same composition, and the consistency requirement imposed on a component and its refinement. Behavioral analysis requires the checking of system properties in FOLTL satisfied in the behavioral models in PrT nets. In this paper, we analyze SAM behavior model by leveraging SPIN [11], which is a popular formal verification tool used worldwide.

III. SAMAT

In this section, we present the functional view and the design view of SAMAT.

A. Functional View of SAMAT

SAMAT is comprised of a modeling component, a SAM model, and an analysis component (Figure 2). The modeling component has three functions: structure modeling creates hierarchical compositions, behavior modeling specifies behaviors of software components/connectors using Petri nets, and property modeling defines property specifications using temporal logic. The SAM specification is a hierarchical structure integrating the results of structure, behavior, and property modeling, which can be transformed into XML format. The analysis component contains a translator to generate a model suitable for model checking.

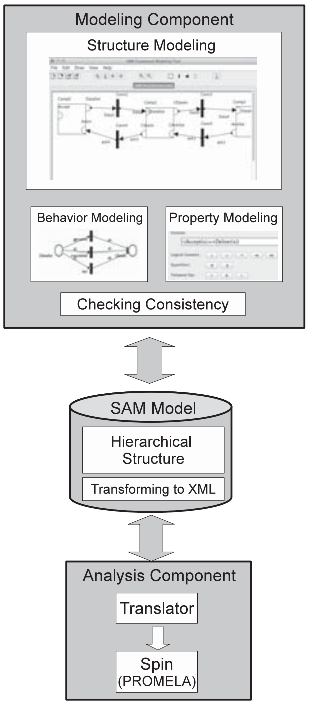


Figure 2. The Functional View of SAMAT

B. Design View of SAMAT

SAMAT is a platform independent (implemented in Java) and visual software architecture modeling and analysis tool. As shown in Figure 3, SAMAT is designed using the Model-Vew-Control pattern.

1) The model of SAMAT includes a hierarchical layer of SAM compositions that builds the SAM model in Figure 2. It also include the functionalities of generating flat Petri net model and conjunctions of FOLTL formulas for analysis purpose.
2) The graphical interface of SAMAT is developed using Java Swing API as it provides full GUI functionalities and mimics the platform it runs on. It consists of a SAM composition editor, a PIPE+ editor, a FOLTL editor and an analysis displayer. The composition editor is used for modeling the SAM compositions into a hierarchical structure; the PIPE+ editor is used for modeling the behaviors of SAM model via PrT nets; the FOLTL editor is used for defining the properties into FOLTL formulas; the analysis displayer is used for showing the analyzing result generated by SPIN [11].

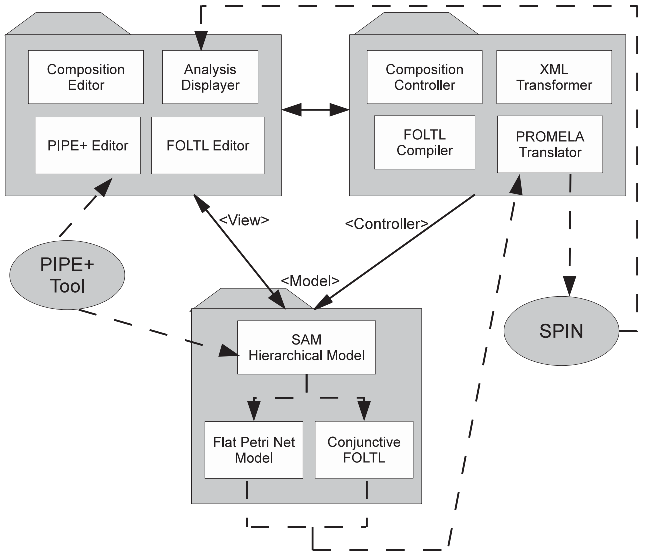


Figure 3. The Design View of SAMAT

3) The controller is comprised of composition controllers, a XML transformer and a PROMELA translator. The composition controllers provide options to specify detailed properties of a SAM composition; the XML transformer transforms SAMAT model into hierarchical XML format for storage purpose; the PROMELA translator translates the generated flat Petri net model and the conjunction of FOLTL formulas into PROMELA language, which is the input to SPIN.

SAMAT integrates two external tools: PIPE+ [14] for behavior modeling and SPIN [11] for model analysis.

C. SAM Hierarchical Model in SAMAT

SAMAT stores the SAM model in a hierarchical way. As we can see in Figure 4, the SAM model's data structure are in layers. In addition to the SAM compositions, the top layer contains a sub-composition model called sub-layer that has the same elements of the parent one except the bottom layer, which instead of a sub-composition model, is a Petri nets model. Therefore, each sub-composition model also has allocated space for its own sub-composition and a user can model arbitrarily number of levels by this recursive layer structure.

The Petri nets layer in the bottom of Figure 4 is the behavior model of its parent composition. In this case, it is a high-level Petri net formalism modeled in PIPE+ editor. Once a Petri net model is created, it is transformed and saved in XML format and is appended to its parent SAM composition.

In this way, SAMAT is capable of storing hierarchical layers of the SAM architecture model. SAMAT supports a top-down approach to develop a software architecture specification by decomposing a system specification into specification of components and connectors and by refining a higher level component into a set of related sub-components and connectors at a lower level. From the SAMAT's GUI, each component provides options for a user to define a sub layer or a behavior model. If the sub-layer is selected, a new tab of drawing canvas is built in the mainframe editor with designated title of "parent name :: sub composition name". Furthermore, if the sub composition can be further decomposed, another new tab will be built. If the behavior model option is selected, PIPE+ is triggered for the user to build a behavior model using Petri nets. Therefore, the top-down decomposition process is straightforward.

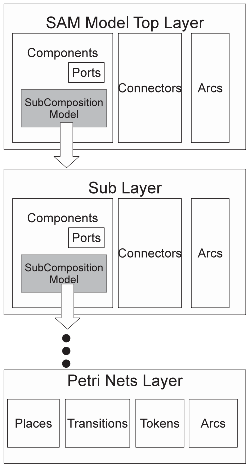


Figure 4. The SAM Hierarchical Model

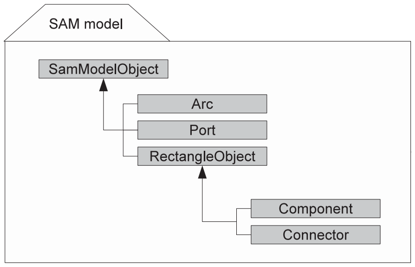


Figure 5. The Architecture of SAM Model Package

D. Inheritance Class Design in SAMAT

The design of the SAM model package in SAMAT must include all the SAM's graphical elements (i.e. components, connectors, arcs and ports). Figure 5 illustrates the class design hierarchy diagram. For the reusability and extensibility purpose, all of the SAM graphical elements are derived from SamModelObject class that holding basic features of a graphical object such as position, label and handler. Furthermore, Arc, Port and RectangleObject classes are inherited from SamModelObject, and Component and Connector classes are inherited from RectangleObject class.

E. FOLTL Editor

One of the underlying formalism in SAMAT is FOLTL. The vocabulary and models of FOLTL used in SAMAT are based on the high-level Petri net formalism and follow the approach defined in [13]. An example FOLTL formula is $\Box((x>y) \Rightarrow \Diamond (b=1))$, where variables are restricted to the underlying behavior models' arc variables. Since in each composition, SAMAT integrates a FOLTL formula editor where a user can specify system properties, the composition-level property specification is obtained by conjoining the property specifications of all components and connectors. The FOLTL compiler checks the syntax of a FOLTL formula and the translator generates constraint code in PROMELA.

F. PIPE+

The other formalism in SAMAT is PrT nets, which are a class of high-level Petri nets. We integrate an existing open source high-level Petri net tool PIPE+ [14] to specify the behavior model of the SAM architecture. PIPE+ is capable of specifying and simulating high-level Petri nets proposed in [2]. SAMAT leverages PIPE+'s editing mode in which a high-level Petri net behavior model can be developed graphically with dragging and dropping actions. The high-level Petri net model is comprised of:

1) A net graph consists of places, transitions and arcs.
2) Place types: These are non-empty sets restricting the data structure of tokens in the place.
3) Place markings: A collection of elements (tokens) associated with places. For analysis purpose, a bound of tokens' capacity on each place is necessary, so that verification run on SPIN can always stop.
4) Arc annotations: Arcs are inscribed with variables that contributes to the transition expression formula variables;
5) Transition conditions: A restricted first order logic formula Boolean expression is inscribed in a transition. It is called restricted because the grammar doe not permit free variables.

With all of the above high-level Petri net concepts specified, the behavior model is formally defined and can be verified by model checking engines.

G. XML Transformer

SAMAT transforms a SAM structure model into a XML model based on its hierarchical structure; and then appends the high-level Petri net XML model generated by PIPE+ to it. In this way, the SAM structural and behavior models are complete and are stored and loaded via XML saver and loader.

IV. Verifying SAM Specifications

To ensure the correctness of a software architecture specification in SAM, we have to check all the constraints are satisfied by the corresponding behavior models. To automate the verification process in SAMAT, we leverage an existing linear time temporal logic model checking tool SPIN [11].

A. SPIN and PROMELA

1) The SPIN Model Checker: SPIN [11] is a model checker for automatically analyzing finite state concurrent systems. It has been used to check logical design errors in distributed systems, such as operating systems, data communications protocols, switching systems, concurrent algorithms, railway signaling protocols, etc. A concurrent system is modeled in the PROMELA (Process or Protocol Meta Language) modeling language [11] and properties are defined as linear temporal logic formulas. SPIN can automatically examine all program behaviors to decide whether the PROMELA model satisfies the stated properties. In case a property is not satisfied, an error trace generated, which illustrates the sequence of executed statements from the initial state. Besides, SPIN works on-the-fly, which means that it avoids the need to preconstruct a global state graph or Kripke structure, as a prerequisite for the verification of system properties.

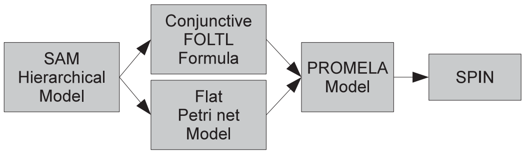


Figure 6. Verifying SAM Specifications

2) PROMELA: SPIN models in PROMELA consist of three types of objects: processes, message channels and variables. Processes specify the behavior, while the channels and variables define the environment for processes to run. The processes are global objects and can be created concurrently, which communicate via message passing through bounded buffered channels and via shared variables. Variables are typed, where a type can either be primitive or composite in the form of arrays and records.

B. From SAM Model to PROMELA Model

As shown in Figure 6, SAMAT starts by generating a flat (high-level) Petri net model from its hierarchical SAM model. Then, SAMAT automatically translates the flat Petri net model into a PROMELA model. Combined with FOLTL constraint formulas, SPIN can check the PROMELA model and output a verification result to SAMAT.

1) Generating An Integrated Flat Petri Net Model: Because a SAM model is hierarchically specified and each component in a different layer has its own behavior model, direct translation of a hierarchical SAM specification into PROMELA could result in a complex model not preserving the original semantics. Thus, SAMAT preprocesses the model by flattening the hierarchical structure.

In this phase, all the individual Petri net models created in different components of a SAM model need to be connected by directed arcs in both horizontal and vertical dimensions. Therefore, selecting interfaces among all the Petri net models are important. Because each Petri net model has input places (places without input arc, e.g. Sender in Figure 1) and output places (places without output arc, e.g. Receiver in Figure 1), which are used to communicate with other models, these input and output places are chosen as candidate interface places heuristically. Similarly, each SAM component has its input ports (P1 in Figure 1) and output ports (P4 in Figure 1) for the communication with other components, these input and output ports form the interface of the component.

The connection strategies are :

- **Horizontally**: Each SAM component has its input ports and output ports specified by one of the interface places

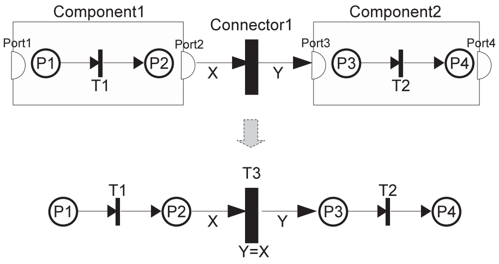


Figure 7. Generating Analysis Model by Horizontal Connection

High-level Petri net	PROMELA
Place	Channel
Place Type	Typedef Structure
Token	Message
Transition	Inline Function
Initial Marking	Message in Channel

Table I
MAP FROM HIGH-LEVEL PETRI NET TO PROMELA

of the underlying Petri net model (e.g. in Figure 7, Port 1 specified by P1 and Port 2 specified by P2). Integrating Petri net models from different components in the same hierarchical layer is by connecting the interface places. Moreover, the components in the same layer are connected by SAM connectors and arcs, so that SAMAT transforms them into Petri net transitions and arcs respectively. A new transition is created for each connector during the transformation (e.g. in Figure 7 is T3). The pre-condition of such transition is true by default; however a post-condition may be added. In the example, a post-condition “Y=X” is added. The variables in the new transition formula match the connector’s input and output arc variables. The sort of the variables is exactly the sort of the interface places, specified in ports, through connected arcs. Corresponding new arcs are added to reserve the flow relationships, which are connected with the interface places in ports and related transitions. For example, a new arc between place P2 in Port2 and T3 in Connector1.

- **Vertically**: The input or output ports not connected with any arcs in a component are mapped to the corresponding input and output ports in the parent component. For example in Figure1, ports P1 and P2 in top layer component A1 map to the second layer’s input port P5 and output port P8.

Thus, the Petri net models are connected and flattened into an integrated flat Petri net model that is ready to be translated into PROMELA model.

2) Translating Behavior Models to PROMELA: The translation process maps a high-level Petri net model to a PROMELA model. The resulting PROMELA model should catches the concept of high-level Petri nets defined in [2] and preserves the semantics. The PROMELA program’s major parts are definitions of places and place types, transition enabling and firing inline functions, a main process and an init process that defines the initial marking.

The translation map is shown in Table 1:

- **Translating places**: We predefined each message type (place type) into a new structure. Places and place types are mapped to PROMELA’s buffered channels and predefined message types. Besides, structured tokens are mapped to typed messages in PROMELA. Because SPIN verifies a model by exhaustive state searching, we set bounds to limit the number of tokens in places. The bounds are then mapped to the length of the channels. A sample PROMELA program resulted from place translation is shown below:

```
#define Bound_Place0 10
typedef type_Place0 {
        int field1;
        short field2
};
chan Place0 = [Bound_Place0]
        of {type_Place0};
```

- **Translating transitions**: In high-level Petri nets, a transition expression consists of a precondition and a postcondition. The precondition defines the enabling condition of the transition, and the postcondition defines the result of the transition firing. Each precondition and postcondition are translated into two inline functions, is_enabled() and fire(), respectively. fire() is triggered by the Boolean variable in is_enabled() evaluated to be true, otherwise it is skipped as the transition is not enabled. To check the precondition of a transition expression, we first consider a default condition that whether each of the input place has at least one token by checking the emptiness of each mapped channel. Because the expression is usually defined by conjunct clauses, we then evaluate each clause (the postcondition clauses are not evaluated this time). The evaluation process includes non-deterministically receiving a message from an input channel to a local variable, instantiating the Boolean expression, and evaluating it. A sample PROMELA program from transition translation is shown below:

```
inline is_enabled_Transition0{
}
inline fire_Transition0{...}
inline Transition0{
        is_enabled_Transition0();
        if
        :: Transition0_is_enabled
                -> fire_Transition0
        :: else -> skip
        fi
}
```

- **Defining main process**: The dynamic semantics of Petri nets is to non-deterministically check and fire enabled transitions, so the main process is defined by including all the transitions in a loop, “do ... od”. Since PROMELA has finer granularity that a transition firing process includes multiple sub-steps, we aggregate them into an atomic construct. A sample PROMELA program for an overall

PrT net structure is shown below:

```
proctype Main(){
          ...
          do
          :: atomic{Transition0}
          :: atomic{Transition1}
          od
}
```

- **Defining initial marking**: Since PROMELA has a special process "init{}", which is often used to prepare the true initial state of a system. Therefore, the initial marking is defined in init process by declaring typed messages and send them into buffered channels. A PROMELA prototype is shown below:

```
init{
          type_Place0 P0;
          P0.field1 = 1;
          P0.field2 = 0;
          Place0!P0;
          run Main()
}
```

- **Using basic data types**: Since the basic data types supported in PIPE+ are integer and string, which are mapped to "int" and "short" in PROMELA respectively.
- **Handling non-determinism**: In high-level Petri nets, tokens are meaningful data and usually different from each other and thus different firing orders result in different markings. Therefore, a non-deterministic inline function is defined and is called to non-deterministically pick a token from an input place each time a precondition is evaluated.
- **Supporting power set**: Because PIPE+ supports quantifiers in restricted first order logic formulas in transition expression, the domain of each quantified variable is a list of tokens as a power set contained in a place. For this kind of places, we are not dealing with one message but all the messages in the channel, we do not put all received messages into a local variable but directly manipulate the channel. The strategy is when the first message is received from the channel, it is used and then is sent back immediately.

3) FOLTL Formula: Since the FOLTL constraint formula extracted from the SAM composition model is conjoined into one integrated conjunctive FOLTL formula, the translation process is straightforward. We only need to wrap the formula by following PROMELA's syntax:

```
ltl f{/*formula*/}
```

4) Translation Correctness: The translation correctness is ensured by the following completeness and consistency criteria [18], [3]

Let N be a given Petri net and P_N be the resulting PROMELA program from the translation.

- **Completeness** Each element in N is mapped to P_N according to the mapping rules described in Section 4.2.2.
- **Consistency** The dynamic behavior of N is preserved by P_N as follows:
  - A marking of N defines the current state of N in terms of tokens in places, our place translating rule correctly maps each marking into a corresponding state in P_N;
  - The initial marking of N is correctly captured by the initial values of variables in the init{} process of P_N;
  - The enabling condition and firing result of each transition t in N is correctly inline functions "is_enabled_Transition_i" and "fire_Transition_i" respectively;
  - The atomicity of enabling and firing a transition in N is preserved in P_N by language feature "atomic{}".
  - An execution of N is firing sequence $\tau = M_0 t_0 M_1 t_1 ... M_n t_n ...$, where $M_i (i \in nat)$ is a marking and $t_i (i \in nat)$ denotes the firing of transition t_i. Each execution is correctly captured by the construct "do ... od" in the "Main" Promela function, which produces an equivalent execution sequence $\sigma = S_0 T_0 S_1 T_1 ... S_n T_n$, where $S_i (i \in nat)$ is a state and T_i denotes the execution of inline function "Transition_i".

The proofs of the completeness and consistency are straightforward and can be found in [18], [3] .

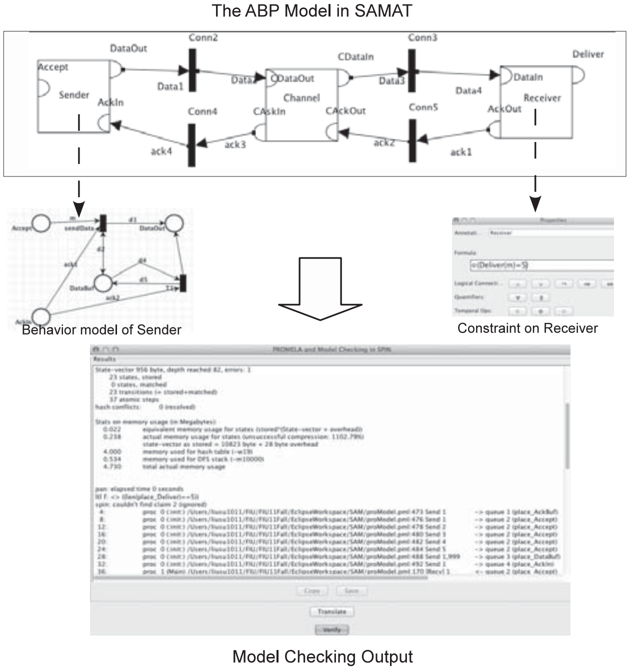
C. Verification using SPIN

The two inputs to SPIN are a PROMELA model and a property formula. SPIN performs verification by going through all reachable states produced by the model behaviors to check the property formula. If the property formula is unsatisfied, it produces a trail file indicates the error path. SPIN also provides a simulation function to replay the trail file so that any error path that leads to the design flaw can be visualized. SAMAT encapsulates the verification process in SPIN and displays the verification result as well as captured error path by SPIN in the GUI.

V. Using SAMAT

The alternating bit protocol (ABP) is a simple yet effective protocol for reliable transmission over lossy channels that may lose or corrupt, but not duplicate messages. It has been modeled and analyzed in [10]. In this section, we show some snapshots of using SAMAT in modeling and verifying this protocol.

In Figure 8, the top layer of ABP model in SAMAT consists of three components and four connectors. The first component "Sender" has a behavior model shown in Petri net. On the right, it shows the FOLTL editor to edit formula <>(Deliver(m) = 5). After the modeling process, SAMAT automatically generates PROMELA code as an input for SPIN and displays the model checking result after SPIN finished model checking. In this case an error is found, the replayed simulation on the error path is shown below the model checking result. The error indicates the ABP specification model in [10] is incorrect. A deadlock state (a none final state such that none of the transitions are enabled) can be reached when an acknowledgement message was corrupted in the channel and a resend message successfully reached the receiver's DataIn place. This discovery highlights the great benefits and usefulness of SAMAT.



```
State-vector 956 byte, depth reached 82, errors: 1
      23 states, stored
       0 states, matched
      23 transitions (= stored+matched)
      37 atomic steps
hash conflicts:         0 (resolved)

Stats on memory usage (in Megabytes):
   0.022        equivalent memory usage for states (stored*(State-vector + overhead))
   0.238        actual memory usage for states (unsuccessful compression: 1102.79%)
                state-vector as stored = 10823 byte + 28 byte overhead
   4.000        memory used for hash table (-w19)
   0.534        memory used for DFS stack (-m10000)
   4.730        total actual memory usage


pan: elapsed time 0 seconds
ltl f: <> ((len(place_Deliver)==5))
spin: couldn't find claim 2 (ignored)
  4:            proc  0 (:init:) /Users/liusu1011/FIU/FIU11Fall/EclipseWorkspace/SAM/proModel.pml:473 Send 1        -> queue 1 (place_AckBuf)
  8:            proc  0 (:init:) /Users/liusu1011/FIU/FIU11Fall/EclipseWorkspace/SAM/proModel.pml:476 Send 1        -> queue 2 (place_Accept)
 12:            proc  0 (:init:) /Users/liusu1011/FIU/FIU11Fall/EclipseWorkspace/SAM/proModel.pml:478 Send 2        -> queue 2 (place_Accept)
 16:            proc  0 (:init:) /Users/liusu1011/FIU/FIU11Fall/EclipseWorkspace/SAM/proModel.pml:480 Send 3        -> queue 2 (place_Accept)
 20:            proc  0 (:init:) /Users/liusu1011/FIU/FIU11Fall/EclipseWorkspace/SAM/proModel.pml:482 Send 4        -> queue 2 (place_Accept)
 24:            proc  0 (:init:) /Users/liusu1011/FIU/FIU11Fall/EclipseWorkspace/SAM/proModel.pml:484 Send 5        -> queue 2 (place_Accept)
 28:            proc  0 (:init:) /Users/liusu1011/FIU/FIU11Fall/EclipseWorkspace/SAM/proModel.pml:488 Send 1,999    -> queue 3 (place_DataBuf)
 32:            proc  0 (:init:) /Users/liusu1011/FIU/FIU11Fall/EclipseWorkspace/SAM/proModel.pml:492 Send 1        -> queue 4 (place_AckIn)
 36:            proc  1 (Main) /Users/liusu1011/FIU/FIU11Fall/EclipseWorkspace/SAM/proModel.pml:170 [Recv] 1       <- queue 2 (place_Accept)
```

Figure 8. Model the ABP in SAM Tool

Framework Name	Hierarchical Structure	Formalism	Tool	Verification Engine
Darwin/FSP	Yes	FSP and FLTL	Darwin	LTSA
Archware	Yes	Archware ADL and Archware AAL	ArchWare	CADP
CHARMY	No	State and Sequence Diagram and PSC	Charmy	SPIN
Auto FOCUS	Yes	Model-based	AF3	NuSMV and Cadence SMV
PoliS	No	PoliS and PoliS TL	N/A	PolisMC
Fujaba	Yes	UML and LTL/CTL	Fujaba	UPPAAL
SAM	*Yes*	*Petri Nets and FOLTL*	*SAMAT*	*SPIN*

Table II
SOFTWARE ARCHITECTURE FRAMEWORKS

VI. RELATED WORK

In the past decades, many software architecture modeling and analysis framework were proposed and their supporting tools were built. Several comparative studies[19], [4], [5] on these frameworks were published. Table 2 presents a comparison of several architecture modeling and analysis frameworks and their supporting tools.

Besides, CPN Tools [1] is a widely used Coloured Petri nets [12] tool that can also be used for modeling and analyzing software architecture. In terms of software architectural modeling, CPN Tools do not directly support architecture level concepts and features and thus the resulting models do not match user's abstraction well and can be difficult to understand. In terms of analysis, CPN Tools generates a full state space during verification and thus is often limited by the memory size while SPIN can perform verification on-the-fly to avoid full state space preconstruction so that it can handle complex behavior models.

VII. CONCLUSION

In this paper, we present a tool SAMAT for modeling and analyzing software architecture specifications in SAM. SAMAT leverages two existing tools, PIPE+ for building Petri net models and SPIN for analyzing system properties. SAMAT can be a valuable tool for complex concurrent and distributed system modeling and analysis. SAMAT is an open source tool and is available for sharing and continuous enhancements from worldwide research community.

Acknowledgments This work was partially supported by NSF grants HRD-0833093.

REFERENCES

[1] Cpn tools. http://cpntools.org.

[2] *High-level Petri Nets - Concepts, Definitions and Graphical Notation*, 2000.

[3] Gonzalo Argote-Garcia, Peter J. Clarke, Xudong He, Yujian Fu, and Leyuan Shi. A formal approach for translating a sam architecture to promela. In *SEKE*, pages 440–447, 2008.

[4] Paul Clements and Mary Shaw. The golden age of software architecture: A comprehensive survey. Technical report, 2006.

[5] L. Dobrica and E. Niemela. A survey on software architecture analysis methods. *Software Engineering, IEEE Transactions on*, 28(7):638 – 653, jul 2002.

[6] H.J. Genrich and K. Lautenbach. System modelling with high-level petri nets. *Theoretical Computer Science*, 13(1):109 – 135, 1981.

[7] Xudong He. A formal definition of hierarchical predicate transition nets. In *Application and Theory of Petri Nets*, pages 212–229, 1996.

[8] Xudong He and Yi Deng. Specifying software architectural connectors in sam. *International Journal of Software Engineering and Knowledge Engineering*, 10(4):411–431, 2000.

[9] Xudong He and Yi Deng. A framework for developing and analyzing software architecture specifications in sam. *Comput. J.*, 45(1):111–128, 2002.

[10] Xudong He, Huiqun Yu, Tianjun Shi, Junhua Ding, and Yi Deng. Formally analyzing software architectural specifications using sam. *Journal of Systems and Software*, 71:1–2, 2004.

[11] Gerard Holzmann. *Spin model checker, the: primer and reference manual*. Addison-Wesley Professional, first edition, 2003.

[12] Kurt Jensen, Lars Michael Kristensen, and Lisa Wells. Coloured petri nets and cpn tools for modelling and validation of concurrent systems. In *INTERNATIONAL JOURNAL ON SOFTWARE TOOLS FOR TECHNOLOGY TRANSFER*, page 2007, 2007.

[13] Leslie Lamport. The temporal logic of actions. *ACM Trans. Program. Lang. Syst.*, 16:872–923, May 1994.

[14] Su Liu, Reng Zeng, and Xudong He. Pipe+ - a modeling tool for high level petri nets. *International Conference on Software Engineering and Knowledge Engineering (SEKE11)*, pages 115–121, 2011.

[15] Zohar Manna and Amir Pnueli. *The temporal logic of reactive and concurrent systems*. Springer-Verlag New York, Inc., New York, NY, USA, 1992.

[16] Mary Shaw and Paul Clements. The golden age of software architecture. *IEEE Softw.*, 23:31–39, March 2006.

[17] Jiacun Wang, Xudong He, and Yi Deng. Introducing software architecture specification and analysis in sam through an example. *Information & Software Technology*, 41(7):451–467, 1999.

[18] Reng Zeng and Xudong He. Analyzing a formal specification of mondex using model checking. In *ICTAC*, pages 214–229, 2010.

[19] Pengcheng Zhang, Henry Muccini, and Bixin Li. A classification and comparison of model checking software architecture techniques. *Journal of Systems and Software*, 83(5):723 – 744, 2010.

Singular Formulas for Compound Siphons, Complementary Siphons and Characteristic Vectors for Deadlock Prevention in Cloud Computing

Gaiyun Liu[1], D.Y.Chao[2], Yao-Nan Lien[3]

***Abstract*—Unmarked siphons in a Petri net modeling concurrent systems such as those in cloud computing induce deadlocks. The number of siphons grows exponentially with the size of a net. This problem can be relieved by computing compound (or strongly dependent) siphons based on basic siphons. A basic (resp. compound) siphon can be synthesized from an elementary (resp. compound called alternating) resource circuit. It however cannot be extended to cases where two elementary circuits intersect at a directed path rather than a single place (i.e., corresponding to a weakly dependent siphon). This paper develops a uniform formula not only for both cases but also valid for the complementary set of siphon and characteristic vectors. We further propose to generalize it to a compound siphon consisting of n basic siphons. This helps simplify the computation and the computer implementation to shorten the program size. Also, the formula is easier to be memorized without consulting the references due to the same underlying physics.**

***Index Terms*—Petri nets, siphons, control, cloud computing, concurrent systems.**

I. Introduction

Lately, Service Oriented Architectures (SOA) and Cloud Computing are increasingly prominent. Deadlocks can easily occur [1], [2] when resources are inappropriately shared among large concurrent processes. The degree of concurrency explodes compared with multicore PCs. The number of concurrent processes competing for a limited amount of resources becomes huge and deadlocks can easily occur negating the advantages of fast speed and automation. Currently, there is no effective solution for this.

Li and Zhou [4] propose the concept of elementary siphons (generally much smaller than the set of all emptiable siphons in large Petri nets) to minimize the new addition of control places. They classify emptiable siphons into two kinds: elementary and dependent. By adding a control place for each elementary siphon S_e, all dependent siphons S too are controlled too, thus reducing the number of monitors required rendering the approach suitable for large Petri nets.

As a result, for complex systems, it is essential to apply the concept of elementary siphons to add monitors; the number of which is linear to the size of the nets modeling the systems. However, the number of dependent siphons is exponential to the size of the net, even though that of elementary siphons is linear.

We discover that different cases can be unified by the same physics resulting in a single formula to compute dependent siphons, their complementary set of places, and characteristic T-vectors. This helps to memorize the formula and simplify the implementation since the codes for different cases can be shortened with a single set of codes for the uniform formula. This relieves the problem of computing siphons (and related variables for controlling the siphons), the number of which grows exponentially with the size of a net.

II. PRELIMINARIES

Here only the definitions used in this paper are presented. The reader may refer to [3], [5] for more Petri net details.

Let $\Omega \subseteq P$ be a subset of places of *N. P–vector* λ_Ω is called the characteristic P-vector of Ω *iff* $\forall p \in \Omega, \lambda_\Omega(p)$ *=1;* otherwise $\lambda_\Omega(p)$ *=0.* η is called the characteristic T-vector of Ω, if $\eta^T = {\lambda_\Omega}^T \bullet [N]$, where [N] is the incidence matrix. Physically, the firing of a transition t where [$\eta(t)>0$, $\eta(t) =0$, and $\eta(t)<0$] increases, maintains and decreases the number of tokens in S, respectively. Let $\eta_{S_\alpha}, \eta_{S_\beta}, ...,$ and η_{S_γ} $(\{\alpha, \beta, ..., \gamma\} \subseteq \{1, 2, ..., k\})$ be a linear independent maximal set of matrix $[\eta]$. Then $\Pi_E = \{S_\alpha, S_\beta, ..., S_\gamma\}$ is called a set of elementary siphons. $S \notin \Pi_E$ is called a strongly dependent siphon if $\eta_S = \sum_{S_i \in \Pi_E} a_i \eta_{S_i}$ where $a_i \geq 0$. $S \notin \Pi_E$ is called a weakly dependent siphon if $\exists$non-empty A, $B \subset \Pi_E$, such that $A \cap B = \emptyset$ and $\eta_S = \sum_{S_i \in A} a_i \eta_{S_i} - \sum_{S_i \in B} a_i \eta_{S_i}$ where $a_i > 0$.

Definition 1: [5], [6] The strongly connected circuit from which an SMS can be synthesized is called a core circuit. An elementary resource circuit is called a basic circuit, denoted by c_b. The siphon constructed from c_b is called a basic siphon. A compound circuit $c = c_1 \circ c_2 \circ \ldots \circ c_{n-1} \circ c_n$ is a circuit consisting of multiply interconnected elementary circuits c_1, c_2, $\ldots$, c_n extending between Process 1 and 2. A transition t in a path $[r_i \; t \; r_j]$, r_i and r_j being in c, is called an $internal$ one if $\eta(t) = 0$. The SMS synthesized from compound circuit c using the Handle-Construction Procedure in [5] is called an n-compound siphon S, denoted by $S = S_1 \circ S_2 \circ \ldots \circ S_{n-1} \circ S_n$ (resp. $S = S_1 \oplus S_2 \oplus \ldots \oplus S_{n-1} \oplus S_n$). if $c_i \cap c_{i+1} = \{r_i\}$ (resp. $\{r_a, \ldots, r_b\}$), r_i, r_a, $\ldots$, $r_b \in P_R$ i.e., c_i and c_{i+1} intersects at a resource place r_i (resp. more than a resource place).

This work was supported by the National Science Council under Gant NSC 99-2221-E-004-002.

1. Gaiyun Liu is with the school of Electro-Mechanical Engineering, Xidian University, Xi'an, 710071 China.

2. D. Y. Chao is with the Department of Management and Information Systems, National ChengChi University, Taipei 116, Taiwan, Republic of China.

3. Yao-Nan Lien and Keng-Cheng Lin are with the Department of Computer Science, National ChengChi University, Taipei 116, Taiwan, Republic of China.

$(r_1r_2\ldots r_k)_1$ denotes Path $[r_1t_1r_2\ldots r_it_i\ldots r_{k-1}t_{k-1}r_k]_1$ for simple presentation.

t_2 in $[r_2\ t\ r_1]$ in Fig. 1 is an internal one of c_3 since $\eta_3(t_2) = 0$.

III. MOTIVATION

First we observe that (Table I) for a strongly dependent 2-compound siphon,

$$\varsigma_3 = \varsigma_1 + \varsigma_2 - \varsigma_{12} \tag{1}$$

where ς_{12} is the ς value for the siphon with $R(S) = R(S_1 \cap S_2)$, where $\varsigma = S$, $[S]$, and η, $R(S)$ is the set of resource places in S. Next extend this equation to a weakly dependent 2-compound siphon (Table II).

In Fig. 2, there are 3 elementary siphons S_1-S_3 and 1 weakly dependent siphon S_4; their characteristic T-vectors η are shown in Table I. Since the number of SMS grows exponentially with the size of a net, the time complexity of computing η for elementary siphons is exponential and quite time consuming.

IV. THEORY

In the sequel, we assume that all core circuits extend between two processes is an S^{3}PR. Eq. (1) for 2-compound siphon will be proved first based on Theorem 2 followed by the theory for n-compound (n>2) siphons.

We first deal with strongly 2-compound siphons based on the following lemma.

Lemma 1: Let $r \in P_R$, the minimal siphon containing r is $S = \rho(r) = \{r\} \cup H(r)$ (also the support of a minimal P-invariant) with $[S] = \emptyset$ and η_S =0.

Theorem 1: For every compound circuit made of $c_{b1}$and c_{b2}in an S^{3}PR corresponding to an SMS S_0 such that $c_{b1} \cap c_{b2}$= {r},

1) $\eta_0 = \eta_1 + \eta_2$, where $r \in P_R$ and η_0 is the η value for S_0

2) $\eta_0 = \eta_1 + \eta_2 - \eta_{12}$, where η_{12} is the characteristic T-vector of the minimal siphon containing r.

3) $[S_0] = [S_1] + [S_2] - [S_{12}]$.

4) S$_0 = S_1 + S_2 - S_{12}$.

5) ς = ς_1 + ς_2 - ς_{12}, where $\varsigma = S,\ [S],\ \eta$.

This theorem proves Eq. (1) for a strongly 2-compound siphon. We now deal with weakly 2-compound siphons. We shows that if S_0 weakly depends on S_1 and S_2, then there exists a third siphon S_3 — synthesized from core circuits formed by $c_1 \cap c_2$, respectively, such that $\eta_0 = \eta_1 + \eta_2 - \eta_3$. We [5] show that in an S^{3}PR, an SMS can be synthesized from a strongly connected resource subnet and any strongly dependent siphon corresponds to a compound circuit where the intersection between any two elementary circuits is at most a resource place.

Let S_0 be a strongly dependent siphon, S_1, $S_2\ldots$, and S_n be elementary siphons, with $\eta_{S0} = \eta_{S1} + \eta_{S2} + ... + \eta_{Sn}$. We show in [5] that c_0 (the core circuit from which to synthesize S_0) is a compound resource circuit containing c_1, $c_2\ldots, c_n$ and the intersection between any two c_i and c_j, *i =j-1>0*, is exactly a resource place, where c_i (*i =0,1,2, …, n*) is the core circuit from which to synthesize S_i. Thus, if S_0 is a WDS (weakly dependent siphon), the intersection between any two c_i and $c_j, i = j - 1 > 0$ must contain more than one resource place.

The following theorem from [7] shows that if S_0 weakly depends on S_1 and S_2, then $\eta_0 = \eta_1 + \eta_2 - \eta_3$.

Theorem 2: (Theorem 2 in [7]) For every compound circuit made of c_{b1} and c_{b2} in an S^{3}PR corresponding to an SMS $S*$, if $c_1 \cap c_2$ contains a resource path that contains transitions, and there is a minimal siphon S_{12} with $R(S_{12}) = R(S_1 \cap S_2)$. Then $\eta_0 = \eta_1 + \eta_2 - \eta_3$, where η_3 is the characteristic T-vector of S_{12}. (Let c_i be the core circuit for SMS S_i, $i = 1, 2$ and $c_1 \cap c_2 \neq \emptyset$. Then there is a third core circuit formed by parts of c_1 and c_2 (${c^a}_1$ and ${c^b}_2$, respectively; i.e., $c_3 = {c^a}_1 \cup {c^b}_2$).

The theorem definitely holds for the net in Fig. 2 as shown in Table II where $\Gamma = c_1 \cap c_2 = [p_{15}t_2p_{14}]$ is in Process 1. $\Gamma_1 = [p_{15}t_2p_{14}t_3p_{13}]_1$ plus $[p_{13}t_8p_{15}]_2$ forms basic circuit c_1, $\Gamma_2 = [p_{16}t_1p_{15}t_2p_{14}]_1$ plus $[p_{14}t_4p_{16}]_2$ form basic circuit c_2, $\Gamma_3 = [p_{15}t_2p_{14}]_1$ plus $[p_{14}t_6p_{15}]_2$ forms basic circuit c_3.

In general, the resource path of $c_1 \cup c_2$ in Process 1 can be expressed as $\Gamma_\alpha\Gamma\Gamma_\lambda$ (serial concatenation of Γ_α, Γ, and Γ_λ) where $\Gamma_\alpha\Gamma$ belongs to c_2, $\Gamma\Gamma_\lambda$ belongs to c_1, $\Gamma_\alpha = [r_1t_1r_2\ldots r_it_ir_j\ldots t_{k-1}r_k]_1$ (In Fig. 2, $\Gamma_\alpha = [p_{14}\ t_3\ p_{13}]_1$), $\Gamma_\lambda = [r_it_ir_j\ldots t_{k-1}r_kt_kr_{k+1}\ldots t_{m-1}r_m]_1$ (In Fig. 2, $\Gamma_\lambda = [p_{16}\ t_1\ p_{15}]_1$), and $\Gamma = [r_it_ir_j\ldots t_{k-1}r_k]_1$ (In Fig. 2, $\Gamma = c_1 \cap c_2 = [p_{15}t_2p_{14}]_1$). It remains true that $\eta_0(t) = \eta_1(t) + \eta_2(t) - \eta_3(t)$ for every transition in Processes 1 and 2 since each of $\Gamma_\alpha, \Gamma, \Gamma_\lambda$ is expanded by adding internal transitions t with $\eta(t) = 0$.

Define $S_{1,2} = S_3$ and $S_0 = S_1 \oplus S_2$ since $R(S_1 \cap S_2) = R(S_3)$. $S_1 \oplus S_2$ is similar to S_1oS_2 in terms of controllability to be shown later. $S_1 \oplus S_2$ is different than S_1oS_2 in that $R(S_1 \cap S_2)$ for the former contains more than one resource place while the latter contains only one resource place.

Definition 2: : Let (N$_0, M_0$) be a net system and $S_0 = S_1 \oplus S_2$ denotes the fact that S_0 is a weakly dependent SMS w.r.t. elementary siphons S_1, S_2, and $S_{1,2} = S_3$ such that $\eta_0 = \eta_1 + \eta_2 - \eta_3$.

We now propose the following:

Theorem 3: Let $S_0 = S_1 \oplus S_2$ as defined in Definition 1, then

1) $[S_0] = [S_1] \cup [S_2]$,

2) $[S_0] = [S_1] + [S_2] - [S_3]$,

3) $[S_1] \cap [S_2] = S_{1,2}]$,

4) $S_0 = S_1 + S_2 - [S_{12}$,

5) ς = ς_1 + ς_2 - ς_{12}, where $\varsigma = S,\ [S],\ \eta$, and

6) $M([S_0]) = M([S_1]) + M([S_2]) - M([S_{1,2}])$.

Consider the S^{3}PR in Fig. 3(a), $[S_0]$= $\{p_2, p_3, p_7, p_{8,} p_{9,} p_{10}\}$. Based on Table I, one can verify that 1) $[S_0]$= $[S_1]$ $\cup [S_2]$ and 2) $[S_1] \cap [S_2]$= $[S_{1,2}]$= S_3.

This theorem confirms the uniform computation for a weakly 2-compound siphon. The following theorem extends the uniform computation to a weakly n-compound siphon.

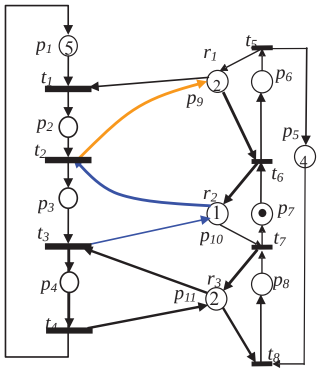
Theorem 4: Let S_0 = $S_1 \oplus S_2 \oplus \ldots \oplus S_n$ Then

1) η_0 = η_1 + η_2 + ... + η_n - $\eta_{1,2}$ - $\eta_{2,3}$ - …- $\eta_{n-1,n}$.

2) $[S_0]$ = $[S_1]$ + $[S_2]$ + ... + $[S_n]$ - $[S_{1,2}]$ - $[S_{2,3}]$ - …- $[S_{n-1,n}]$.

3) $S_0 = S_1 + S_2$ + ... + S_n - $S_{1,2}$-$S_{2,3}$ - …- $S_{n-1,n}$.

4) ς = $\varsigma_1 + \varsigma_2$ + ... + ς_n - $\varsigma_{1,2}$ - $\varsigma_{2,3}$ - …- $\varsigma_{n-1,n}$.



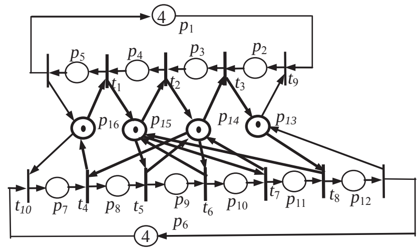


Fig. 2 Example weakly 2-compound siphon. $\eta_0 = \eta_1+\eta_2-\eta_3$.

Fig. 1. Example S^3PR with stronglydependent siphon. $\eta_3 = \eta_1 + \eta_2$.

Fig. 2. Example weakly 2-compoundsiphon. $\eta_0 = \eta_1 + \eta_2 - \eta_3$.

TABLE I
TYPES OF SIPHONS FOR THE NET IN FIG. 1

SMS	$[S]$	η	Set of places	c
S_1	$\{p_2, p_7\}$	$[-t_1 + t_2 + t_6 - t_7]$	$\{p_9, p_{10}, p_3, p_6\}$	$c_1 = [p_9 t_6 p_{10} t_2 p_9]$
S_2	$\{p_3, p_8\}$	$[$-$t_2 + t_3 + t_7 - t_8]$	$\{p_{10}, p_{11}, p_4, p_7\}$	$c_2 = [p_{10} t_7 p_{11} t_3 p_{10}]$
S_3	$\{p_2, p_3, p_7, p_8\}$	$[-t_1 + t_3 + t_6 - t_8]$	$\{p_9, p_{10}, p_6, p_{11}, p_4\}$	$c_3 = c_1 o c_2$

TABLE II
FOUR SMS IN FIG. 2 AND THEIR η. $\eta_4 = \eta_1 + \eta_2 - \eta_3$.

SMS	$[S]$	η	Set of places	c
S_1	$\{p_2, p_3, p_8, p_9, p_{10}, p_{11}\}$	$[+t_2 - t_4 + t_8 - t_9]$	$\{p_4, p_{12}, p_{13}, p_{14}, p_{15}\}$	$c_1 = [p_{15} t_2 p_{14} t_3 p_{13} t_8 p_{15}]$
S_2	$\{p_3, p_4, p_7, p_8, p_9, p_{10}\}$	$[+t_1 - t_3 + t_7 - t_{10}]$	$\{p_5, p_{11}, p_{14}, p_{15}, p_{16}\}$	$c_2 = [p_{14} t_4 p_{16} t_1 p_{15} t_2 p_{14}]$
S_3	$\{p_3, p_8, p_9, p_{10}\}$	$[+t_2 - t_3 - t_4 + t_7]$	$\{p_4, p_{11}, p_{14}, p_{15}\}$	$c_3 = [p_{15} t_2 p_{14} t_6 p_{15}]$
S_4	$\{p_2,\ p_3,\ p_4,\ p_7,\ p_8,\ p_9,\ p_{10},\ p_{11}\}$	$[+t_1 + t_8 - t_9 - t_{10}]$	$\{p_5, p_{12}, p_{13}, p_{14}, p_{15}, p_{16}\}$	$c_4 = c_1 \oplus c_2$

Thus, we prove the uniform formula for weakly n-compound siphons. It also holds for strongly dependent siphons as shown earlier. Thus, the uniform formula holds irrespective to whether the compound siphon is strongly or weakly. This further enhances the uniformity of the formula. A more complicated example in the next section illustrates this for a weakly 3-compound siphon.

V. EXAMPLE

This section employs Fig.3 to demonstrate Theorem 4. As shown in Tables III-VII, S, $[S]$ and η for weakly compound siphons all share the same formula verifying Theorem 4. For strongly dependent siphons, the same formula also holds except $S_{ij} = [S_{ij}] = \emptyset$, η_{ij} = 0, $\forall i \neq j$.

VI. CONCLUSION

In summary, we propose a uniform formula to compute SMS, their complementary set and characteristic T-vectors for both strongly and weakly n-compound siphons based on the same underlying physics. We further propose to generalize it to a compound siphon consisting of n basic siphons. This helps to retain the formula in brain without consulting the references and simplify the computation plus its implementation to reduce the lines of codes. Future work can be directed to large S^3PR and more complicated systems.

REFERENCES

[1] Wang, Y., Kelly, T., Kudlur, M., Lafortune, S., and Mahlke, S.A., “Gadara: Dynamic deadlock avoidance for multithreaded programsc In USENIX Symposium on Operating Systems Design and Implementation, 2008.

[2] Wang, Y., Kelly, T., Kudlur, M., Mahlke, S., and Lafortune, S., vThe application of supervisory control to deadlock avoidance in concurrent software “ In International Workshop on Discrete Event Systems, 2008.

[3] T. Murata, “Petri nets: properties, analysis and application,” in *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541–580, 1989.

[4] Li, Z. W. and M. C. Zhou., ”Elementary Siphons of Petri Nets and Their Application to Deadlock Prevention in Flexible Manufacturing Systems,” *IEEE Trans. Syst. Man Cybern. A.*, 34(1), 38-51, 2004.

[5] D.Y. Chao, “Computation of elementary siphons in Petri nets for deadlock control,” Comp. J., (British Computer Society), vol. 49, no. 4, pp. 470–479, 2006.

[6] D. Y. Chao, ”Improved controllability test for dependent siphons in S^3PR based on elementary siphons,” Asian Journal of Control, vol. 12, no. 3, pp. 377 – 391, doi:10.1002/asjc.217, 2010.

[7] D. Y. Chao, Jiun-Ting Chen, Mike Y.J. Lee, and Kuo-Chiang Wu, ”Controllability of Strongly and Weakly Dependent Siphons under Disturbanceless Control,” Intelligent Control and Automation, vol.2, no.4, pp. 310-319, 2011.

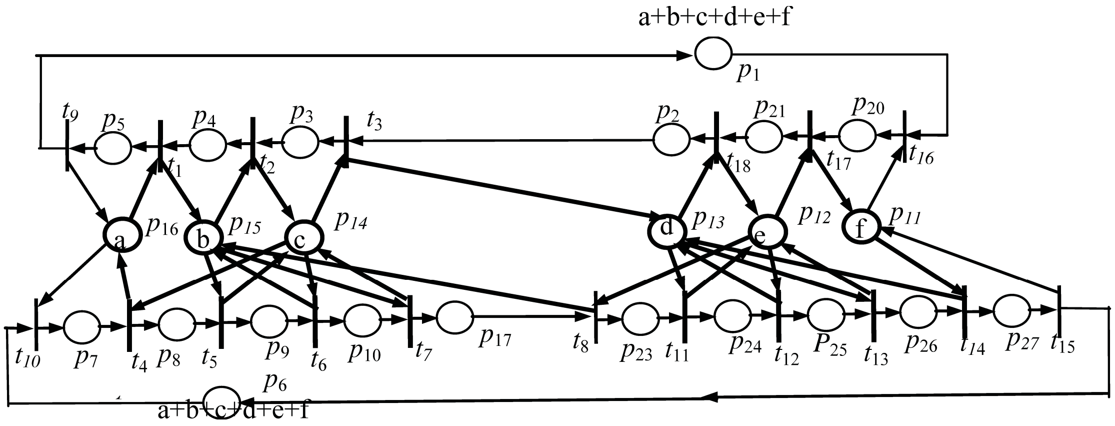


Fig. 3. Example a 3-compound weakly dependent siphon. S_0=S_6=S_1 $\oplus S_2 \oplus S_3$, S_4=$S_{1,2}$, S_5=$S_{2,3}$ and η_0=η_1+η_2+η_3-η_4-η_5.

TABLE III
EIGHT SMS S, AND CORE CIRCUITS IN FIG. 3.

S	Set of places	c
S_1	p_5, p_{17}, p_{14}, p_{15}, p_{16}	c_1=[p_{14} t_4 p_{16} t_1 p_{15} t_2 p_{14}]
S_2	p_4, p_{26}, p_{12}, p_{13}, p_{14}, p_{15}	c_2=[p_{15} t_2 p_{14} t_3 p_{13} t_{18} p_{12} t_8 p_{15}]
S_3	p_2, p_{27}, p_{11}, p_{12}, p_{13}	c_3=[p_{13} t_{18} p_{12} t_{17} p_{17} t_{14} p_{13}]
S_4	p_4, p_{17}, p_{14}, p_{15}	c_4=[p_{15} t_2 p_{14} t_6 p_{15}]].
S_5	p_2, p_{26}, p_{12}, p_{13}	c_5=[p_{13} t_{18} p_{12} t_{12} p_{13}]
S_6	p_5, p_{27}, p_{11}, p_{12}, p_{13}, p_{14}, p_{15}, p_{16}	c_6=$c_1 \oplus c_2 \oplus c_3$
S_7	p_5, p_{26}, p_{12}, p_{13}, p_{14}, p_{15}, p_{16}	$c_7 = c_1 \oplus c_2$
S_8	p_4, p_{27}, p_{11}, p_{12}, p_{13}, p_{14}, p_{15}	$c_8 = c_2 \oplus c_3$

TABLE IV
EIGHT SMS S,[S] AND η IN FIG. 3.

S	Set of places	[S]	η
S_1	p_5, p_{17}, p_{14}, p_{15}, p_{16}	p_3, p_4, p_7, p_8, p_9, p_{10}	t_1-t_3+t_7-t_{10}
S_2	p_4, p_{26}, p_{12}, p_{13}, p_{14}, p_{15}	p_2, p_3, p_8, p_9, p_{10}, p_{17}, p_{21}, p_{23}, p_{24}, p_{25}	t_2-t_4+t_{13}-t_{17}
S_3	p_2, p_{27}, p_{11}, p_{12}, p_{13}	p_{20}, p_{21}, p_{23}, p_{24}, p_{25}, p_{26}	-t_8+t_{14}-t_{16}+t_{18}
S_4	p_4, p_{17}, p_{14}, p_{15}	p_3, p_8, p_9, p_{10}	t_2-t_3-t_4+t_7
S_5	p_2, p_{26}, p_{12}, p_{13}	p_{21}, p_{23}, p_{24}, p_{25}	-t_8+t_{13}-t_{17}+t_{18}
S_6	p_5, p_{27}, p_{11}, p_{12}, p_{13}, p_{14}, p_{15}, p_{16}	p_2, p_3, p_4, p_7, p_8, p_9, p_{10}, p_{17}, p_{20}, p_{21}, p_{23}, p_{24}, p_{25}, p_{26}	t_1-t_{10}+t_{14}-t_{16}
S_7	p_5, p_{26}, p_{12}, p_{13}, p_{14}, p_{15}, p_{16}	p_2, p_3, p_4, p_7, p_8, p_9, p_{10}, p_{17}, p_{21}, p_{23}, p_{24}, p_{25}	t_1-t_{10}+t_{13}-t_{17}
S_8	p_4, p_{27}, p_{11}, p_{12}, p_{13}, p_{14}, p_{15}	p_2, p_3, p_8, p_9, p_{10}, p_{17}, p_{20}, p_{21}, p_{23}, p_{24}, p_{25}, p_{26}	t_2-t_4+t_{14}-t_{16}

Model-Based Metamorphic Testing: A Case Study

Junhua Ding
Department of Computer Science
East Carolina University
Greenville, NC, USA
dingj@ecu.edu

Dianxiang Xu
National Center for the Protection of the Financial
Infrastructure, Dakota State University
Madison, SD, USA 57042
dianxiang.xu@dsu.edu

Abstract—**Metamorphic testing has been successfully used for testing "non-testable" systems, where tests oracles are difficult to define. However, rigorous test generation for metamorphic testing remains largely open. In this paper, we propose a model-based approach to metamorphic testing. The "non-testable" system under test is modeled by a high-level Petri net. The Petri net is then used to automatically generate the initial tests that are adequate for a given test coverage criterion. Due to the absence of test oracles in "non-testable" systems, the correct outputs of the initial tests are unknown. Therefore, additional metamorphic related tests are created by applying metamorphic relations to the initial tests. The tests generated from the formal model of the system under test can substantially improve the quality of the metamorphic testing. This paper demonstrates the effectiveness of this approach through a "non-testable" image processing program.**

Keywords- test generation; Petri nets; metamorphic testing; test coverage criterion

I. Introduction

A "non-testable" system cannot be tested using traditional testing techniques due to the absence of test oracles [5]. Metamorphic testing [2] is a novel testing technique for testing "non-testable" systems [15]. It checks the satisfaction of Metamorphic Relationship (MR) among two or more outputs of the system under test instead of the correctness of individual output [2]. Metamorphic testing has been used for testing such software as numerical computing systems and machine learning systems [2][8].

In metamorphic testing, tests are created based on MRs and the predictable relations among test outputs are verified accordingly. In a metamorphic testing, if an initial test input x to a system under test generates an output $f(x)$, then a MR is used to create a transformation function t, which is applied to the test input x to produce another test input $t(x, f(x))$. The transformation allows us to predict the relationship between the output $f(x)$ and the output $f(t(x, f(x)))$ according to the MR [16]. If two related test outputs do not satisfy the MR, which means the relation between $f(x)$ and $f(t(x, f(x)))$ is inconsistent with the MR, then the system under test must have some defects. MRs are considered as test oracles in metamorphic testing to resolve the absence of test oracles in "non-testable" systems. The general process of metamorphic testing is as follows:

- Identify MRs. MRs will be used to generate additional test inputs based on the existing ones and to verify consistency among the test outputs [4].
- Create metamorphic test cases. Based on the identified MRs, additional test inputs are generated through transforming existing test inputs.
- Execute and evaluate tests. Each test case is executed separately, and its outputs are compared to the outputs of other MR related test cases. Obviously, the quality of metamorphic testing relies on the identification of MRs and test generation. Although some work have been done on the identification of MRs [10][13], test generation for metamorphic testing is still focused on domain-specific applications. A general approach to rigorous test generation for metamorphic testing remains to be seen.

In this paper, we propose a model-based approach to metamorphic testing. We first build a high-level Petri net model for the system under test, and then generate model-level tests and code-level tests using the test generation tool MISTA [18]. The test input data and the generated tests are served as the initial test input data and tests. Then, we apply MRs to these initial tests to generate MR related test input data and tests. Finally, we execute these MR related tests and verify the satisfaction of each MR on corresponding outputs.

The main contribution of this paper is due to a rigorous model-based approach for metamorphic testing. The most challenging problem of metamorphic testing is how to generate high quality tests, especially the initial tests, to adequately test the system. After the initial test input data and tests are available, other test cases can be created by applying MRs to the initial test input data and tests. In our approach, the initial tests generated from the model adequately cover the selected test coverage criterion. We have performed a case study on testing a real world image-processing program. The result has shown that the model-based approach to test generation can improve the effectiveness of metamorphic testing.

The rest of this paper is organized as follows: Section 2 presents a brief introduction to test generation with MISTA, a tool for test generation with high-level Petri nets. Section 3 presents model-based test generation for metamorphic testing through a case study. Section 4 reviews the related work. Section 5 concludes this paper.

II. Model-Based Testing with MISTA

A. Function Nets

Function nets are the modeling notation in the test generation tool MISTA (Model-based Integration and System

Test Automation). The beta release of MISTA is available at http://www.homepages.dsu.edu/dxu/research/MBT.html.

Function nets are a lightweight version of colored Petri nets or Predicate/Transition (PrT) nets, where all weights in each formal sum of tokens or arc labels are 1. The formal definition of function nets can be found in [19][20]. The structure of a function net can be represented by a set of transitions, where each transition is a quadruple <*event, precondition, postcondition, guard*>. The precondition, postcondition, and guard are first-order logic formulas. The precondition and postcondition are corresponding to the input and output places of the transition, respectively. Figure 1 shows the function net that models the 5 dining philosophers' problem. The functions to be tested include *pickup*, and *putdown*. Places *Phi* and *Chop* include tokens that are nature numbers representing philosophers or chopsticks, and each token in place *Down* consists of a philosopher and his/her two chopsticks. Function nets supports inhibitor arcs and reset arcs. The labels and guard conditions for each arc or transition are specified in Figure 1. An inhibitor arc represents a negative precondition of the associated transition. A reset arc represents a reset postcondition of the associated transition – it removes all tokens from the associated place after the transition is fired.

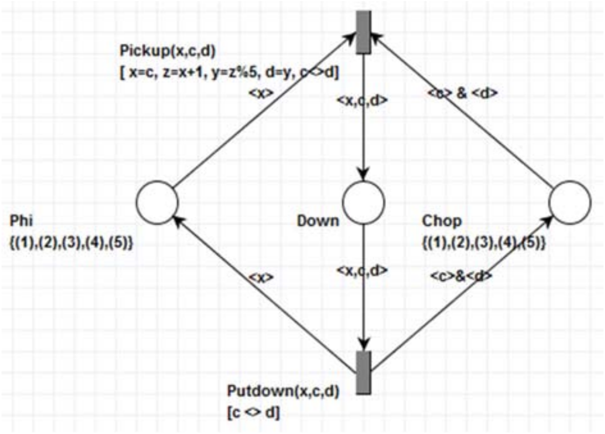
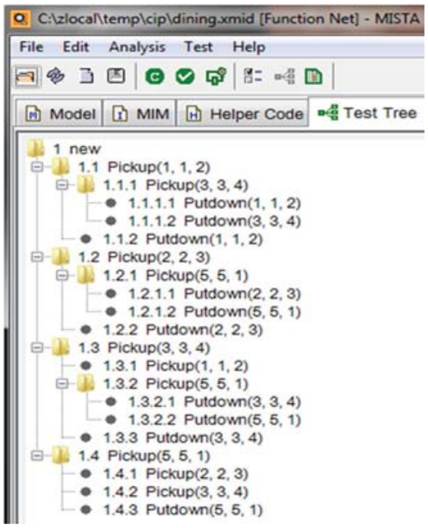


Figure 1. A function nets model for dining philosophers

B. MISTA

MISTA is a tool for automated generation of executable test code. It uses function nets for specifying test models so that complete tests can be generated automatically. It also provides a language for mapping the elements in function nets to implementation constructs. This makes it possible to convert the model-based tests into code that can be executed against the system under test. MISTA includes several important components: model editor, model parser, reachability analyzer, model checker, test generator, and test code generator. Test generator generates model-level tests (i.e., firing sequences) according to a chosen coverage criterion. The tests are organized and visualized as a transition tree. MISTA supports a number of coverage criteria for test generation from function nets, including reachability graph coverage, transition coverage, state coverage, depth coverage, and goal coverage. Figure 2 shows a test tree generated for the dining philosopher model shown in Figure 1. Test code generator generates test code in the chosen target language from a given transition tree. MISTA supports languages/test frameworks including Java/JUnit, C#/NUnit, C/C++, and many others [19].



```
1 new
  1.1 Pickup(1, 1, 2)
    1.1.1 Pickup(3, 3, 4)
      1.1.1.1 Putdown(1, 1, 2)
      1.1.1.2 Putdown(3, 3, 4)
    1.1.2 Putdown(1, 1, 2)
  1.2 Pickup(2, 2, 3)
    1.2.1 Pickup(5, 5, 1)
      1.2.1.1 Putdown(2, 2, 3)
      1.2.1.2 Putdown(5, 5, 1)
    1.2.2 Putdown(2, 2, 3)
  1.3 Pickup(3, 3, 4)
    1.3.1 Pickup(1, 1, 2)
    1.3.2 Pickup(5, 5, 1)
      1.3.2.1 Putdown(3, 3, 4)
      1.3.2.2 Putdown(5, 5, 1)
    1.3.3 Putdown(3, 3, 4)
  1.4 Pickup(5, 5, 1)
    1.4.1 Pickup(2, 2, 3)
    1.4.2 Pickup(3, 3, 4)
    1.4.3 Putdown(5, 5, 1)
```

Figure 2. A generated test tree

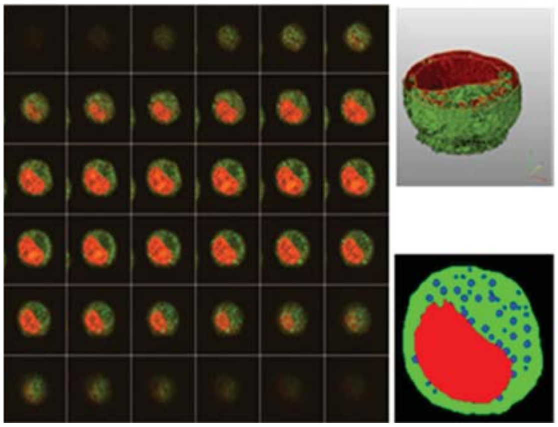


Figure 3. The left panel is a set of confocal images of a cell. The top right image is the sectional view of the reconstructed 3D structures of the cell, and the right bottom one is a processed image of the cell.

III. Model-Based Test Generation for Metamorphic Testing

In order to illustrate the basic idea and the process of the model-based test generation in metamorphic testing, we study a real-world "non-testable" program – an image processing program for reconstructing 3D structure of biology cells. The input to the program is a stack of confocal images of a cell, and each image represents a slice of the cell. Each image is first processed to recognize the nucleus, cytoplasm, and mitochondria, and then the 3D structure of the cell is reconstructed based on the stack of processed images. The number of mitochondria, the volume of the cell, the volume of nucleus, the volume of cytoplasm and the volume of total mitochondria are calculated based on the reconstructed 3D structure of the cell. We only discuss the pattern recognition component that is used for recognizing mitochondria, nuclear and cytoplasm in each slice of image. Figure 3 shows a sample input and output of the program.

A. The Approach

In our approach, first we create a test model for the system under test using function nets. Then we perform several rounds of analysis to ensure the correctness of the model. The analysis functions in MISTA include simulation, reachability analysis, deadlock checking and verification of goals. The simulation is used to execute the model to check its running at different scenarios. The reachability analysis is used to check the reachability of each transition or goal states defined in the model. As soon as the analysis is successfully passed, we need to choose an initial input (the initial marking of the function nets model) and a test coverage criterion to be used for generate tests. The tests generated are adequate to the selected test coverage criterion. In order to generate test code for testing the system under test, the model-implementation mapping for mapping tokens and transitions in the model to data and methods (and objects if the implementation is in an object oriented language) in the implementation has to be defined in MISTA. Additional helper code such as import statements has to be provided to make the generated test code executable. Due to the absence of test oracles in "non-testable" systems, we cannot decide the failure or success of the generated tests alone. Therefore, we need to identify a set of MRs in the system, and then create MR related tests through applying the MRs to the initial generated tests. Finally, the MR related tests are executed in parallel and their results are compared to decide the success or failure of the MR related tests altogether. As shown in Figure 4, the process of the model-based test generation for metamorphic testing comprises an iterative sequence of the activities consisted of design MRs, generate tests for the model, generate test code, create MR related tests, and conduct test.

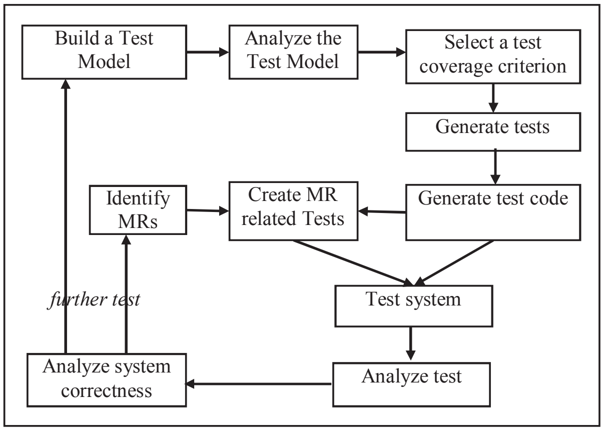


Figure 4. Process of model-based test generation

B. Designing MRs

Testing the image processing program is complex because each cell has many mitochondria in different shape or size. We cannot find a test oracle to tell whether the shape and size of each recognized mitochondrion is same as the real one in the cell. Guided by experience discussed in [11][12][16], we identify a set of MRs:

- Inclusive. If an artificial mitochondrion is added to the original image, then the added one should be recognized, the number of mitochondria in the cell should increase one, and the size of mitochondria shall increase correspondingly.
- Exclusive. If a mitochondrion is removed from the original image, then the removed one shall not exist in the processed image, the number of mitochondria in the cell should decrease one, and the size of the mitochondria shall decrease correspondingly.
- Multiplicative. If the size of a mitochondrion is enlarged with a small percentage from the original image, then the size of the mitochondrion shall be increased in the processed image, the number of the mitochondria in the cell shall keep the same, and the size of mitochondria is expected to increase.
- Shapes. Mitochondria may have different shapes. Artificial mitochondria with different shapes are added to the image to check whether these mitochondria can be recognized as expected, and the processing of other original mitochondria should not be affected.
- Positions. The program processes the mitochondria that are close to the nuclear differently to ones that are close to the cell boundary. Artificial mitochondria are added to the images in different positions. The new added mitochondria should be recognized, and the processing of other mitochondria should not be affected. The number and size of mitochondria in the cell shall keep the same if only the position of a mitochondrion is changed.

C. Building a Test Model

We built a test model for partial features of the cellular image processing program using the function nets. The test model defines the basic functions for recognizing mitochondria and the nuclear from a confocal image of a cell. The input of the program is a confocal image file that has been taken for a slice of a biology cell and a configuration file for setting the image processing. As soon as the files are read into the program, then setting information in the configuration file is ready for using, and the image file is separated two files based on the color of each cell component (e.g. nuclear, mitochondria, and cytoplasm). The blue one represents mitochondria, and the red one represents nuclear. The basic functions for processing mitochondria and nuclear are same including functions such as finding cutoff values for differentiating cell components from the cytoplasm, smoothing the contour line of each recognized cell component, removing noise in the processed image, filling/keeping the hole in each cell component. Each transition defined in the model represents at least one method in the program. Because the function nets model is very large, here we divided it into two pieces, which should be connected via the places/transitions that have the same names (the places/transitions that have the same name are the same place/transitions in the original model). Figure 5 and Figure 6 shows the test model for the image processing program.

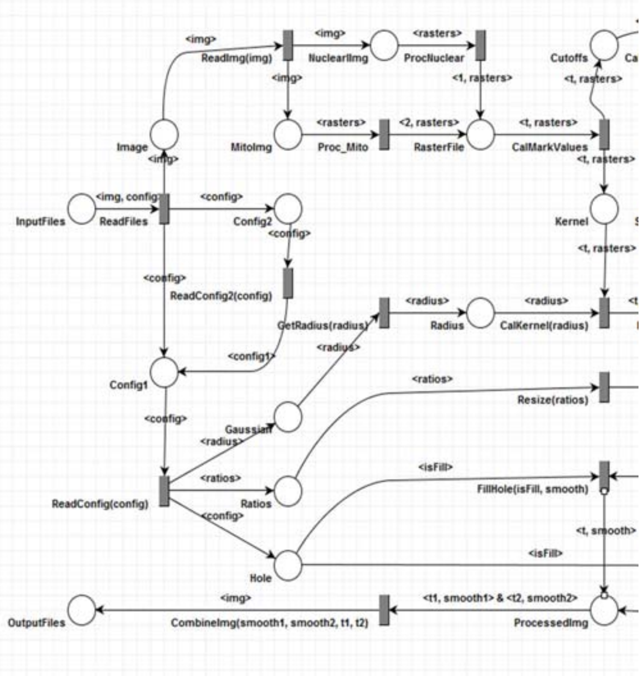


Figure 5. Test model of the image processing program (part 1)

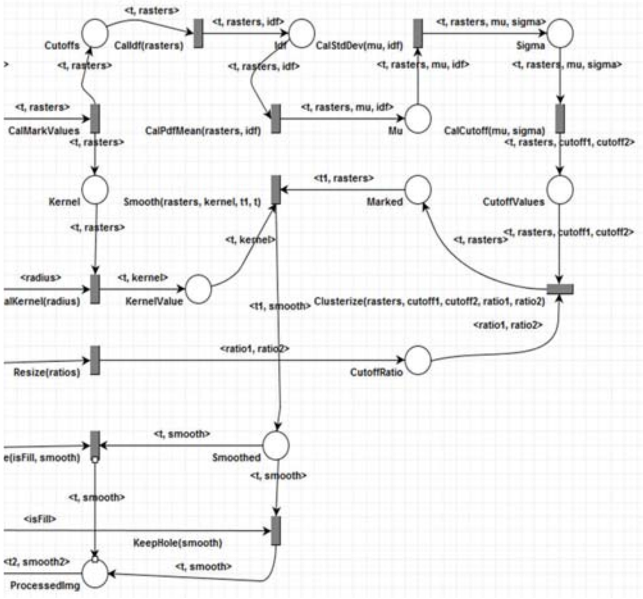
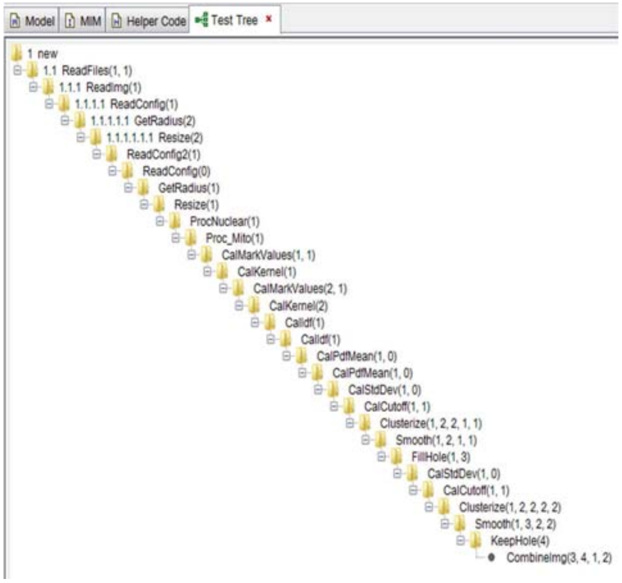


Figure 6. Test model of the image processing program (part 2)

Figured 5 and 6 can be combined using the common places and transitions, and Figure 5 is in the left side, and Figure 6 is in the right side. The guard conditions of each transition are not shown in Figures 5 and 6, but they were defined in the model. We setup the *INIT* input as <1, 1> representing one image file and one configuration file, and the *GOAL* as *tokenCount* (*OutputFiles, 1*) representing an output of a processed file. Then we performed simulation, reachability analysis and other analysis to improve our confidence of the correctness of the test model. Based on the analysis results, we knew that the GOAL state is reachable, and all transitions are reachable, and the running of the model stops normally when the processed file is output.



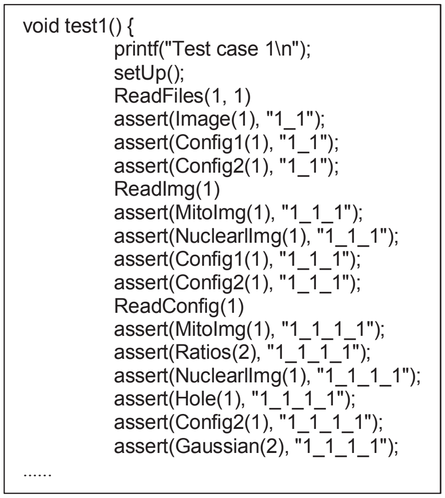
```
1 new
⊟ 1.1 ReadFiles(1, 1)
  ⊟ 1.1.1 ReadImg(1)
    ⊟ 1.1.1.1 ReadConfig(1)
      ⊟ 1.1.1.1.1 GetRadius(2)
        ⊟ 1.1.1.1.1.1 Resize(2)
          ⊟ ReadConfig2(1)
            ⊟ ReadConfig(0)
              ⊟ GetRadius(1)
                ⊟ Resize(1)
                  ⊟ ProcNuclear(1)
                    ⊟ Proc_Mito(1)
                      ⊟ CalMarkValues(1, 1)
                        ⊟ CalKernel(1)
                          ⊟ CalMarkValues(2, 1)
                            ⊟ CalKernel(2)
                              ⊟ CalIdf(1)
                                ⊟ CalIdf(1)
                                  ⊟ CalPdfMean(1, 0)
                                    ⊟ CalPdfMean(1, 0)
                                      ⊟ CalStdDev(1, 0)
                                        ⊟ CalCutoff(1, 1)
                                          ⊟ Clusterize(1, 2, 2, 1, 1)
                                            ⊟ Smooth(1, 2, 1, 1)
                                              ⊟ FillHole(1, 3)
                                                ⊟ CalStdDev(1, 0)
                                                  ⊟ CalCutoff(1, 1)
                                                    ⊟ Clusterize(1, 2, 2, 2, 2)
                                                      ⊟ Smooth(1, 3, 2, 2)
                                                        ⊟ KeepHole(4)
                                                          └● CombineImg(3, 4, 1, 2)
```

Figure 7. Test tree for covering all transitions

D. Generating Tests

As soon as the test model for the system under test is constructed, tests for the model can be automatically generated against a selected test coverage criterion. For example, we can choose the reachability tree coverage as the test coverage criterion, and then the tests generated shall cover all reachable paths in the model. Tests cannot cover all reachable paths in a program in general; so that the model-based test generation provides a rigorous way to generate tests that cover the paths (i.e. the paths defined in the model) that best represent the paths in the program. Since structurally complex data are not defined in the model, we don't apply any MR to the model-level tests directly. Instead, the MRs will be applied to the code-level tests. The code-level tests are translated from the model-level tests in MISTA, and they are the initial tests for testing the system. We do not know the correct outputs of these initial tests in "non-testable" system, but we know the relationship between each test and its MR related tests generated based on a MR. Since the initial tests generated from the model adequately cover the selected test coverage criterion, the tests that include initial tests and their MR related tests shall adequately cover the test coverage criterion as well. MR related tests are executed in parallel to compare their outputs according to the MR in order to decide whether the tests pass or fail.

The test input for the test model of the image processing program includes two files: one is a confocal image file, and another one is a configuration file. Each file is simply represented as a natural number token such as {<1, 1>}. Figure 7 shows the test tree for covering all transitions in the model. Only one test is needed for covering all transitions (i.e. the test is adequate for all transitions in the model). However, more than 1300 tests are needed for covering all state in the model (i.e. the test set that includes the 1300 tests is adequate for all states in the model). In addition, MISTA supports the generation of adequate test sets for covering all paths, all flows or all places or others in the model.



```
void test1() {
            printf("Test case 1\n");
            setUp();
            ReadFiles(1, 1)
            assert(Image(1), "1_1");
            assert(Config1(1), "1_1");
            assert(Config2(1), "1_1");
            ReadImg(1)
            assert(MitoImg(1), "1_1_1");
            assert(NuclearlImg(1), "1_1_1");
            assert(Config1(1), "1_1_1");
            assert(Config2(1), "1_1_1");
            ReadConfig(1)
            assert(MitoImg(1), "1_1_1_1");
            assert(Ratios(2), "1_1_1_1");
            assert(NuclearlImg(1), "1_1_1_1");
            assert(Hole(1), "1_1_1_1");
            assert(Config2(1), "1_1_1_1");
            assert(Gaussian(2), "1_1_1_1");
......
```

Figure 8. Test code for covering all transitions

The code-level test input for the image processing program includes a group of real confocal image files (see Figure 3) and a real configuration file {<Img_i, *Config*>}, where Img_i is the ith confocal image of the cell, and *Config* is the configuration file. The mapping between the test input {<1, 1>} for the model and the test input {<Img_i, *Config*>} for the program is defined before the test code is generated. Additional tests are created through applying each MR to existing tests. For example, we can apply MR *inclusive* to test input {<Img_i, *Config*>} and its corresponding tests to create a new test input {<Img'_i, *Config*>} and new tests, where Img'_i is the modified Img_i that includes an artificial mitochondrion. The corresponding tests are not changed except the test input Img_i is replaced by Img'_i. Using the same idea, we can create tests based on MR *exclusive*, *multiplicative*, *shapes* and *positions*. The outputs of MR related tests are compared based on the MR. Figure 8 shows partial of the test code for covering all transitions. The test code can be translated into a test for testing the program when the natural numbers are represented by real files, and the transitions are represented by functions in the program. However, the MR related tests have to be manually created based on the existing tests, which is fairly easy in this case since the only change is the image files.

IV. RELATED WORK

Metamorphic testing has been used for testing "non-testable" systems such as bioinformatics systems [3][6], machine learning systems [10][16], and online service systems [1]. Same to other software testing techniques, test generation is still one of the most important tasks in metamorphic testing. Software testing mainly has four types of test generation methods: program-based, specification-based, model-based and random test [14]. Program-based test generation falls into two mainly categories: static methods and dynamic methods. The static methods generate test cases through analyzing the program code without running the program. The dynamic methods produce test cases using heuristic analysis of dynamic behaviors during the program's execution. Due to absence of test oracles in "non-testable" programs, program-based test generation cannot be used to generate practical test cases for directly testing "non-testable" systems. Specification-based test generation generates high level test cases from formal specifications like Z or Petri nets of the program under test, and then the high level test cases are transformed into test inputs to test the program. Model based test generation derives model level test cases from program models such as Finite State Machine models or UML models, and then the model level test cases are transformed into test inputs [17]. However, generating structurally complex test input data like the biomedical images remains a challenge issue in specification-based or model-based test generation [14]. In this paper, high-level tests are automatically generated from the model using tool MISTA, and then these high-level tests are translated into code-level tests automatically. These generated tests server as the initial test inputs and tests, and then MRs are applied to them to generate additional test inputs and tests. Radom test generation selects an arbitrary subset of all possible input values over the input space according to certain probabilistic distribution [7].

Research on automatically generating tests or checking MRs in metamorphic testing has been reported. In [8], test generation was implemented based on solving a goal in a constraint system. The research reported in [10] introduced an automated metamorphic testing framework, where the program under test run with the original test inputs in parallel with the test inputs that were transformed from the original inputs based on a MR. A random test generation for testing image processing applications was discussed in [9]. First a new image is generated through randomly selecting each black pixel with a probability from a reference image, and then the images are transformed into other images based on MRs such as rotation or intersection of images, which are the test inputs for testing the image processing applications. Shan and Zhu reported a method to produce structurally complex test cases by data mutation in [14]. First a set of mutation operators are defined and a set of seed test cases are created, and then a set of new test cases are generated through applying mutation operators to the seed test cases [14]. The approach can be directly used for generating structurally complex test cases for metamorphic testing except the MRs instead of mutation operators are applied to the seed test cases. Although metamorphic testing can be applied to individual program component such as the research on function level metamorphic testing reported in

[13], most of the work focuses on system testing by considering the properties of entire systems [2][3] since the correctness of the output of an individual component is even more difficult to be decided in a "non-testable" program due to the complexity of the running context of an individual component. In this paper, we only consider the system level metamorphic testing.

The evaluation of test adequacy is particularly important to metamorphic testing because we need develop not only adequate test cases but also adequate MRs. How can we know the adequacy of the MRs identified or the test cases developed for a metamorphic testing? In this paper, the adequacy of many widely used test coverage criteria is guaranteed when tests are generated by the MISTA tool. The self-checking metamorphic testing reported in [6] discussed the approach for evaluating the test adequacy, but the adequacy of test coverage criteria in this research is automatically ensured.

V. Conclusion

Metamorphic testing has been used for testing systems that do not have test oracles. However, checking only MRs among outputs is not strong enough to ensure the testing quality. A rigorous guideline for generating tests is important for improving the quality of metamorphic testing. In this paper, we proposed an approach to model-based test generation for metamorphic testing. The adequacy of test coverage criteria including path coverage, state coverage, and function coverage can be guaranteed through automated test generation.

The effectiveness of our approach has been demonstrated by testing a "non-testable" image-processing program. The model we built for the image processing program defined all functions and work flows for recognizing cell components in a cellular image. Analysis including simulation, state and transition reachability analysis, deadlock analysis, and verification of goal states had been performed on the model of the cellular image processing to ensure its correctness, which provided a solid foundation for the test generation. Several sets of tests had been generated for adequately covering the test coverage criteria we selected. Comparing to our previous work on self-checking metamorphic testing discussed in reference [6], the adequacy of selected test coverage criteria was guaranteed in the model-based metamorphic testing. In addition, the model-based metamorphic testing can rigorously generate tests that adequately cover complex test coverage criteria like all state coverage or all path coverage. Based on our knowledge, we haven't seen any similar work on metamorphic testing. In the case study, over than 1300 tests were generated to test all states in the model. Without a rigorous test generation approach or a powerful testing tool, it was almost impossible to create so many tests to adequately test the system. Tests generated from the test model were finally translated into tests for testing the system implementation. We identified 5 MRs for testing the image processing program and these MRs were applied to the initial tests to create MR related tests. Due to the space limitation, we did not describe the details of the code-level tests, but the basic idea has been clearly explained. In order to improve the productivity of metamorphic testing, our future work will be focused on automatic generation of MR related tests using the tool MISTA.

References

[1] W. K. Chan, S. C. Cheung, and K. R. Leung, "A metamorphic testing approach for online testing of service-oriented software applications", International Journal of Web Services Research 4, 1, pp. 60 – 80, 2007.

[2] T. Y. Chen, S. C. Cheung, and S. Yiu, "Metamorphic testing: a new approach for generating next test cases", Tech. Rep. HKUST-CS98-01, Dept. of Computer Science, Hong Kong Univ. of Science and Technology, 1998.

[3] T. Y. Chen, J. W. K. Ho, H. Liu, and X. Xie. "An innovattive approach for testing bioinformatics programs using metamorphic testing", BMC Bioinformatics, pp. 10-24, 2009.

[4] T. Y. Chen, D. H. Huang, T. H. Tse, and Z. Q. Zhou, "Case studies on the selection of useful relations in metamorphic testing", In Proc. of the 4th Ibero-American Symposium on Software Engineering and Knowledge Engineering, pp. 569-583, 2004.

[5] T. Y. Chen, T. H. Tse, and Z. Q. Zhou, "Fault-based testing without the need fo oracles". Info. and Soft. Tech., 44(15), pp. 923 – 931, 2002.

[6] J. Ding, T. Wu, J. Q. Lu, X. Hu, "Self-Checked Metamorphic Testing of an Image Processing Program," The 4th IEEE Intl. Conf. on Security Software Integration and Reliability Improvement, Singapore, 2010.

[7] J. W. Duran, S. C. Ntafos, "An Evaluation of Random Testing", IEEE Transactions on Software Engineering, Vol. SE-10, No. 4, pp438-443, July 1984.

[8] A. Gotleib, and B. Botella, "Automated metamorphic testing", In Proc. of 27th Annual International Computer Software and Applications Conference, pp. 34-40, 2003.

[9] J. Mayer, R. Guderlei, "On Random Testing of Image Processing Applications," Proc. of Sixth Int. Conference on Quality Software (QSIC'06), pp. 85-92, 2006.

[10] C. Murphy, K. Shen, and G. Kaiser. "Automatic system testing of program without test oracles". In Proc. of 2009 ACM Intl. Symposium of Software Testing and Analysis (ISSTA), 2009.

[11] C. Murphy, G. Kaiser, L. Hu, and L. Wu. "Properties of machine learning applications for use in metamorphic testing". In Proc. of the 20th Int. conference on software engineering and knowledge engineering (SEKE), pp. 867–872, 2008.

[12] C. Murphy, K. Shen, and G. Kaiser, "Using JML runtime assertion checking to automate metamorphic testing in applications without test oracles", In Proc. of the 2nd IEEE International Conference on Software Testing, Verification and Validation (ICST), 2009.

[13] C. Murphy. "Metamorphic Testing Techniques to Detect Defects in Applications without Test Oracles". Doctoral dissertation, Columbia University, 2010.

[14] L. Shan, and H. Zhu, "Generating Structually Complex Test Cases By Data Mutation: A Case Study of Testing An Automated Modelling Tool". The Computer Journal, Vol. 52, No. 5, 2009.

[15] E.J. Weyuker, "On testing non-testable program", Computer Journal, vol. 25, (4), pp. 465- 470, 1982.

[16] X. Xie, J. Ho, C. Murphy, G. Kaiser, B. Xu, and T. Y. Chen, "Application of metamorphic testing to supervised classifiers". In Proc. of the 9th Intl. Conf. on Quality Software, pp. 135 – 144, 2009.

[17] D. Xu, J. Ding, "Prioritizing State-Based Aspect Tests". Proc. of ICST 2010, pp. 265-274, Paris, France, 2010.

[18] D. Xu, "A Tool for Automated Test Code Generation from High-Level Petri Nets". 32nd Int. Conf. on Apps. and Theory of Petri Nets , Newcastle, UK, June 20-24, 2011.

[19] D. Xu, D., K. E. Nygard, "Threat-Driven Modeling and Verification of Secure Software Using Aspect-Oriented Petri Nets". IEEE Trans. on Software Engineering. 32(4), 265–278, 2006.

[20] D. Xu, R. A. Volz, T. R. Ioerger, J. Yen, "Modeling and Analyzing Multi-Agent Behaviors Using Predicate/Transition Nets". International Journal of Software Engineering and Knowledge Engineering 13(1), 103–124, 2003.

Verifying Aspect-Oriented Activity Diagrams Against Crosscutting Properties with Petri Net Analyzer

Zhanqi Cui, Linzhang Wang, Xi Liu, Lei Bu, Jianhua Zhao, Xuandong Li
State Key Laboratory of Novel Software Technology
Department of Computer Science and Technology
Nanjing University, Nanjing, 210046, China
zqcui@seg.nju.edu.cn, lzwang@nju.edu.cn, liux@seg.nju.edu.cn, {bulei, zhaojh, lxd}@nju.edu.cn

***Abstract*—Aspect-oriented model-driven approaches are proposed to model and integrate crosscutting concerns at design phase. However, potential faults that violate desired properties of the software system might still be introduced during the process. Verification technique is well-known for its ability to assure the correctness of models and uncover design problems before implementation. This paper presents a framework to verify aspect-oriented UML activity diagrams based on Petri net verification techniques. For verification purpose, we transform the integrated activity diagrams into Petri nets. Then, the Petri nets are checked against formalized crosscutting requirements to detect potential faults. Furthermore, we implement a tool named Jasmine-AOV to support the verification process. Case studies are conducted to evaluate the effectiveness of our approach.**

Keywords: aspect-oriented modeling; verification; model checking; activity diagram; Petri net

I. Introduction

Dealing with crosscutting concerns has been a critical problem during software development life cycles. In our previous work [1], we proposed an aspect-oriented model-driven approach based on UML activity diagrams. The approach shifts aspect-oriented techniques [2] from a code-centric to a model-centric, which is employed to handle the crosscutting concerns during design phases. Thus, it alleviates software complexity in a more abstract level. The primary functional concerns are modeled with activity diagrams, and crosscutting concerns are modeled with aspectual activity diagrams, respectively. Then the overall system design model, which is also an activity diagram, is integrated by weaving aspect models into primary models.

Design models are widely used as a basis of subsequent implementation [3][4] and testing [5][6][7] processes. It is costly if defects in design models are discovered at later implementation and testing stages. Aspect-oriented modeling techniques cannot guarantee the correctness of produced design models. For instance, wrong weaving sequences may cause the integrated models violate system crosscutting requirements. Therefore, assuring the correctness of the aspect-oriented design models is vitally important. So far, the applicable approach is manual review. It is time consuming and dependent on reviewers' expertise. However, existing automatic verification tools cannot deal with UML diagrams directly.

As an ongoing work, in this paper, in order to ensure crosscutting concerns are correctly modeled, we propose a rigorous approach to automatically verify aspect-oriented models (activity diagrams) by using Petri net based verification techniques. Firstly, the integrated activity diagram is translated into a Petri net. Then, crosscutting concerns in system requirements are refined to properties in the form of CTL formulas. Finally, the Petri net is verified against the formalized properties.

The rest of this paper is organized as follows. Section 2 presents backgrounds of activity diagrams, Petri nets, and a running example. Section 3 discusses the verification of aspect-oriented activity diagrams. Section 4 presents 2 case studies and evaluations of our approach. Section 5 reviews the related work. Finally, section 6 concludes the paper and discusses future work.

II. Background

In this section, we briefly introduce UML activity diagrams and Petri nets, and a running example that will be employed to demonstrate our approach in following sections.

A. Activity Diagrams and Petri nets

The UML activity diagram is a powerful tool to describe control flow based program logic at different levels of abstraction. Designers commonly use activity diagrams to describe the sequence of behaviors between classes in a software system. Nodes and edges are two kinds of elements in activity diagrams. Nodes in activity diagrams are connected by edges. We formally define activity diagrams as follows.

Definition 1. (Activity Diagram). An activity diagram AD is a 4-tuple (N, E, F), where:

- $N = \{n_1, n_2, \ldots, n_i\}$ is a finite set of **nodes**, which contains action, initial/final, decision/merge and fork/join nodes, $n_I \in N$ is the initial activity state, $N_F \subseteq N$ is a set of final activity states;
- $E = \{e_1, e_2, \ldots, e_j\}$ is a finite set of **edges**;
- $F \subset (N \times E) \cup (E \times N)$ is the flow relation between nodes and edges.

Due to the nature of UML is semi-formal and UML diagrams are design-oriented models, translating activity diagrams into formal verification-oriented models is necessary before verification. In this approach, we translate activity diagrams into Petri nets, because in UML 2, the semantics of activity diagrams is explained in terms of Petri net notations [9], like tokens, flows etc. Petri net is a formal specification language that is widely used to model software behaviors. A Petri net consists of places, transitions, and arcs. Like UML activity diagrams, Petri nets offer a graphical notation for stepwise processes that include choice, iteration, and concurrent execution. On the other hand, Petri nets have a precise mathematical definition of their execution semantics, with a well-developed mathematical theory for process analysis. A Petri net is formally defined as follows.

Definition 2. (Petri net) A Petri net [8] is a 4-tuple $PN = \{P, T, A, M_0\}$, where

- P is a finite set of **places** and T is a finite set of **transitions**, and P and T are disjoint.
- A is a finite set of **arcs** connect between places and transitions, where $A \subseteq (P \times T \bigcup T \times P)$.
- M_0 is the initial **marking**, $M_0(p)$ denotes the number of tokens at place p under initial marking M_0.

Places, transitions and arcs in A are drawn as circles, boxes and arrows, respectively. We do not consider weights of arcs in this paper for simplification.

B. Running Example

We adapt the order processing scenario from [9] as a running example to demonstrate our approach. There are 4 crosscutting concerns related to this scenario: authentication, validation, logging, and informing.

Figure 1 is the primary model of the order processing scenario, which consists of 3 main steps: fill order, ship order, and close order.

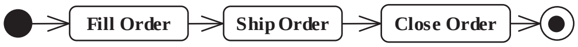
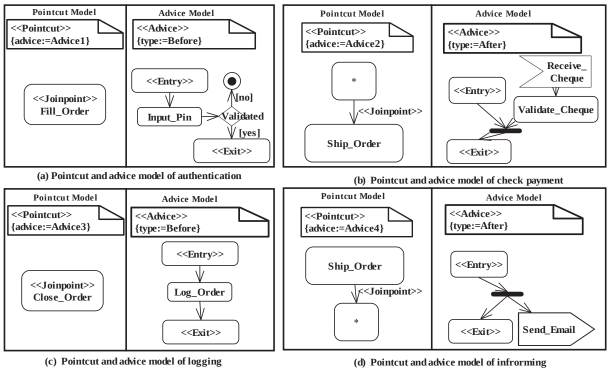


Figure 1. The primary model of the order processing scenario

Based on our previous aspect-oriented modeling approach [1], the crosscutting concerns of the running example are modeled in Figure 2.



(a) Pointcut and advice model of authentication

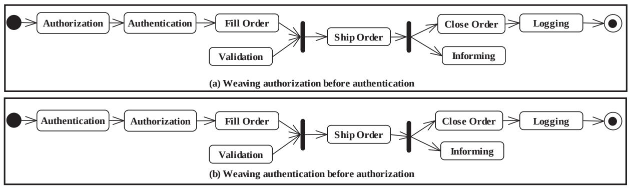
(b) Pointcut and advice model of check payment

(c) Pointcut and advice model of logging

(d) Pointcut and advice model of infrorming

Figure 2. Pointcut and advice models of the order processing scenario

In order to understand how crosscutting concerns will affect primary functionalities, aspect models are integrated with primary models to generate an overall system design model. Different weaving sequences would produce different integrated models. For example, we add an authorization aspect in the running example, which describes the logged-in user need to be checked whether she/he has the permission to fill orders. If the authorization aspect is woven before authentication, then the result of integration is shown in Figure 3 (a). Otherwise, if the authentication aspect is woven before authentication, then the result of integration is shown in Figure 3 (b). As we know, the legal user has to be logged-in before being checked whether the corresponding permission is granted or not. As a result, the authentication aspect should be woven firstly, and Figure 3 (b) is the correct integration result we expected. Extensive explanations about the integration process can be also found in [1].



(a) Weaving authorization before authentication

(b) Weaving authentication before authorization

Figure 3. Two different integrated models of the order processing scenario

III. VERIFYING ASPECT-ORIENTED MODELS

In our previous work, aspect-oriented models, including primary models, aspect models, as well as integrated models, were all depicted with UML activity diagrams. Since the correctness of the integration process cannot be guaranteed, how to ensure the consistence between the integrated activity diagrams and crosscutting requirements becomes a critical research problem. In UML 2, the semantics of activity diagrams is explained in terms of Petri net. There are also various automatic tools, i.e., LoLA (a Low Level Petri Net Analyzer) [10], verifying Petri nets against specified properties. As a result, if we can translate activity diagrams into Petri nets, we could verify the activity diagram models by verifying corresponding Petri net models for specific properties. In this section, we first discuss transformation from activity diagrams to Petri nets, and then present the verification against crosscutting concerns.

A. Transforming from Activity Diagrams to Petri Nets

We adapt the mapping semantics of control-flows in UML 2 activities in [9] to convert activity diagrams into Petri nets. Basically, action nodes and fork/join nodes are translated to net transitions, control nodes (initial, final, decision, and merge nodes) become net places, and edges are transformed to net arcs. Auxiliary transitions or places are added when the ends of an arc both are transitions or both are places. For simplification, we restrict an activity diagram only consists of action nodes, control nodes, and control flows in this approach. The transformation of more complex activity diagrams (containing data flows, exceptions, and expansions etc.) is straightforward based on transformation rules in [11].

Based on the mapping rules in [12], we construct an algorithm to transform activity diagrams to Petri nets and implement in our verification tool to provide automatic transformation support. The algorithm is described in List 1. With the algorithm, the activity diagram of the running example in Figure 3 (b) is converted to the Petri net in Figure 4. The transformed Petri net is a bi-simulation of the activity diagram, which means they are semantically equal. So we can achieve the verification of the activity diagram by verifying the equivalent Petri net against same system properties.

List 1. Convert an activity diagram into a Petri net

```
**Input**: $AD$ := an activity diagram
**Output**: $PN\{P, T, A, M_0\}$ := a Petri net
**for each** node $N$ **in** $AD$
        **if** $N$ is an initial node, final node, decision node, or merge node
                Generate a corresponding place in $PN.P$
        **else** // action node, fork node, or join node
                Generate a corresponding transition in $PN.T$
**for each** edge $E$ **in** $AD$
        $N_1$ := source node of $E$ in $AD$
        $N_2$ := target node of $E$ in $AD$
        $M_1$ := corresponding place or transition of $N_1$ in $PN.P$
        $M_2$ := corresponding place or transition of $N_2$ in $PN.P$
        **if** both $N_1$ and $N_2 \in$
            (initial nodes $\cup$ final node $\cup$ decision node $\cup$ merge node)
                Generate an auxiliary transition $T_1$ in $PN.T$
                Generate an arc start from $M_1$ to $T_1$ in $PN.A$
                Generate an arc start from $T_1$ to $M_2$ in $PN.A$
        **else if** both $N_1$ and $N_2 \in$ (action node $\cup$ fork node $\cup$ join node)
                Generate an auxiliary place $P_1$ in $PN.P$
                Generate an arc start from $M_1$ to $P_1$ in $PN.A$
                Generate an arc start from $P_1$ to $M_2$ in $PN.A$
        **else**
                Generate an arc start from $M_1$ to $M_2$ in $PN.A$
**for each** place without an incoming arc
        Generate an initial token for that place in $PN.M_0$
**return** $PN$
```

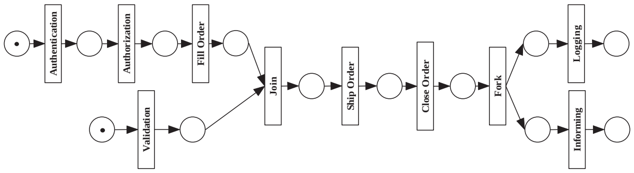


Figure 4. The Petri net transformed from the order processing scenario

B. Verifying Petri Nets

Crosscutting concerns describe the running sequences between advices and primary behaviors in all paths of integrated models. These properties can be described in the form of Computation Tree Logic (CTL) formulas [13] naturally. CTL formulas cannot be generated from aspect models by synthesizing conditions of join points specified by pointcut models and checking the corresponding advice models appears at right places. This is because that the context specified by a pointcut model would be changed after integration, and the join points matched by the pointcut model could no longer exist. In this approach, the properties to be checked are directly refined from crosscutting requirements.

1) Properties specified from the requirement

Based on the Petri net generated, we can easily analysis reachability, safety, liveness, and fairness properties [8]. In this approach, we only focus on checking properties that are closely related to crosscutting concerns. We categorize crosscutting concerns from two facets. Firstly, according to the execution sequence between action in advice models and join points, a crosscutting concern can be either executing before or after join points. Secondly, the execution of a crosscutting concern is either sequential or parallel with the primary behaviors. Sequential crosscutting concerns are synchronous features that their running positions are restricted by the join points. Parallel crosscutting concerns are asynchronous features that are running concurrently with primary actions and they are finished or started by the join points.

a) Before-crosscutting concerns

A before-crosscutting concern specifies some extra behaviors must be performed before matched join points. Actions specified by a sequential before aspect model are executed between the join point node and the predecessor node of the join point in the primary model. The key word of sequential before-crosscutting concerns in requirements level is " before". A parallel before aspect specifies crosscutting actions that must be finished by the join point edge. The key word of parallel before-crosscutting concerns in requirements is "be finished by". In the integrated model, the actions of the crosscutting concern are running concurrently with the primary behaviors, and then synchronized at the join node which replaced the join point edge.

In corresponding Petri nets, assume *jp* is the transition transformed from one of the join point, *ad* is the transition transformed from the structured activity node that represents the advice model. The requirement of a before aspect can be represented in the form of the CTL formula as: $\mathbf{AG}\neg((ad \wedge \mathbf{EX}(\neg ad \wedge \neg jp)) \vee ((\neg ad \wedge \neg jp) \wedge \mathbf{EX}\ jp))$.

b) After-crosscutting concerns

An after-crosscutting concern specifies some actions must be performed after matched join points. An after-crosscutting concern can also be either a sequential or a parallel aspect with respect to the flows of primary models. Actions specified by a sequential after aspect model are the actions executed between the join point node and the successor node of the join point in the primary model. The key word of sequential after-crosscutting concerns in requirements level is "after". A parallel after aspect specifies crosscutting actions must be started by the join point edge. The key word of parallel after-crosscutting concerns in requirements is "be started by". In the integrated model, the actions of the crosscutting concern are enabled by the fork node, which replaced the join point edge, and then running concurrently with primary behaviors.

In corresponding Petri nets, assume *jp* is the net transition transformed from the join point, *ad* is the net transition transformed from the structured activity node that represents the advice model. The requirement of a sequential after aspect can be represented in the form of the

CTL formula as: $\mathbf{AG}\neg((jp \wedge \mathbf{EX}(\neg jp \wedge \neg ad)) \vee ((\neg jp \wedge \neg ad) \wedge \mathbf{EX}\ ad))$.

2) Conflicts of Multiple Crosscutting Concerns

The CTL formula need to be adjusted if more than one crosscutting concerns (which are all "before" aspects or are all "after" aspects) match a same join point. Because the running sequence between one aspect and a join point can be affected by other aspects of the same before/after kind, which match the same join point. For instance, in the running example, the authentication and authorization concerns are conflicted because they both are before-crosscutting aspects and they have same join point, the "Fill_Order" action. The running sequence of authentication aspect and "Fill_Order" operation will be changed from "Authentication->Fill_Order" to "Authentication->Authorization->Fill_Order" after the weaving of authorization aspect.

a) Conflicts between two before-crosscutting concerns

For a before-crosscutting concern cc_1 with advice model ad_1 and join point jp_1, if any other before aspect, which matches the same join point jp_1 and weaves after cc_1, then some extra actions are performed after ad_1 and before jp_1. Assume it's a before-crosscutting concern cc_2 with advice ad_2 weaves after cc_1, then jp_1 should be replaced by ad_2 in the CTL formula of cc_1 as: $\mathbf{AG}\neg((ad_1 \wedge \mathbf{EX}(\neg ad_1 \wedge \neg ad_2)) \vee ((\neg ad_1 \wedge \neg ad_2) \wedge \mathbf{EX}\, ad_2))$.

b) Conflicts between two after-crosscutting concerns

For an after-crosscutting concern cc_1 with advice ad_1 and join point jp_1, if any other after aspect , which matches the same join point $jp_1$and weaves after cc_1, then some extra actions are performed after jp_1 and before ad_1. Assume it's an after-crosscutting concern cc_2 with advice ad_2 weaves after cc_1, then jp_1 should be replaced by by ad_2 in the CTL formula of cc_1 as: $\mathbf{AG}\neg((ad_2 \wedge \mathbf{EX}(\neg ad_2 \wedge \neg ad_1)) \vee ((\neg ad_2 \wedge \neg ad_1) \wedge \mathbf{EX}\, ad_1))$.

3) Verification

After the system crosscutting properties are refined as a set of CTL formulas. We can verify the Petri net against specified CTL formulas generated. If the verification is passed, it means the model satisfies the corresponding crosscutting requirements. Otherwise, the model violates the corresponding crosscutting requirements to some extent, which means further revision about the model is needed.

In the running example, the integrated model in Figure 1 (a) and (b) are both verified against the crosscutting requirements of authentication, authorization, validation, logging, and informing. First, the integrated models are transformed to Petri nets. Then the 5 crosscutting requirements are refined to 5 CTL formulas. Finally, Petri net analyzer LoLA is employed to verify the two Petri nets against the formalized crosscutting requirements, respectively.

The Petri net transformed from the model in Figure 3 (b) passes the verification process and output "result: true" for all the 5 CTL formulas. While the Petri net transformed from the model in Figure 3 (a) fails when verifying against the 2 CTL formulas generated from authentication and authorization requirements, and passes the verification against the other 3 CTL formulas. This verification result shows that the crosscutting requirements of authentication and authorization do not hold in this aspect-oriented model. After correcting the weaving preference fault and integrating the aspect model again, the new integrated model passes the verification process.

C. Tool Implementation

We implemented a tool named Jasmine-AOV[1] based on Topcased[2] and LoLA[3]. As Figure 5 shows, this tool is composed of 4 main parts: Model Transformer, Crosscutting Concern Manager, CTL Generator, and Model Checker. The Model Transformer converts an activity diagram to a Petri net automatically. The inputs of Model Transformer are UML diagrams designed by Topcased in the form of XML file and the outputs of the tool are Petri net files which are readable for LoLA to perform verification tasks. The Crosscutting Concern Manager is used to manage mapping relations between crosscutting concerns in requirements and elements in corresponding activity diagrams. It provides an assistant for mapping textual crosscutting requirements to design activity diagrams. The CTL Generator can automatically generate CTL formulas from crosscutting requirements that are mapped to design models. The CTL Generator also supports users to input CTL formulas manually. Model Checker is implemented by directly wrapped an existing checker, LoLA. It can verify the Petri net against crosscutting properties in the format of CTL formulas and report the result.

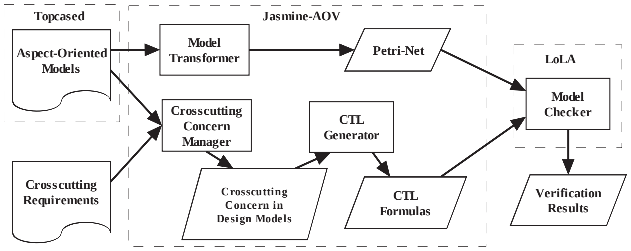


Figure 5. The framework of Jasmine-AOV

The screenshot of Jasmine-AOV is in Figure 6. The "Crosscutting concerns" area manages the crosscutting requirements which are mapped to design models. The "New Crosscutting Concern" dialog provides an assistant for mapping textual crosscutting requirements to design activity diagrams. The "Petri net" area displays the Petri net transformed from the corresponding activity diagram. The "CTL Formulas" area lists the formulas refined from crosscutting concerns in the "Crosscutting concerns" area automatically or wrote by users manually. The "Verification Results" area outputs the results of verifying the Petri net in

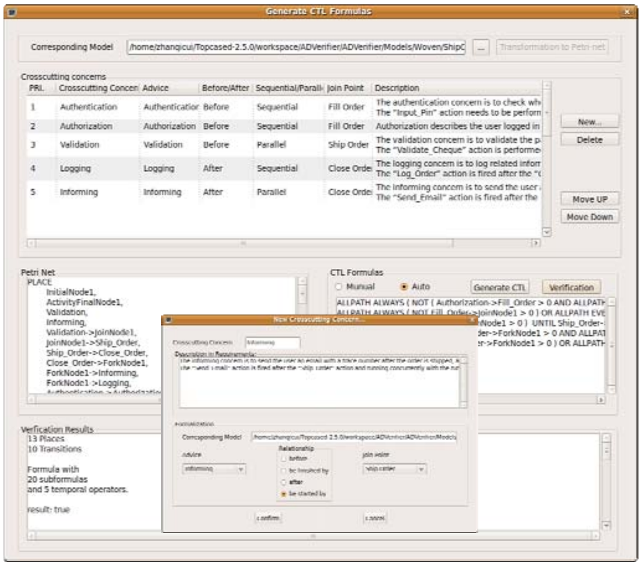
[1] Jasmine-AOV, http://seg.nju.edu.cn/~zqcui/Jasmine-AOV

[2] Topcased, http://www.topcased.org/

[3] LoLA, http://www.informatik.uni-rostock.de/tpp/lola/

the "Petri net" area against the CTL formulas in the "CTL Formulas" area by LoLA.

Writing complex CTL formulas is not easy for a software engineer without proper training about formal methods. To tackle this problem, we implemented the CTL Generator to assist generating CTL formulas automatically. As Figure 6 shows, the user only need to select actions which is the advice, the join points, and the relationship between the advice and the join points, based on the textual description of the crosscutting concern. After this information is inputted, the CTL Generator generates a CTL formula for the crosscutting concern and adjusts CTL formulas if there is more than one aspect of the same before/after type apply on a same join point.



Corresponding Model /home/zhangicui/Topcased-2.5.0/workspace/ADVerifier/ADVerifier/Models/Woven/ShipC

Crosscutting concerns

PRI.	Crosscutting Concen	Advice	Before/After	Sequential/Parall	Join Point	Description
1	Authentication	Authenticatior	Before	Sequential	Fill Order	The authentication concern is to check wh The "Input_Pin" action needs to be perforn
2	Authorization	Authorization	Before	Sequential	Fill Order	Authorization describes the user logged in
3	Validation	Validation	Before	Parallel	Ship Order	The validation concern is to validate the p The "Validate_Cheque" action is performe
4	Logging	Logging	After	Sequential	Close Orde	The logging concern is to log related infor The "Log_Order" action is fired after the "c
5	Informing	Informing	After	Parallel	Close Orde	The informing concern is to send the user The "Send_Email" action is fired after the

Petri Net

```
PLACE
    InitialNode1,
    ActivityFinalNode1,
    Validation,
    Informing,
    Validation->JoinNode1,
    JoinNode1->Ship_Order,
    Ship_Order->Close_Order,
    Close_Order->ForkNode1,
    ForkNode1->Informing,
    ForkNode1->Logging,
```

CTL Formulas

○ Manual ● Auto

```
ALLPATH ALWAYS ( NOT ( Authorization->Fill_Order > 0 AND ALLPATH
ALLPATH ALWAYS ( NOT Fill_Order->JoinNode1 > 0 ) OR ALLPATH EVE
Node1 > 0 )  UNTIL Ship_Order-
der->ForkNode1 > 0 AND ALLPAT
er->ForkNode1 > 0 ) OR ALLPATH
```

Verification Results

```
13 Places
10 Transitions

Formula with
20 subformulas
and 5 temporal operators.

result: true
```

New Crosscutting Concern...

Crosscutting Concern: Informing

Description in Requirements:

The informing concern is to send the user an email with a trace number after the order is shipped. ...
The "Send_Email" action is fired after the "Ship_Order" action and running concurrently with the ...

Formalization

Corresponding Model /home/zhangicui/Topcased-2.5.0/workspace/ADVerifier/ADVerifier/Models

Advice: Informing

Relationship: ○ before ○ be finished by ○ after ● be started by

Join Point: Ship Order

Figure 6. The screenshot of Jasmine-AOV

IV. Evaluation and Case Suites

To evaluate the effectiveness of our approach, we have applied our approach to the design models adapted from the Ship Order example in [9] and the Telecom System[4]. The Ship Order example contains 5 crosscutting concerns and the Telecom System contains 6 crosscutting concerns. For both of the 2 case studies, we transformed the integrated models to Petri nets, and mapped crosscutting requirements to the design models with the help of the tool. Then, corresponding CTL formulas of verification tasks are generated automatically. Finally, the Petri nets are checked against the CTL formulas generated.

The faults of aspect-oriented models, which can be caused by design defects or incorrect integration processes, are categorized as follows:

1. Aspect model faults

   a) Incorrect weaving preference. The priorities of aspect models are incorrectly assigned. This kind of fault will lead to match join pints faults or running sequence changed unexpected.

   b) Incorrect binding between pointcut model and advice model. The pointcut model is incorrectly mapped to an unrelated advice model. This kind of fault will result in improper advice models apply at some join points.

2. Pointcut model faults

   a) Overmatch/Mismatch join points. The pointcut model matches extra join points or miss some join points should be matched. The consequence of this kind of faults is that extra advices are performed at unexpected join points or desired advices are not going to be performed at join points.

   b) Incorrect position of join points. The element which serves as a join point in the pointcut model is incorrectly appointed. The phenomenon of this kind of faults is that advices are applied at incorrect points of the primary model.

3. Advice model faults

   a) Incorrect type of advice models. The type of the advice model is declared incorrectly. This kind of fault will cause the running sequence between the advice model and the primary model change unexpectedly.

To further evaluate the ability of our approach to detect the faults of aspect-oriented models, mutated models are created based on preceding category of aspect model faults. 26 and 28 model mutants are constructed for the 2 case studies, respectively. Table 1 classifies all these model mutants by their fault types. All of them are killed because they violate the crosscutting requirements from various ways and these violations are detected by the verification process. This result illustrates the ability of our approach to find the faults in aspect-oriented models and to improve the quality of design models.

TABLE I. Model Mutants of the 2 Case Studies

Fault Types		Ship Order	Telecom System
Aspect model faults	Incorrect weaving preference	1	1
	Incorrect binding	5	3
Pointcut model faults	Overmatch join points	5	6
	Mismatch join points	5	6
	Incorrect position of join point	5	6
Advice model faults	In correct type of advice models	5	6
Number of model mutants in total		26	28
Mutants killed		26	28

V. Related Work

There are many research projects on bringing aspect-oriented ideas to software requirement engineering from different perspectives. Whittle and Araujo [14] focus on scenario-based requirements and composing them with aspects to generate a set of state machines that represent the composed behaviors from both aspectual and non-aspectual scenarios. In contrast, our approach is carried out at the design level instead of requirement level. However, our approach can be enhanced with the aspect mining tool at

[4] AJDT toolkit: http://www.eclipse.org/ajdt

requirements level, like EA-Miner [15], by inputting crosscutting concerns detected by these tools to our Jasmine-AOV tool for verification.

There is also a large body of research on aspect-oriented modeling. But most of them do not concern about the correctness of the integrated model and provides verification supports. In a ddition to support asp ect-oriented modeling and integration, our approach also formally checks whether crosscutting concerns in requirements are correctly designed. Xu et al. proposed to model and compose aspects with finite state machines, and then transformed to FSP processes and cheeked by LTSA model checker against all system requirements [16]. Whereas our approach is carried out on activity diagrams and only focuses on checking crosscutting concerns. Furthermore, we cat egorize 4 k inds of crosscutting concerns and generate CTL formulas automatically from crosscutting concern specifications, which bridges the gaps between crosscutting requirements and aspect-oriented design models. We also provide a solution for the conflicts between crosscutting concerns.

Several model checking techniques have been presented for aspect-oriented programs. Denaro et al. fi rst reported a preliminary experience on verifying deadlock freedom of a concurrent aspect [17]. They first derived PROMELA process templates from aspect-oriented units, and then analysis the aspect-oriented program with SPIN. Ubayashi and Tamai [18] proposed to app ly model checking techniques to v erify whether the result of weaving classes and aspects contained unexpected behaviors like deadlocks. The approach in this paper is different from these methods, because our approach is carried out at the model level other than the program level. In comparison, our approach can identify system faults at an earlier stage, and save costs to revise programs when detecting design faults at implementation or maintenance phase.

VI. Conclusions and Future Work

This paper presents a f ramework to verify aspect-oriented UML activity diagrams by using Petri net based verification techniques. For verification purpose, we transform the integrated activity diagrams into Petri nets. Then, crosscutting properties of the system are refined as a set of CTL formulas. Last, the Petri net is verified against the refined CTL formulas. The v erification result shows whether the Petri net satisfy the requirements or not. We can reason whether the integ rated activity diagram meets the requirement since they ar e equivalent. In ot her words, we can claim that the aspect-oriented modeling is correct with respect to speci fied crosscutting requirements. Two case studies have been carried out to demonstrate the feasibility and effectiveness of our approach. Concerning the future work, we wil l focus on testing system implementations against aspect-oriented models have been verified.

Acknowledgment

We would like to th ank Professor Karsten Wolf at University Bamberg, who is th e author of LoLA, for his help in dealing with problems encountered when integrating LoLA into our tool Jasmine-AOV. This work is supported by the National Natural Science Foundation o f China (No.61021062， No. 61170066), the Nat ional 863 High-Tech Program of China (No. 20 12AA011205), and the National Grand Fundamental Research 973 Program of China (No. 2009CB320702).

References

[1] Z. Cui, L. Wang, X. Li, and D. Xu. Modeling and integrating aspects with UML activity diagrams. In Proceedings of the ACM Symposium on Applied Computing, Honolulu, Hawaii, ACM, New York, NY, 2009, pp. 430-437.

[2] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-Oriented Programming. In the annual European Conference on Obj ect-Oriented Programming, 1997, pp. 220–242.

[3] X. Li, Z. Liu, J. He, and Q. Long.Ge nerating a Prototy pe From a UML Models of System Requirements. Distributed Computing and Internet Technology. Lecture Notes in Computer Science 3347, Berlin, Heidelberg: Springer, 2005, pp. 135-154.

[4] A. Fischer. Mapping UML designs to Java. In Proceedings of the 15th ACM SIGPLAN conference on O bject-oriented programming, systems, languages, and applications, 2000, pp. 178-187.

[5] L. Wang, J. Yuan, X. Yu, J. Hu, X. Li, and G. Zheng. Generating Test Cases from UML Activity Diagram based on Gray-Box Method. In Proceedings of the 11th Asi a-Pacific Software Engineering Conference, 2004, pp. 284-291.

[6] C. Nebut, F. Fleurey, Y. L. Traon, and J. Jezequel. Automatic Test Generation: a Use Case Driven Approach. IEEE Transactions on Software Engineering, Vol.32, No.3, 2006, pp. 140-155.

[7] M. Chen, X. Qiu, W. Xu, L. Wang, J. Zhao, and X. Li. UML Activity Diagram Based Automatic Test Ca se Generation for Java Program s. In The Computer Journal, Vol.52, No.5, Oxford Press, 2009, pp. 545-556.

[8] T. Murata, Petri nets: Properties, analysis and applications, Proceedings of the IEEE , Vol.77, No.4, Apr 1989, pp. 541-580.

[9] OMG, UML Superstructure v2.1, http ://www.omg.org/technology/documents/formal/uml.htm.

[10] K. Schmidt. LoLA A Low Level Analyser. In Proceedings of the Application and Theory of Petri Nets, 2000, pp. 465-474.

[11] H. Störrle, Structured nodes in UML 2.0 activities. Nordic J. of Computing, 11(3), 2004, pp. 279-302.

[12] H. Störrle. Semantics of Control-Flow in UML 2.0 Activities. In Proceedings of the 2004 IEEE Symposium on Visual Languages - Human Centric Computing, 2004, pp. 235-242.

[13] E. M. Clarke, E. A. Emerson, and A. P. Sistla, Automatic verification of finite-state concurrent systems using temporal logic specifications. ACM Trans. Program. Lang. Syst. 8, 2, pp. 244-263.

[14] J. Whittle, J. Araujo. Scenario Modelling with As pects. In IEEE Software, Vol 151, Issue 4, Aug. 2004, pp. 157–172.

[15] A. Sampaio, A. Rashid, R. Chitchyan, and P. R ayson. EA-Miner: Towards Automation in Aspect-Oriented Requirements Engineering. Transactions on Aspect-Oriented Software Development III, Lecture Notes In Computer Science, Vol. 4620. Springer-Verlag, Berlin, Heidelberg, pp. 4-39.

[16] D. Xu, O. E. Ariss, W. Xu, and L. Wang, Aspect-Oriented Modeling and Verification with Finite State Machines, Journal of Computer Science and Technology, 24(5), Sept. 2009, pp. 949-961.

[17] G. Denaro, and M. Monga. An experience on ve rification of aspect properties. In Proceedings of the 4th international Works hop on Principles of Software Evolution, ACM, New York, NY, 2001, pp. 186-189.

[18] N. Ubayashi, and T. Tamai. Aspect-oriented programming with model checking. In Proceedings of the 1st international Conference on Aspect-Oriented Software Development, ACM, New York, NY, 2002, pp. 148-154.

Parametric Verification of Time Workflow Nets

Hanifa Boucheneb
Laboratoire VeryForm, École Polytechnique de Montréal,
P.O. Box 6079, Station Centre-ville, Montréal, Québec,Canada, H3C 3A7
hanifa.boucheneb@polymtl.ca
Kamel Barkaoui
Laboratoire CEDRIC, Conservatoire National des Arts et Métiers,
292 rue Saint Martin, Paris Cedex 03, France
kamel.barkaoui@cnam.fr

Abstract

We consider here the time Workflow nets [6, 10] with parametric initial marking and investigate the parametric verification of reachability properties. We propose a concise state space abstraction, based on the state class graph method, which allows reachability analysis for all possible initial markings. We establish a necessary and sufficient condition for the finiteness of the abstraction and discuss the verification of soundness property.

Keywords: *Time Workflow nets; state class graphs; parametric reachability analysis.*

1 Introduction

A Workflow defines a set of activities and the specific order they are to be enabled into a work process to achieve a common goal such as physical transformation, service provision, or information processing [10]. The development of work process-oriented applications in various domains leads to Workflow specification with complex synchronization mechanisms, routing constructs and time constraints. Therefore, the need for helping designers to check the correctness of Workflow systems in which a large number of flow instances run concurrently is crucial. Workflow nets (WF-net) [14] are Place/Transition nets successfully used in modelling the control-flow dimension of a Workflow. Activities are modelled by transitions and causal dependencies are modelled by places and arcs. Time Workflow nets integrate time constraints of activities in form of firing time intervals associated with transitions [6, 10].

We investigate here the verification of reachability properties of time Workflow nets with parametric initial marking (i.e., number of initial resources, data, etc.). Among methods allowing parametric verification that we can find in the literature, we can cite the three following approaches applicable in the context of untimed Workflow nets: structural analysis [2, 9], forward reachability analysis and backward reachability analysis [1, 7]. All these methods, take advantage of the monotonic property w.r.t. the well ordering relation $\leq$ over markings. Unfortunately, this property is lost in time Workflow nets.

We propose here a forward reachability analysis of time Workflow nets for all its possible initial markings. This analysis consists in constructing a superposition of state class graphs [5] of the model for all its initial markings, while merging paths supporting the same firing sequences. We show how to handle efficiently and symbolically finite or infinite sets of state classes to construct an abstraction which preserves reachability properties. Finally, we establish a necessary and sufficient finiteness condition for the abstraction and discuss briefly (for lack of space) the verification of soundness property.

Section 2 is devoted to the definition and semantics of time parametric Workflow nets. Section 3 proposes a parametric verification approach based on the state class graph method and proves its correctness w.r.t. reachability properties. Finally, the conclusion is presented in Section 4.

2 Time parametric Workflow nets

2.1 Multi-sets

Let P be a nonempty set. A multi-set over P is a function $m : P \longrightarrow \mathbb{N}$, $\mathbb{N}$ being the set of natural numbers, defined also by the formal sum: $\sum_{p \in P} m(p) \bullet p$. We denote P_{MS} and 0 the set of all multi-sets over P and the empty multi-set, respectively. Let $m_1 \in P_{MS}$, $m_2 \in P_{MS}$ and $\prec \in \{\leq, =, <, >, \geq\}$. Operations on multi-sets are defined as usual:

$\forall p \in P,\ p \in m_1$ iff $m_1(p) > 0$.

$m_1 \prec m_2$ iff $(\forall p \in P, (m_1(p) \prec m_2(p)))$.

$m_1 + m_2 = \sum_{p \in P} (m_1(p) + m_2(p)) \bullet p$.

if $m_1 \leq m_2$ then $m_2 - m_1 = \sum_{p \in P} (m_2(p) - m_1(p)) \bullet p$.

2.2 Time parametric Workflow nets

Let $\mathbb{Q}^+$ and $\mathbb{R}^+$ be the sets of non-negative rational and real numbers, respectively and $INT_{\mathbb{X}} = \{[a, b] | (a, b) \in \mathbb{X} \times (\mathbb{X} \cup \{\infty\}\}$, for $\mathbb{X} \in \{\mathbb{N}, \mathbb{Q}^+, \mathbb{R}^+\}$ the set of intervals whose bounds are in $\mathbb{X}$. A Time parametric Workflow net (T-PWN for short) is a parametric Workflow net, where a firing time interval is associated with each transition [11].

Definition 1 *Formally, a T-PWN is a tuple* $\mathcal{N} = (P, T, pre, post, in, out, Is)$ *where:*

- P *and* T *are finite and non empty sets of places and transitions such that* $P \cap T = \emptyset$.
- pre *and* $post$ *are the backward and the forward incidence functions (*$pre, post : T \longrightarrow P_{MS}$*).*
- $in \in P$ *and* $out \in P$ *with* $in \neq out$ *are the input and output places of the Workflow net. Place* in*, called the source place, has no input transition (*$post(t)(in) = 0$*, for* $t \in T$*) and* out*, called sink or final place, has no output transition (*$pre(t)(out) = 0$*, for* $t \in T$*).*
- *For every node* $n \in P \cup T$*, there exists a path from* in *to* n *and a path from* n *to* out*.*
- Is *is the static firing function (*$Is : T \rightarrow INT_{Q^+}$*).* $\downarrow Is(t)$ *and* $\uparrow Is(t)$ *denote the lower and the upper bounds of the static firing interval of* t*.*

The initial marking of $\mathcal{N}$ is not fixed and may be any marking $M_0^K = K \bullet in$ with $K \geq 1$. Then, $\mathcal{N} = \{\mathcal{N}^K | K \geq 1\}$, the initial marking of each $\mathcal{N}^K$ is $K \bullet in$.
Let us first define the behavior of $\mathcal{N}^K$. By definition, the place in of the model has no input transitions. Therefore, the marking of in cannot increase but may decrease by firing transitions. So, the marking of $\mathcal{N}^K$ can be defined as a multi-set over P: $M^K = (K - c) \bullet in + m$, where $c \in \mathbb{N}$, $K - c$ is the current number of tokens in place in (the parametric part) and m is the marking of other places (the non parametric part). Let $M^K = (K - c) \bullet in + m$ be a marking of $\mathcal{N}^K$ and $t \in T$ a transition. The transition t is enabled in M^K iff all required tokens for firing t are present in M^K, i.e., $M^K \geq pre(t)$. In case t is enabled in M^K, its firing leads to the marking $M^K - pre(t) + post(t)$. We denote $En(M^K)$ the set of transitions enabled for M^K, i.e., $En(M^K) = \{t \in T \mid M^K \geq pre(t)\}$. For $t \in En(M^K)$, we denote $CF(M^K, t)$ the set of transitions enabled in M^K but in conflict with t, i.e.,

$$CF(M^K, t) = \{t' \in En(M^K) \mid t' = t \vee \neg M^K \geq pre(t) + pre(t')\}$$

Let M'^K be the successor marking of M^K by t. We denote $Nw(M^K, t)$ the set of transitions enabled by firing t from M^K. The set $Nw(M^K, t)$ contains t, if t is enabled in M'^K, and also all transitions enabled in marking M^K but not enabled in the intermediate marking $M^K - pre(t)$, i.e.,

$$Nw(M^K, t) = \{t' \in En(M'^K) \mid t' = t \vee \neg M^K \geq pre(t) + pre(t')\}$$

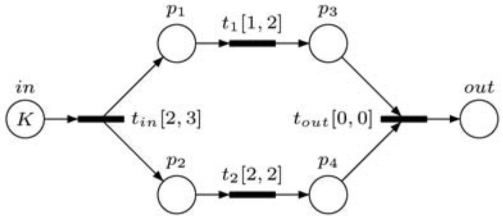


Figure 1. A time parametric Workflow net

Starting from the initial marking M_0^K, $\mathcal{N}^K$ evolves by firing transitions and time progressions. When a transition t becomes enabled, its firing interval is set to its static firing interval. Bounds of its interval decrease synchronously with time until it is fired or disabled by a conflicting firing. Transition t can fire, if the lower bound of its firing interval reaches 0. It must be fired immediately, without any additional delay, when the upper bound of its firing interval reaches 0, unless it is disabled by another firing. The firing of a transition takes no time.
The state of $\mathcal{N}^K$ is defined as a pair $s^K = (M^K, I^K)$, where M^K is a marking of $\mathcal{N}^K$ and I^K is a firing interval function, $I^K : En(M^K) \rightarrow INT_{R^+}$. The initial state of $\mathcal{N}^K$ is $s_0{}^K = (M_0^K, I_0^K)$, where $I_0^K(t) = Is(t)$, for all $t \in En(M_0^K)$.
Let $s^K = (M^K, I^K)$ and $s' = (M'^K, I'^K)$ be two states of $\mathcal{N}^K$, $dh \in \mathbb{R}^+$ be a nonnegative real number, $t \in T$ a transition and $\rightarrow$ the transition relation defined by:

- $s^K \overset{dh}{\rightarrow} s'^K$, also denoted $s^K + dh$, iff the state s'^K is reachable from state s^K by dh time units, i.e.,

$$\forall t \in En(M^K), \uparrow I^K(t) + dh \leq \uparrow Is(t), M^K = M'^K \text{ and}$$

$$\forall t' \in En(M'^K), I^K(t') = [Max(0, \downarrow I^K(t') - dh), \uparrow I^K(t') - dh]$$

- $s^K \overset{t}{\rightarrow} s'^K$ iff state s'^K is immediately reachable from state s^K by firing transition t, i.e.,

$$t \in En(M^K), \ \downarrow I^K(t) = 0,$$

$$M'^K = M^K - pre(t) + post(t), \text{and}$$

$$\forall t' \in En(M'^K), I'^K(t') = \begin{cases} Is(t') & \text{if } t' \in Nw(M^K, t) \\ I^K(t') & \text{otherwise} \end{cases}$$

The semantics of $\mathcal{N}^K$ is defined by the transition system $(S^K, \rightarrow, s_0^K)$, where S^K is the set of all states reachable from s_0^K by $\overset{*}{\rightarrow}$ (the reflexive and transitive closure of $\rightarrow$). Transition systems of $\mathcal{N}$ are those of $\mathcal{N}^K$ for $K \geq 1$.

3 Abstraction of the T-PWN state space

The reachability analysis of timed models (such as timed automata and time extensions of Petri nets) is generally based on abstraction, which preserves reachability properties. For timed models with fixed initial states, such an abstraction consists in grouping in the same node all states reachable by the same firing sequence independently of their firing times. The grouped states are then considered modulo some equivalence relation preserving properties of interest. All states within the same node share the same untimed information and the union of their time domains is represented by a set of atomic constraints[1] handled efficiently by means of a *Difference Bound Matrix* (DBM) [4]. Among the abstraction proposed in the literature, we consider here the *state class graph* method [5], for its advantage, over the others, to enjoy the finiteness property for all bounded time Petri nets. In other abstractions based on clocks, the finiteness is enforced using an over-approximation operation over DBM [8].

3.1 State class graph of $\mathcal{N}^K$

In the state class graph (SCG for short), each node, called state class, is defined by a marking and a formula characterizing the firing domains of states grouped in that node. For $\mathcal{N}^K$, a state class is the pair $\alpha^K = (M^K, F^K)$, where in F^K, each transition enabled in M^K is represented with a real-valued variable having the same name to ease notations. This variable represents the firing delay of the transition. The initial state class is represented by the pair $\alpha_0^K = (M_0^K, F_0^K)$, where $M_0^K = K \bullet in$ and $F_0^K = \bigwedge_{t \in En(M_0^K)} \downarrow Is(t) \leq t \leq \uparrow Is(t)$.

Let $\alpha^K = (M^K, F^K)$ be a state class and t_f a transition. α^K has a successor by t_f, denoted $succ(\alpha^K, t_f) \neq \emptyset$, iff $t_f \in En(M^K)$ and the following formula, denoted $F_{t_f}^K$, is consistent: $F^K \wedge (\bigwedge_{t \in En(M^K)} t_f \leq t)$.

It means that t_f can be fired before all other enabled transitions. Let $O(M^K, t_f) = En(M^K) - CF(M^K, t_f)$. If $succ(\alpha^K, t_f) \neq \emptyset$ then $succ(\alpha^K, t_f) = (M'^K, F'^K)$, where $M'^K = M^K - pre(t_f) + post(t_f)$ and $F'^K = ((F_{t_f}^K)_{[\bullet \leftarrow \bullet + t_f]})_{|O(M^K, t_f)} \wedge$

$$\bigwedge_{t \in Nw(M^K, t_f)} \downarrow Is(t) \leq t \leq \uparrow Is(t).$$

In words, F'^K is obtained from the firing condition $F_{t_f}^K$ of t_f from α^K by replacing each t, except t_f, with $t + t_f$; restricting the domain of the resulting formula to transitions of $O(M^K, t_f)$, and adding constraints of the newly enabled transitions. Let $\longrightarrow$ be the transition relation over state classes of $\mathcal{N}^K$ defined by: $\alpha^K \xrightarrow{t_f} \alpha'^K$ iff $succ(\alpha^K, t_f) \neq \emptyset \wedge \alpha'^K = succ(\alpha^K, t_f))$. Formally, the SCG of $\mathcal{N}^K$ is the structure $(\mathcal{C}^K, \longrightarrow, \alpha_0^K)$, where α_0^K is the initial state class of $\mathcal{N}^K$, $\mathcal{C}^K$ is the set of reachable state classes by $\xrightarrow{*}$.

This approach has been shown to produce finite SCG for all bounded TPN and then for bounded $\mathcal{N}^K$, while preserving markings and firing sequences (reachability properties) [5].

3.2 State class graph of $\mathcal{N}$

For the parametric verification purpose, we investigate here the possibility to superpose the state class graphs of $\mathcal{N}^K$, for $K \geq 1$, while merging paths supporting the same firing sequences.

Let us first introduce, by means of an example, the challenges of this superposition. Consider the model at Figure 1 and its set of initial state classes: $\alpha_0 = \{\alpha_0^K = (K \bullet in, 2 \leq t_{in} \leq 3) | K \geq 1\}$. Transition t_{in} is firable from all state classes of α_0. Its firing leads to the set of state classes $\alpha_1 = \{\alpha_1^K = (M_1^K, F_1^K) | \exists \alpha_0^K \in \alpha_0, \alpha_0^K \xrightarrow{t_{in}} \alpha_1^K\}$, with $M_1^K = (K-1) \bullet in + 1 \bullet p_1 + 1 \bullet p_2$, for $K \geq 1$.

The set of transitions enabled in marking M_1^K depends on the value of K: $\{t_{in}, t_1, t_2\}$ for $K > 1$, and $\{t_1, t_2\}$ for $K = 1$. Its corresponding firing domain depends consequently on the value of K: $2 \leq t_{in} \leq 3 \wedge 1 \leq t_1 \leq 2 \wedge t_2 = 2$, for $K > 1$ and $1 \leq t_1 \leq 2 \wedge t_2 = 2$ for $K = 1$. From α_1, the transition t_{in} is not enabled for $K = 1$ but enabled and firable for $K > 1$; t_1 and t_2 are enabled and firable for $K \geq 1$. The three transitions are then firable from α_1. All state classes reached from state classes of α_1 by firing the same transition are grouped in the same set. For instance, all successors of state classes of α_1 by t_2 are grouped in the same set α_2, i.e., $\alpha_2 = \{\alpha_2^K | \exists \alpha_1^K \in \alpha_1, \alpha_1^K \xrightarrow{t_2} \alpha_2^K\}$. From α_1, t_2 is firable for $K \geq 1$. Then, $\alpha_2 = \{\alpha_2^K = ((K-1) \bullet in + 1 \bullet p_1 + 1 \bullet p_4, 0 \leq t_{in} \leq 1 \wedge t_1 = 0) | K > 1\} \cup \{\alpha_2^1 = (1 \bullet p_1 + 1 \bullet p_4, t_1 = 0)\}$.

To apply this abstraction, it is necessary to handle symbolically and concisely such sets of state classes. We first extend the notions of enabledness and firability to the sets of state classes and then define the state class graph of $\mathcal{N}$.

Definition 2 *Let α be a set of state classes, $\mathcal{M}$ its set of markings and $t \in T$ a transition.*

- *t is enabled in α (denoted $t \in EN(\mathcal{M})$) iff $\exists M^K \in \mathcal{M}, t \in En(M^K)$.*
- *t is firable in α (denoted $Succ(\alpha, t) \neq \emptyset$) iff $\exists \alpha^K \in \alpha, succ(\alpha^K, t) \neq \emptyset$.*
- *If $Succ(\alpha, t) \neq \emptyset$ then $Succ(\alpha, t) = \{succ(\alpha^K, t) | \alpha^K \in \alpha \wedge succ(\alpha^K, t) \neq \emptyset\}$*
- *Let $\twoheadrightarrow$ be the transition relation over sets of state classes defined by: $\alpha \overset{t}{\twoheadrightarrow} Succ(\alpha, t)$ iff $Succ(\alpha, t) \neq \emptyset$. The state class graph of $\mathcal{N}$ is the structure $(\mathcal{C}, \twoheadrightarrow, \alpha_0)$, where α_0 is the set of its initial state classes and $\mathcal{C} = \{\alpha | \alpha_0 \overset{*}{\twoheadrightarrow} \alpha\}$.*

3.3 Abstract state classes

We fix a T-PWN $\mathcal{N}$, $\alpha_0 = \{\alpha_0^K | K \geq 1\}$ its set of initial state classes, $\sigma \in T^*$ a transition sequence firable from α_0, $\alpha = \{\alpha^K | \exists \alpha_0^K \in \alpha_0, \alpha_0^K \xrightarrow{\sigma} \alpha^K\}$ the set of state classes reachable from α_0 by σ, and $\mathcal{M}$ the set of markings of α.

Note that markings of $\mathcal{M}$ are reached from initial markings $K \bullet in, K \geq 1$ by firing the same sequence σ. Therefore, they differ only in the number of tokens in place in. The set $\mathcal{M}$ is then a totally ordered set w.r.t. $\leq$. We denote M^a and M^b the smallest and the biggest markings in $\mathcal{M}$, a and b are the smallest and the biggest values of K in α.

[1] An atomic constraint is of the form $x \prec c, -x \prec c$ or $x - y \prec c$, where x, y are real-valued variables, $\prec \in \{<, =, \leq, \geq, >\}$ and c is a rational number.

Moreover, for each marking M^K of $\mathcal{M}$, there is exactly one state class in α with marking M^K (i.e., $Cardinal(\mathcal{M}) = Cardinal(\alpha)$).

Definition 3 • *The abstract state class of α is defined by the quadruplet $\beta = (a, b, m, F)$, where:*
- a and b are the smallest and the biggest values of K in α, respectively,
- m is the smallest marking in α (the triplet (a, b, m) is called the abstract marking of α), and
- F is an over-approximation of the union of firing domains of state classes of α, represented as a conjunction of atomic constraints over transitions enabled in the biggest marking in α.

- *The abstract state class of α_0 is $\beta_0 = (0, \infty, 0, F_0)$, where $F_0 = \bigwedge_{t \in En(M_0^\infty)} \downarrow Is(t) \leq t \leq \uparrow Is(t)$.*
  Note that $En(M_0^\infty) \subseteq T$.
- *An abstract state class β of α is sound and complete iff*
  $$\alpha = \{\alpha^K = (M^K, F^K) \mid a \leq K \leq b, M^K = (K - a) \bullet in + m \text{ and } F^K = (F \wedge \bigwedge_{t \in En(M^K)} t \geq 0)_{|En(M^K)}\}.$$
  Intuitively, it means that state classes of α can be retrieved from β.

We establish, in Lemma 1, some sufficient conditions for enabledness and firability of transitions in α.

Lemma 1 *Let $\beta = (a, b, m, F)$ be a sound and complete abstract state class of α, $t_f \in T$ a transition. Then:*

1. *$t_f \in EN(\mathcal{M})$ iff $t_f \in En(M^b)$.*
2. *If $t_f \in EN(\mathcal{M})$ then the smallest marking in $\mathcal{M}$ s.t. t_f is enabled, is $M^{a_f} = pre(t_f)(in) \bullet in + m$. The corresponding value a_f of K is $a_f = a + pre(t_f)(in)$.*
3. *$Succ(\alpha, t_f) \neq \emptyset$ iff $t_f \in En(M^b)$ and $F^{a_f} \wedge \bigwedge_{t \in En(M^{a_f})} t_f \leq t$.*
4. *If $Succ(\alpha, t_f) \neq \emptyset$ then the smallest marking in $\mathcal{M}$ s.t. t_f is firable, is M^{a_f}.*
5. *Let $bf(\beta, t_f) = \{t \in En(M^b) | F \wedge t_f \leq t$ is not consistent$\}$ be the set of transitions which must fire before t_f from all markings of α.*
   If $Succ(\alpha, t_f) \neq \emptyset$ then the biggest marking in $\mathcal{M}$ s.t. t_f is firable, is $M^{b_f} = (b_f - a) \bullet in + m$, where:
   $$b_f = \begin{cases} a + \underset{t \in bf(\beta, t_f)}{Min} pre(t)(in) - 1 & \text{if } bf(\beta, t_f) \neq \emptyset \\ b & \text{otherwise.} \end{cases}$$
6. *If $Succ(\alpha, t_f) \neq \emptyset$ then: $Succ(\alpha, t_f) = \{succ(\alpha^K, t_f) | \alpha^K \in \alpha \wedge a_f \leq K \leq b_f\}$.*

Proof 1 *1.: The proof is immediate from the fact that $En(M^a) \subseteq En(M^K) \subseteq En(M^b)$, for $a \leq K \leq b$.*
2.: The markings of $\mathcal{M}$ differ only in the number of tokens in place in. The smallest marking in α s.t. t_f is enabled is the marking where the number of tokens in place in is $pre(t_f)(in)$ (i.e., $M^{a_f} = pre(t_f)(in) \bullet in + m$). Then, $a_f = a + pre(t_f)(in)$.
3-4.: By assumption β is sound and complete. The firing domain F^{a_f} is then the restriction, to transitions of $En(M^{a_f})$, of $F \wedge \bigwedge_{t \in En(M^{a_f})} t \geq 0$. Since $\forall M^K \in \mathcal{M}$ s.t. $M^{a_f} \leq M^K$, the domain of $F^K_{|En(M^{a_f})}$ is included in the domain of F^{a_f}, it follows that t_f is firable from states of α iff there is a valuation in the domain of F^{a_f} s.t. $t_f \leq t$ for $t \in En(M^{a_f})$. Moreover, the smallest marking from which t_f is firable is M^{a_f}.
5.: If $F \wedge t_f \leq t$ is not consistent then for $a \leq K \leq b$ s.t. $t \in En(M^K)$, $F^K \wedge t_f \leq t$ is not consistent (because[2] $Dom(F^K) \subseteq Dom(F_{|En(M^K)})$). It means that t must fire before t_f from α. The biggest marking from which t_f is firable is the biggest marking s.t. its set of enabled transitions does not contain any transition of $bf(\alpha.t_f)$. If $bf(\alpha, t_f) = \emptyset$ then $M^{b_f} = M^b$ and $b_f = b$. If $bf(\alpha, t_f) \neq \emptyset$ then $\forall t \in bf(\alpha, t_f), in \in^\bullet t$ (markings of $\mathcal{M}$ differ only in the number of tokens in place in). We must have at most $\underset{t \in bf(\beta, t_f)}{Min} pre(t) - 1$ in place in, to prevent enabledness of transitions of $bf(\beta, t_f)$. Then:
$M^{b_f} = (\underset{t \in bf(\beta, t_f)}{Min} pre(t) - 1) \bullet in + m$ *and*
$b_f = a + (\underset{t \in bf(\beta, t_f)}{Min} pre(t) - 1)$.
6.: The firing domain of M^K is $F^K = (F \wedge \bigwedge_{t \in En(M^K)} t \geq 0)_{|En(M^K)}$.
For $a_f \leq K \leq b_f$, it holds that $En(M^K) \subseteq En(M^{b_f})$ and then $Dom(F^{b_f}_{|En(M^K)}) \subseteq Dom(F^K)$. Then, if t_f is firable from M^{a_f} and M^{b_f}, it is also firable from each marking M^K between M^{a_f} and M^{b_f}.

3.4 Computing abstract state classes

Definition 4 *Let $\beta = (a, b, m, F)$ be the abstract state class of α and t_f a transition firable from α. The abstract state class of $\alpha' = Succ(\alpha, t_f)$ is $\beta' = (a', b', m', F')$ where:*

1. *$a' = a_f, b' = b_f$ and $m' = M^{a_f} - pre(t_f) + post(t_f)$.*
2. $F' = ((F \wedge \bigwedge_{t \in En(M^{a'})} 0 \leq t_f \leq t)_{[\bullet \leftarrow \bullet + t_f]})_{|O(M^{b'}, t_f)} \wedge \bigwedge_{t \in Nw(M^{b'}, t_f)} \downarrow Is(t) \leq t \leq \uparrow Is(t)$.

Theorem 1 *If β is sound and complete then β' is sound and complete too.*

Proof 2 *According to Lemma 1, $\alpha' = Succ(\alpha, t_f) = \{succ(\alpha^K, t_f) = (M'^K, F'^K) | \alpha^K \in \alpha \wedge a_f \leq K \leq b_f\}$. With $a' = a_f$ and $b' = b_f$, β' is sound and complete iff for $a' \leq K \leq b'$,*
1) $M'^K = (K - a') \bullet in + m'$ and
2) $F'^K = (F' \wedge \bigwedge_{t \in En(M'^K)} t \geq 0)_{|En(M'^K)}$.
By assumption, β is sound and complete. Then, state classes of α are $\alpha^K = (M^K, F^K)$, for $a \leq K \leq b$, with:
$M^K = (K - a) \bullet in + m$ and $F^K = (F \wedge \bigwedge_{t \in En(M^K)} 0 \leq t)_{|En(M^K)}$.
1) $M'^K = M^K - pre(t_f) + post(t_f) =$
$$(K - a - pre(t_f)(in)) \bullet in + m',$$
where $m' = m - (pre(t_f) - pre(t_f)(in) \bullet in) + post(t_f)$. Then, $M'^K = (K - a') \bullet in + m'$ (see Lemma 1, $a' = a_f = a + pre(t_f)(in)$).
2) By definition, F'^K is:
$((F^K_{t_f})_{[\bullet \leftarrow \bullet + t_f]})_{|O(M^K, t_f)} \wedge \bigwedge_{t \in Nw(M^K, t_f)} \downarrow Is(t) \leq t \leq \uparrow Is(t)$,

[2] We denote $Dom(F)$ the value domain represented by the formula F.

where $F^K_{t_f} = F^K \wedge \bigwedge_{t \in En(M^K)} t_f \leq t$.

By assumption, β *is sound and complete. Then:*
$F^K = (F \wedge \bigwedge_{t \in En(M^K)} 0 \leq t)_{|En(M^K)}$ *and*
$F^K_{t_f} = (F \wedge \bigwedge_{t \in En(M^K)} 0 \leq t_f \leq t)_{|En(M^K)}$.

By replacing, in $F^K_{t_f}$, t *with* $t + t_f$, *the constraint* $t_f \leq t$ *becomes* $0 \leq t$. *Now, with the fact that* $O(M^K, t_f) \subseteq En(M^K)$ *and* $En(M^{a'}) \subseteq En(M^K)$, *we can rewrite* $((F^K_{t_f})_{[\bullet \leftarrow \bullet + t_f]})_{|O(M^K, t_f)}$ *as follows:*

$$((F \wedge \bigwedge_{t \in En(M^{a'})} 0 \leq t_f \leq t)_{[\bullet \leftarrow \bullet + t_f]} \wedge \bigwedge_{t \in En(M^K)} 0 \leq t)_{|O(M^K, t_f)}.$$

Then: (i) $F'^K =$

$$((F \wedge \bigwedge_{t \in En(M^{a'})} 0 \leq t_f \leq t)_{[\bullet \leftarrow \bullet + t_f]} \wedge \bigwedge_{t \in En(M^K)} 0 \leq t)_{|O(M^K, t_f)}$$

$$\wedge \bigwedge_{t \in Nw(M^K, t_f)} \downarrow Is(t) \leq t \leq \uparrow Is(t)$$

Let us now develop the expression $(F' \wedge \bigwedge_{t \in En(M'^K)} 0 \leq t)_{|En(M'^K)}$:

(ii) $(((F \wedge \bigwedge_{t \in En(M^{a'})} 0 \leq t_f \leq t)_{[\bullet \leftarrow \bullet + t_f]})_{|O(M^{b'}, t_f)} \wedge \bigwedge_{t \in Nw(M^{b'}, t_f)} \downarrow Is(t) \leq t \leq \uparrow Is(t) \wedge \bigwedge_{t \in En(M'^K)} 0 \leq t)_{|En(M'^K)}$.

Using $O(M^{b'}, t_f) \cap Nw(M^{b'}, t_f) = \emptyset$ *and* $En(M'^K) = O(M^K, t_f) \cup Nw(M^K, t_f)$, *expression (ii) can be rewritten as follows:*

$$(((F \wedge \bigwedge_{t \in En(M^{a'})} 0 \leq t_f \leq t)_{[\bullet \leftarrow \bullet + t_f]})_{|O(M^{b'}, t_f) \cap En(M'^K)} \wedge$$

$$\bigwedge_{t \in Nw(M^{b'}, t_f) \cap En(M'^K)} \downarrow Is(t) \leq t \leq \uparrow Is(t) \wedge$$

$$\bigwedge_{t \in En(M'^K)} 0 \leq t)_{|En(M'^K)}.$$

To complete the proof, it suffices to show that:
a) $Nw(M^{b'}, t_f) \cap En(M'^K) = Nw(M^K, t_f)$ *and*
b) $O(M^{b'}, t_f) \cap En(M'^K) = O(M^K, t_f)$.

a) Suppose that $t \in Nw(M^{b'}, t_f) \cap En(M'^K)$ *and* $t \notin Nw(M^K, t_f)$. *It follows that* $pre(t) \leq M^K - pre(t_f) \leq M^{b'} - pre(t_f)$. *Therefore,* $t \notin Nw(M^{b'}, t_f)$, *which contradicts the assumption.*

Suppose now that $t \notin Nw(M^{b'}, t_f) \cap En(M'^K)$ *and* $t \in Nw(M^K, t_f)$. *It follows that* $t \neq t_f$, $pre(t) \leq M^{b'} - pre(t_f)$ *and* $\exists p \in P, pre(t)(p) > M^K(p) - pre(t_f)(p)$. *As* $\forall p \in P - \{in\}, M^K(p) = M^{b'}(p)$ *and* in *is an input place, it follows that* $pre(t)(in) > M^K(in) - pre(t_f)(in) = M'^K(in)$. *Then,* $t \notin En(M'^K)$, *which is in contradiction with the assumption.*

b) We have $En(M'^K) = O(M^K, t_f) \cup Nw(M^K, t_f)$ *then: (iii)* $En(M'^K) \cap O(M^{b'}, t_f) =$

$$O(M^K, t_f) \cap O(M^{b'}, t_f) \cup Nw(M^K, t_f) \cap O(M^{b'}, t_f).$$

Since $M^K \leq M^{b'}$, $En(M^K) \subseteq En(M^{b'})$, $CF(M^{b'}, t_f) \subseteq CF(M^K, t_f)$, *it follows that* $O(M^K, t_f) \subseteq O(M^{b'}, t_f)$.

With $Nw(M^K, t_f) \cap O(M^{b'}, t_f) \subseteq Nw(M^{b'}, t_f) \cap O(M^{b'}, t_f) = \emptyset$, *equality (iii) above can be rewritten:* $En(M'^K) \cap O(M^{b'}, t_f) = O(M^K, t_f)$. *Finally, using (i), (ii), a) and b), we can state that* $F'^K = (F' \wedge \bigwedge_{t \in En(M'^K)} t > 0)_{|En(M'^K)}$.

Let us redefine equality of sets of state classes by means of their abstractions.

Lemma 2 *Let* α *and* α' *be two sets of state classes,* $\beta = (a, b, m, F)$ *and* $\beta = (a', b', m', F')$ *their sound and complete abstract state classes, respectively.* $\alpha = \alpha'$ *iff*

1) $b' - a' = b - a$, *2)* $m' = m$ *and 3)* $\forall K$ *s.t.* $a \leq K \leq c$, $(F \wedge \bigwedge_{t \in En(M^K)} t > 0)_{|En(M^K)} = (F' \wedge \bigwedge_{t \in En(M^K)} t > 0)_{|En(M^K)}$, *where c is the smallest integer s.t.* $En(M^c) = En(M^{c+1})$.

Proof 3 $\mathcal{M} = \{(K - a) \bullet in + m | a \leq K \leq b\}$, $\mathcal{M}' = \{(K' - a') \bullet in + m' | a' \leq K' \leq b'\}$. *Then, if* $m = m'$ *and* $b - a = b' - a'$ *then* $\mathcal{M} = \mathcal{M}'$. *The rest of the proof is straightforward from the fact that* β *and* β' *are sound and complete abstract state classes of* α *and* α', *respectively.*

Theorem 2 • *The abstraction, proposed here, preserves reachability properties of* $\mathcal{N}$.
• *The graph obtained is finite iff the model has a finite number of abstract markings.*

Proof 4 • *The initial abstract state class is sound and complete, according to Theorem 1, all its successor abstract state classes are sound and complete. Therefore, the graph of abstract state classes is isomorph to the state class graph of* $\mathcal{N}$. *So, it preserves state classes and firing sequences of* $\mathcal{N}$ *(reachability properties).*
• *Each abstract marking may have an infinite number of markings but has a finite number of sets of enabled transitions. For each set of enabled transitions, there is a finite number of non equivalent formulas [5]. If the model has a finite number of abstract markings then it has a finite number of reachable sets of state classes.*

The finiteness of state class graph is known to be undecidable for time Petri nets and also for T-WPN. However, from the practical point of view, there are some useful sufficient conditions which allow deciding infiniteness [5].

One of the main required property in Workflow is the K-soundness [2] defined as follows: A T-PWN is K-sound iff $\mathcal{N}^K$ satisfies:

1) For every marking M^K reachable from initial marking $K \bullet in$, there exists a firing sequence leading from marking M^K to final marking $K \bullet out$.
2) Final Marking is the only marking reachable from initial marking with at least K tokens in place out.
3) There are no dead transitions in $\mathcal{N}^K$ (i.e., $\forall t \in T, \exists \alpha^K \in \mathcal{C}^K, succ(\alpha^K, t) \neq \emptyset$).

In [6], the author has identified subclasses[3] of time Workflow nets ($\mathcal{N}^1$) where the verification of 1-soundness property is reduced to the problem of 1-soundness for the underlying untimed Workflow net and then is decidable [3, 13]. The extension of this result to the K-soundness property is straightforward for the same subclasses of T-PWN. In [10], the authors have addressed the soundness property for a subclass of one-safe time Workflow nets and established some sufficient conditions based on the structure of the model. As a future work, we will investigate further the verification of K-soundness property for other classes of T-PWN using the abstraction proposed here and the extension of results shown in [2, 12].

[3] Time Workflow nets s.t. $\forall t \in T, \downarrow Is(t) = 0 \vee \forall t \in T, \uparrow Is(t) = \infty$.

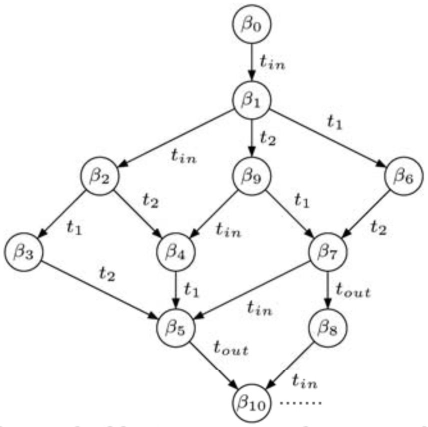


Figure 2. Abstract state class graph of the T-PWN at Figure 1

As an example, we report at Figure 2 and Table 1 the abstract state class graph of the model shown at Figure 1. State classes β_1 and β_{10} are different but equivalent (have the same firing sequences) because the only difference resides in the number of tokens in the output place out. So, they can be grouped in the same node without altering reachability properties. More generally, we can state that the quotient abstract state class graph w.r.t. this relation of equivalence is finite iff all places, except in and out, are bounded.
Consider now the abstract state class graph resulting from grouping in the same node β_1 and $\beta 10$. When we reach a marking where the K tokens of place in are consumed, the transition t_{in} will not be fired anymore. If we eliminate all arcs labeled t_{in}, except the arc $(\beta_0, t_{in}, \beta_1)$, we obtain an acyclic graph with β_8 as a final node and no dead transition. In β_8, only the output place out is marked, so, we can conclude that the T-PWN at Figure 1 satisfies the K-soundness property.

Note that we can extract from this compact representation of state class graphs of $\mathcal{N}$, the SCG of each $\mathcal{N}^K$. It suffices to eliminate each abstract state class (a, b, m, F) s.t. $K \notin [a, b]$.

β_0	$(1, \infty, 0, 2 \leq t_{in} \leq 3)$
β_1	$(1, \infty, p_1 + p_2, 2 \leq t_{in} \leq 3 \wedge 1 \leq t_1 \leq 2 \wedge t_2 = 2)$
β_2	$(2, \infty, 2p_1 + 2p_2, 2 \leq t_{in} \leq 3 \wedge t_1 = t_2 = 0)$
β_3	$(2, \infty, p_1 + 2p_2 + p_3, 2 \leq t_{in} \leq 3 \wedge 1 \leq t_1 \leq 2 \wedge t_2 = 0)$
β_4	$(2, \infty, 2p_1 + p_2 + p_4, 2 \leq t_{in} \leq 3 \wedge t_1 = 0 \wedge t_2 = 2)$
β_5	$(2, \infty, p_1 + p_2 + p_3 + p_4,$ $2 \leq t_{in} \leq 3 \wedge 1 \leq t_1 \leq 2 \wedge t_2 = 2 \wedge t_{out} = 0)$
β_6	$(1, \infty, p_2 + p_3, 1 \leq t_{in} \leq 2 \wedge 0 \leq t_1 \leq 1)$
β_7	$(1, \infty, p_3 + p_4, 0 \leq t_{in} \leq 1 \wedge t_{out} = 0)$
β_8	$(1, \infty, out, 0 \leq t_{in} \leq 1)$
β_9	$(1, \infty, p_1 + p_4, 0 \leq t_{in} \leq 1 \wedge t_1 = 0)$
β_{10}	$(2, \infty, p_1 + p_2 + out, 2 \leq t_{in} \leq 3 \wedge 1 \leq t_1 \leq 2 \wedge t_2 = 2)$

Table 1. Abstract state classes of T-WPN at Figure 1

4 Conclusion

We have proposed a reachability analysis approach for time Workflow nets with parametric initial marking (T-PWN). Our approach consists in superposing state classes graphs of the T-PWN. All state classes reached by the same firing sequence from all initial markings are grouped and handled symbolically with lesser resources. We have shown its correctness w.r.t. reachability properties and established a necessary and sufficient condition for its finiteness. The results established here are still valid for T-PWN with initial resources provided that their initial numbers are fixed.
As immediate perspective, we will investigate further the verification of K-soundness and quantitative time properties for T-PWN.

References

[1] Abdulla P.A.: Forcing Monotonicity in Parameterized Verification: From Multisets to Words, in Proc. of SOFSEM 2010: Theory and Practice of Computer Science, LNCS no 5901, pp 1-15, 2010.

[2] Barkaoui K. and Ben Ayed R.: Uniform Verification of Workflow Soundness, in Transactions of the Institute of Measurement and Control Journal, vol. 31, pp. 1-16, 2010 SAGE Publisher.

[3] Barkaoui K. and Petrucci L.: Structural Analysis of Workflow Nets with Shared Resources, Proc. Workflow Management: Net-based Concepts, Models, Techniques and Tools (WFM98), Lisbon, Portugal, June 1998, vol 98/7 of Computing Science Reports, Eidhoven University of Technology, pp. 82-95, 1998.

[4] Bengtsson J.: Clocks, DBMs and States in Timed Systems, *PhD thesis, Dept. of Information Technology, Uppsala University*, 2002.

[5] Berthomieu B. and Diaz M.: Modeling and verification of time dependent systems using time Petri nets, IEEE Transactions on Software Engineering, 17(3), 1991.

[6] Camerzan I.: On Soundness for Time Workflow Nets, Computer Science Journal of Moldova, vol. 15, no. 1(43), pp 74-87, 2007.

[7] Finkel A., Raskin J.f., Samuelides M. and Van Begin L.: Monotonic Extensions of Petri Nets: Forward and Backward Search Revisited, In proc. INFINITY'2002, pp 1-22, 2002.

[8] Hadjidj, R. and Boucheneb, H.: On-the-Fly TCTL Model-Checking for Time Petri Nets. Theoretical Computer Science, 410(42), p. 4241-4261, 2009.

[9] van Hee K., Sidorova N. and Voorhoeve M.: Generalised Soundness of Workflow Nets is Decidable, ICATPN 04, LNCS no 3099, 2004.

[10] Ling S. and Schmidt H.: Time Petri Nets for Workflow Modelling and Analysis, IEEE International Conference onSystems, Man, and Cybernetics, pp 3039 - 3044 vol.4, 2000.

[11] Merlin P.M.: A study of the recoverability of computing systems, Department of Information and Computer Science, University of California, Irvine CA, 1974.

[12] Tiplea F.L. and Marinescu D.C.: Structural soundness of workflow nets is decidable, Information Processing Letters 96, 2005, 54-58.

[13] van der Aalst W.M.P. et al.:Soundness of Workflow Nets: Classification, Decidability, and Analysis, Formal Aspects of Computing, Vol. 23, Issue: 3, pp 333-363, 2010.

[14] van der Aalst W.M.P.: Verification of Workflow nets, ICATPN 97, LNCS 1248, 1997.

Resource Modeling and Analysis for Workflows: A Petri Net Approach

Jiacun Wang
Monmouth University
W. Long Branch, NJ 07764, USA
jwang@monmouth.edu

Demin Li
Donghua University
Shanghai, China
deminli@dhu.edu.cn

Abstract-Petri nets are a powerful formalism in modeling workflows. A workflow determines the flow of work according to pre-defined business process definitions. In many situations, business processes are constrained by scarce resources. The lack of resources can cause contention, the need for some tasks to wait for others to complete, and the slowing down of the accomplishment of larger goals. In our previous work, a resource-constrained workflow model is introduced and a resource requirement analysis approach is presented for emergency response workflows, in which support of on-the-fly workflow change is critical [6]. In this paper, we propose a Petri net based approach for recourse requirements analysis, which can be used for more general purposes. The concept of resource-oriented workflow nets (ROWN) is introduced and the transition firing rules of ROWN are presented. Resource requirements for general workflow can be done through reachability analysis. For a class of well-structured workflows, an efficient resource analysis algorithm is developed.

***Keywords*—workflows, workflow nets, Petri nets, resource requirements analysis.**

I. INTRODUCTION

A workflow consists of processes and acti vities, which are represented by well-defined tasks. T he entities t hat execute these tasks are h umans, application programs or machines. These tasks are r elated and dep endent on one another based on business policies and rules.

This paper focuses on the resource requirements of a workflow. In many situations, business processes are constrained by scarce resources. The lack of resources can cause contention, the need for some tasks to wait for others to complete, and the slowing down of the accomplishment of larger goals. This is particularly true in an emergency response system where large quantities of resources, including emergency responders, ambulances, medical care personnel, fire trucks, medications, food, clothing, etc., are required. Often potential delays can be avoided or reduced by using resource analysis to identify ways in which tasks can be executed in parallel, in the most efficient way. A workflow with resource usage defined can help keep track of resource availability, disable the paths that are not executable, and present all executable paths, thus allowing the automatic selection of feasible path for system execution.

Petri nets are a powerful tool to m odel and anal yze resources-constrainted systems. Some excellent research results on resource allocation and deadlock avoidance are published. For example, workflow nets have been identified and widely used as a solid m odel of business processes [1][3][5]. Resource-constrained workflow nets are introduced and discussed in [3][4]. In them, the authors study extensions of workflow nets in which processes must share some global resources. A resource belongs to a t ype. One place is used for one type, where the resource is located when it is free. There are static and dynamic places. Static places are for resources that will be shared by cases. An important assumption in resource-constrained workflow nets is all re sources are du rable: they cannot be created or destroyed.

Resource-oriented Petri nets are introduced in [9][10], which deal with finite capacity Petri nets. In a resou rce-oriented Petri net, a t ransition is enab led if and on ly if tokens, which represent resource, in a place won't exceed the predefined capacity of the place if the transition fires, as well as there are enough tokens in each input place.

Both resource-constrained workflow nets and resource-oriented Petri nets only deal with durable resources, which are claimed and relea sed during the workflow execution but cannot be created and d estroyed [3]. However, there are a lot of workflows where task execution consumes and/or produces resources. For exa mple, in a incident response workflow, some resources (e.g. medication) can be consumed, and som e resources (e.g. fire trucks) can be replenished during emergency response.

The paper presents a P etri net based ap proach for resource requirements analysis that applies to du rable resources and n on-durable resources. The approach is based on our previous work on workflow modeling [6][7][8], in which a W IFA model is develo ped for emergency response workflows.

The paper is organized as f ollows: Section II briefly introduces the resource-oriented workflow model. Section III defines well-nested workflow and s hows how it can be constructed recursively, and then proposes an efficient resource requirement analysis algorithm. Finally, Section IV presents conclusions and ideas f or the continuation of the work.

II. Resource-Oriented Workflow Model

A workflow is composed of *tasks* that execute in a specific order. A task is an activity or an event. We assume a task is atomic. When it gets started, it is guaranteed to finish. An important implication of this assumption is all required resources to finish the task will be held by the task until it is done.

Workflows are *case-based*. Every piece of work is executed for a specific case. For example, an incident is a case for an incident response workflow. A patient visit is a case for the emergency department workflow.

Workflow execution requires *resources*. Workflows are executed by humans or machines, which are durable resources. When Petri nets are used to model a workflow, these kinds of resources can be specified using tokens. Besides workflow executors, executing a task in a workflow may consume/occupy other resources and when a task execution is finished some of these resources may be released or some new resources may be produced. Petri net tokens are not suitable to model this class of resources.

A. Task Modeling

A task can be modeled with a transition in a Petri net. However, in order to catch resources in use when a task is in execution and released when the task is done, we use two sequential transitions to model a task, one modeling the beginning of the task execution and the other the end of the task execution.

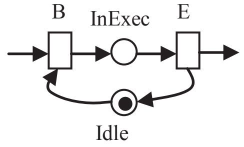


Fig. 1 Petri model of a task.

As shown in Fig. 1 (a), transition *B* represents the beginning of a task execution; *E* represents the end of the task execution. Place *InEx* represents the task is in execution, while *Idle* represents the task is idle. From reachability analysis perspective, Fig. 1(a) can be reduced to a single transition, which represents the entire task execution as a single logic unit. If there is no resource involved in a task execution, then we don't need to model it using two transitions.

B. Resource Modeling

We assume there are *n* types of resources in a system, and the quantity of each type of resource can be represented by a non-negative real number. Hence, resources are described by $S = (r_1, r_2, \ldots, r_n)$, where each r_i is a non-negative real number.

A task may hold or consume particular resources during execution and release or produce particular resources once execution is completed. We use R^+ to describe the resources consumed and/or held when executing a task and use R^- to describe resources produced and/or released after task execution is finished, where $r_i^+ \geq 0$ and $r_i^- \leq 0$ and $i = 1, 2, \ldots n$.

Resource modeling is based on the two-transition model. More specifically, for a task TS_k, $R^+(TS_k)$ is associated with transition B_k and $R^-(TS_k)$ is associated with transition E_k.

C. Resource-Oriented Workflow Nets (ROWN)

In [1], a Petri net that models a workflow process is called a *workflow net*. A workflow net is a Petri net that satisfies two requirements. First, it has one source place and one sink place. A token in the source place corresponds to a case needs to be handled. A token in the output place corresponds to a case that has already been handled. Secondly, there are no dangling transitions or places.

Formally, a Petri net is $PN = (P, T, I, O, M_0)$ if and only if

1) PN has two special places: *i* and *o*. Place *i* is a source place: $*i = \phi$; Place *o* is a source place: $*i = \phi$,
2) If we add a transition *t* to PN that connects place *o* with *i*, then the resulting Petri net is strongly connected.

A resource-constraint workflow net (ROWN) is workflow net in which a task whose execution requires resources is represented in two sequential transitions: a transition representing the start of the task and with resources consumption defined on it, and a transition representing the end of the task with resources production/release defined on it. Mathematically, it is defined as a 7-tupe: $ROWN = (P, T, R, I, O, M_0, S_0)$, in which (T, R) is a pair: for each $T_k \in T$, there is an $R_k \in R$ that specifies resource change associate with the firing of transition T_k. S_0 represents the initially available resources.

D. Transition Firing

A transition t_k is enabled under state (M_i, S_i) if and only if

$M_i \geq I(t_k)$, (1)

$S_i \geq R(t_k)$ (2)

Condition (1) stands for *control-ready*, while condition 2 stands for *resource-ready*. Notice that $R(t_k)$ is $R^+(t_k)$ if t_k represents the start of a task; it is $R^-(t_k)$ if t_k represents the end of a task; it is a 0 vector if the task represented by t_k does not involve any resource changes. The transition's enabledness is affected by resource availability only if it represents the start of a task.

After an enabled transition t_k fires, the new state is determined by

$M_j = M_i + O(t_k) - I(t_k)$, (3)

$S_j = S_i - R(t_k)$. (4)

Eq. (3) is exactly the same as it is for regular Petri nets. Eq. (4) reflects resource change after firing a transition: if the transition represents the start of a task, $R(t_k)$ is a positive vector and available resources are decreased . If the transition represents the end of a task, $R(t_k)$ is negative and available resources are increased. If no resource involved, then available resources remain unchanged.

Based on the transition firing rule reachability analysis can be performed, which explores all possible states and execution paths, reveals possible deadlocks due to resource contention.

E. An Example

Fig. 2 shows an ROWN with four tasks. Assume three types of resources are involved in the workflow execution and their initial quantities are $S_0 = (30, 25, 20)$. Resource changes by task execution are specified as follows:

$R(B_1) = R^+(TS_1) = (3, 8, 5)$
$R(E_1) = R^-(TS_1) = (-3, -2, -5)$

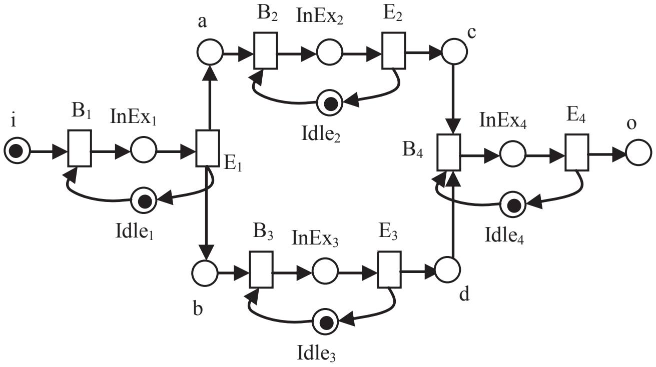


Fig. 2 An RCWN of 4 tasks.

$R(B_2) = R^+(TS_2) = (5, 0, 10)$
$R(E_2) = R^-(TS_2) = (0, 0, -5)$
$R(B_3) = R^+(TS_3) = (15, 10, 5)$
$R(E_3) = R^-(TS_3) = (-5, -2, -5)$
$R(B_4) = R^+(TS_4) = (0, 5, 15)$
$R(E_4) = R^-(TS_4) = (0, -5, -5)$

At the initial state, B_1 is the only transition that is enabled. After it fires, $S_1 = S_0 - R(B_1) = (27, 17, 15)$. Then E_1 is the only transition that is enabled. After E_1 is fired, $S_2 = S_1 - R(E_1) = (30, 12, 20)$. At state (M_2, S_2), both B_2 and B_3 are enabled. If B_2 fires, $S_3 = S_2 - R(B_2) = (25, 1 2, 10)$. Then if B_3 fires, $S_4 = S_3 - R(B_3) = (10, 2, 5)$. We can continue this process until we get to a state th at no transitions are enabled.

III. Resource Requirement Analysis

After a workflow is built and the resource consumption and production are defined for all tasks, the resource requirement for executing the workflow can be formally analyzed. We are interested in the *maximum resource consumption* (MRC), which is defined as the minimum amount of resources that if satisfied, the workflow can be executed along any possible path till finish without the occurrence of resource shortage. Meeting MRC requirement is very important for emergency response workflows because it is desired to not see any resource shortage in any case in emergence response. To analyze MRC, we need to find out the maximum amount of each type of resources that can be held or consumed in the execution of a workflow. We present two approaches in this section. One is through reachability analysis, the other one is based on ROWN structure and applies to a class of ROWNs.

A. Reachability Analysis Based Approach

Let H be th e reachable state set. For each t ype of resource r_i, Find a state (M_k, S_k) such that

$$S_k(r_i) = \min\{S_j(r_i) \mid (M_j, S_j) \in H\} \quad (5)$$

$S_k(r_i)$ is the lowest possible level of availability of resource r_i during the workflow execution. Therefore, $S_0(r_i) - S_k(r_i)$ indicates the minimum requirement on resource r_i in order for the workflow to be executed along all possible path. Denote it by $\mathrm{Rq}(r_i)$.

$$\mathrm{Rq}(r_i) = S_0(r_i) - \min\{S_j(r_i) \mid (M_j, S_j) \in H\} \quad (6)$$

There are several advantages using this approach for resource requirements analysis: First, it is simple. One can easily modify an existing Petri net tool to s upport this functionality. Secondly, it allows multi-case resources requirements analysis. The number of cases are specified by the number of tokens in the source place i at the initial state. Thirdly, it v irtually works for all workflows, regardless of whether they are complex or simple in terms of control flow and whether they are big or small in size.

B. Free-choice Workflow Nets

We only consider *free-choice* workflows. A workflow net free-choice if and only if

$$p_1{*}\cap p_2{*} \neq \phi \Rightarrow |p_1{*}|=|p_2{*}| = 1, \forall p_1, p_2 \in P,$$

Free-choice workflow net does not allow confusion, a situation where conflict and concurrency are mixed.

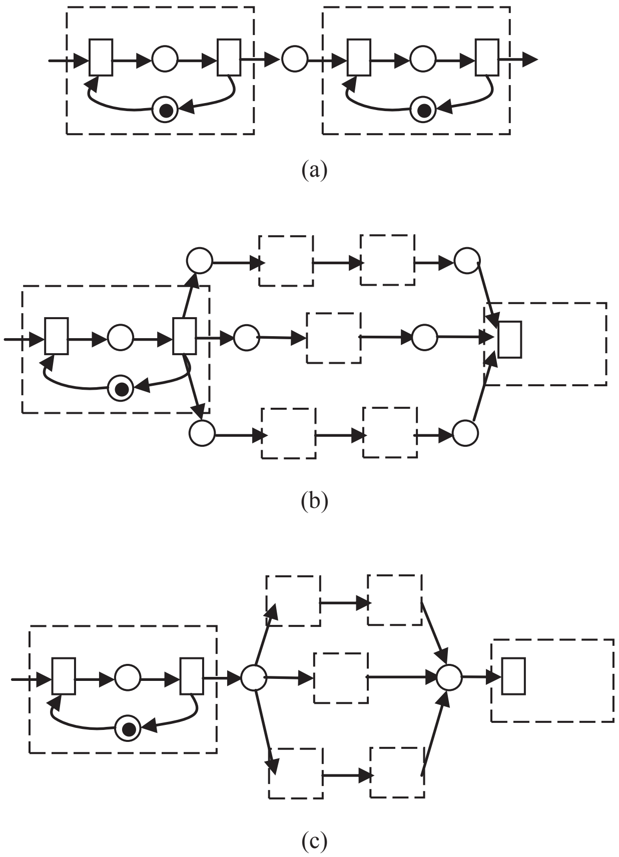
C. Well-nested Workflows

Based on the concept of free-choice workflows, we further introduce *well-nested* workflows. To facilitate the definition of well-nested workflows, a piece of workflow in which tasks are serially connected to form a single branch is called a *procedural branch*, and a piece of workflow in which two or more procedural branches are sprung out from a start task and then join in an end task is called a *fork-join block*. There are two types of fork-join blocks: one is *parallel split-synchronization block* (*PS-blocks*), in which multiple tasks are trigged by one task, run currently and are eventually synchronized at another task. The other type is *exclusive choice-simple merge blocks* (*ES-blocks*), in which multiple tasks are trigged by one task, run exclusively, and whichever is selected to run eventually triggers a common task. Fig. 3 shows an example of a procedural branch, a PS-block and an ES-block.

A *well-nested workflow* is recursively defined as following:

1. A procedural branch is said to be a well-nested workflow.
2. A workflow resulted from replacing a task in a well-nested workflow with a fork-join block is also a well-nested workflow

Fig. 4 shows three well-nested workflows and their relationship.


(a)

(b)

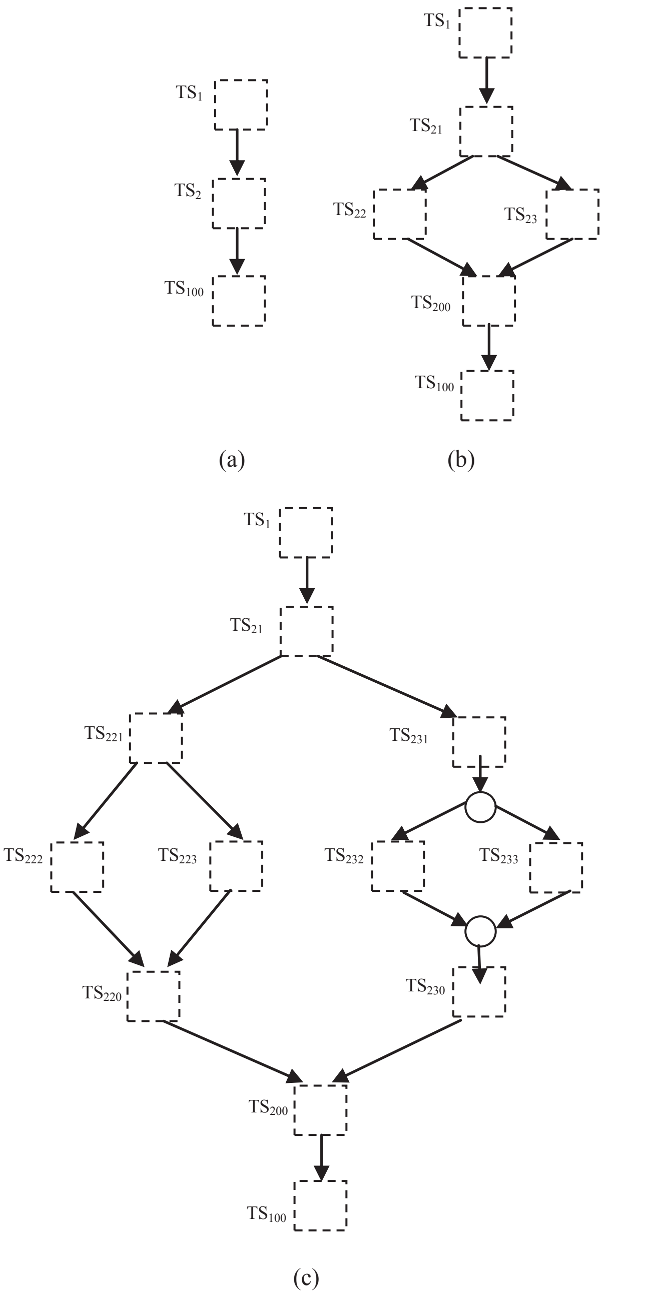
(c)

Fig. 3. (a) A procedural branch. (b) A PS-block.
(c) An ES-block

C. Resource Analysis

The purpose of resource analysis is to track the maximum resource consumption of each type of resource during the workflow execution by structurally traversing each branch of the workflow from the start task to end task.

A workflow's maximum resource consumption (MRC) is, for each type of resources, the minimum amount of the resource that allows the workflow to complete its execution along all possible execution paths.



(a) (b)

(c)

Fig. 4. (a) A three-task procedural workflow.
(b) The middle task in (a) is replaced with an ES-block.
(c) One task in (b) is replaced with and

MRC analysis for a well-nested workflow is mainly based on the analysis of a procedural branch. Consider a procedural branch constituting of tasks TS_1, TS_2, …, TS_n, sequentially. In order to execute TS_1, the requirement on resource r_i is:

$r_i^+(TS_1)$;

In order to execute T_2, the requirement on resource r_i is:

$r_i^+(TS_2) + r_i^+(TS_1) + r_i^-(TS_1)$;

(Notice that $r_i^-(TS_1) \leq 0$) In order to execute T_3, the requirement on resource r_i is:

$$r_i^+(TS_3) + r_i^+(TS_2) + r_i^-(TS_2) + r_i^+(TS_1) + r_i^-(TS_1);$$

…

Therefore, the requirement on resource r_i of executing the procedural branch, denoted by $\mathrm{Rq}(r_i)$ is

$$\begin{aligned}\mathrm{Rq}(r_i) = \max\{ & r_i^+(TS_1), \\ & r_i^+(TS_2) + r_i^+(TS_1) + r_i^-(TS_1), \\ & \quad \ldots \\ & r_i^+(TS_\mathrm{n}) + \textstyle\sum_{k=1}^{n-1}(r_i^+(TS_k) + r_i^-(TS_k))\} \end{aligned} \qquad (7)$$

The net resource consumption is

$$\mathrm{Rc}(r_i) = \textstyle\sum_{k=1}^{s}(r_i^+(TS_k) + r_i^-(TS_k)) \qquad (8)$$

The overall resource requirement of executing the procedural branch is

$$R_r = (\mathrm{Rq}(r_1), \mathrm{Rq}(r_2), \ldots \mathrm{Rq}(r_s)) \qquad (9)$$

For an ES-block composed of w procedural branches, we use formula (9) to calculate the resource requirement for each branch, excluding the starting and ending tasks. Denote by $\mathrm{Rq}_k(r_i)$ the requirement on resource r_i of branch k. Because in any given execution instance the workflow can only choose one branch to execute, therefore, the resource requirement of the ES-block on r_i should be

$$\mathrm{Rq}(r_i) = \max\{\mathrm{Rq}_k(r_i) \mid k=1,2, .., w\} \qquad (10)$$

Notice that it is possible that the maximum value in formula (7) may be identified on different branches for different resources. For example, executing branch one may demand more on resource A than B, while executing branch two may demand more on resource B than A. But the overall resource requirement is a combination of worst case (the maximum demand) for each type of resources.

The maximum net resource consumption on r_i is

$$\mathrm{Rc}(r_i) = \max\{\mathrm{Rc}_k(r_i) \mid k=1,2, .., w\} \qquad (11)$$

For a PS-block composed of w procedural branches, we again use formula (9) to calculate the resource requirement for each branch, excluding the starting and ending tasks. Denote by $\mathrm{Rq}_k(r_i)$ the requirement on resource r_i of branch k. Because these branches will execute concurrently, the worst case (maximum demand for a resource) would be a state in which all branches reach maximum demand for a given resource. Therefore, the resource requirement of the PS-block on r_i should be

$$\mathrm{Rq}(r_i) = \textstyle\sum_{k=1}^{w}\mathrm{Rq}_k(r_i) \qquad (12)$$

The net resource consumption on r_i is

$$\mathrm{Rc}(r_i) = \textstyle\sum_{k=1}^{w}\mathrm{Rc}_k(r_i) \qquad (13)$$

The overall workflow MRC analysis is carried out by the analysis of PS-blocks and ES-blocks and then replacing each of them with a task that is equivalent to the block in terms of maximum resource requirement and resource consumption. It starts from inter-most blocks. Before we formally describe the algorithm, let us illustrate the idea using the example the right-most workflow in Fig. 4(c) first. In this workflow, there are two internal most blocks:

- ES-block, composed of TS_{221}, TS_{222} and TS_{223} and TS_{220}.
- PS-bock, composed of TS_{231}, TS_{232} and TS_{233} and TS_{230}.

For the ES block, we replace the two branches with a new task T_{22}, while for the PS-block, we replace the three branches with a new task T_{23}, as illustrated in Fig. 5(a). Now there is only one block in Fig. 5, which starts with TS_{21} and ends with TS_{200} and has two branches, each having three tasks. We replace each branch with a single new task, and then replace the two branches (now each branch has only one task) with a new task. After that, only one branch is left in the ROWN.

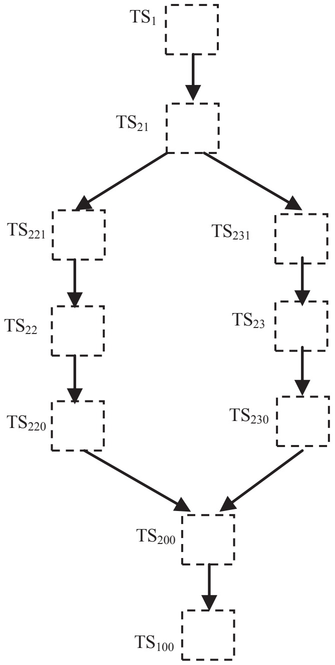


Fig. 5. An equivalent ROWN to Fig. 4(c).

For any type of blocks, when all its branches are replaced with a single task TS_b, the equivalent resource parameters of TS_b for resource r_i are:

$$r_i^+(TS_\mathrm{b}) = \mathrm{Rq}(r_i) \qquad (14)$$

$$r_i^-(TS_\mathrm{b}) = \mathrm{Rq}(r_i) - \mathrm{Rc}(r_i) \qquad (15)$$

If the block is an ES-block, $\mathrm{Rq}(r_i)$ and $\mathrm{Rc}(r_i)$ are given by (10) and (11), respectively. If it is a PS-block, they are given by (12) and (13).

MRC analysis algorithm

<u>*Input*</u>: A resource oriented workflow

<u>*Output*</u>: The maximum requirement on any type of resource r_i to allow the workflow execute along all possible paths.

Step 1: Identify all blocks.

Step 2: If no blocks exist, go to Step 4.

Step 3: Identify an inter-most block.

Step 3.1: Calculate each branch's resource requirement Rq(r_i) and net consumption Rc(r_i) using formulas (9) and (10).

Step 3.2: Replace all branches with a single connected task T_b, calculate its r_b^c and r_b^p using formulas (10), (11), (14) and (15) if it is an ES-block, or using (12), (13), (14) and (15) if it is an PS-block.

Step 3.3: Go to *Step* 1.

Step 4: Using formula (12) to calculate Rq(r_i). Output Rq(r_i).

For example, when we convert the ROWN shown in Fig. 4(c) to the one shown in Fig. 5, the resource parameters for the equivalent task TS$_{22}$ should be

$$r_i^+(TS_{22}) = \max\{\, r_i^+(TS_{222}),\, r_i^+(TS_{222})\};$$

$$r_i^-(TS_{22}) = \max\{\, r_i^+(TS_{222}),\, r_i^+(TS_{222})\} - \max\{(r_i^+(TS_{222}) - r_i^-(TS_{222})) + (r_i^+(TS_{222}) - r_i^-(TS_{222}))\}.$$

IV. Concluding Remarks

Resource-oriented workflow nets (ROWN) are introduced in this paper to analyze resources usage and requirements for workflows. ROWN are an extension to workflow nets by associating resource change with each transition that represents resource consumed, occupied, produced or released when the event that the transition stands for happens. Transition firing and state change rules are presented for ROWN. Two approaches for resource analysis are proposed. One is through reachability analysis, the other one is based the ROWN structure. The reachability approach applies to any kind of ROWN, while the structure based approach only applies to well-nested workflows but the analysis algorithm is very efficient. ROWN are different from resource-constrained workflow nets and resource-oriented Petri nets in that these two nets can only deal with durable resources but ROWN are good for both durable and non-durable resource analysis.

More work needs to be done in workflow resource analysis. One is to find conditions for a ROWN to be deadlock free. Another one is considering the possibility that a task is abolished in the middle of execution.

V. References

[1] W.M.P. van der Aalst, "Verification of workflow nets", *Proceedings of Application and Theory of Petri Nets*, Volume 1248 of Lecture Notes in Computer Science, pp. 407-426, 1997.

[2] M. P. Fanti and M. Zhou, "Deadlock control methods in automeated manufacturing systems," IEEE Transactions on Systems, Man and Cybernetics, Part A, 34(1), 5-21, 2004.

[3] K. van Hee, N. Sidorova and M. Voorhoeve, "Resource-constrained workflow ntes," Fundamenta Informaticae, 71(2-3):243-257, 2005.

[4] G. Juh´as, I. Kazlov, and A. Juh´asov´a. Instance Deadlock: A Mistery behind Frozen Programs. PETRI NETS 2010, LNCS vol. 6128, pp. 1-17. Springer, 2010.

[5] María Martos-Salgado, Fernando Rosa-Velardo: Dynamic Soundness in Resource-Constrained Workflow Nets. FMOODS/FORTE 2011: 259-273

[6] J. Wang, W. Tepfenhart and D. Rosca, Emergency Response Workflow Resource Requirements Modeling and Analysis, *IEEE Transactions on Systems, Man and Cybernetics*, Part C, vol. 39, no. 3, 270-283, 2009.

[7] J. Wang, D. Rosca, W. Tepfenhart, and A. Milewski, "Incident command systems workflow modeling and analysis: A case study," *Proceedings of the 3rd International ISCRAM Conference* (B. Van de Walle and M. Turoff, eds.), Newark, NJ, May 2006.

[8] J. Wang, D. Rosca, W. Tepfenhart, A. Milewski and M. Stoute, Dynamic workflow modeling and analysis in incident command systems, *IEEE Transactions on Systems, Man and Cybernetics*, Part A, vol. 38, no. 5, 1041-1055, 2008.

[9] N. Wu and M. Zhou, Resource-Oriented Petri Nets in Deadlock Avoidance of AGV Systems, *Proceedings of the 2001 IEEE International Conference on Robotics & Automation*, Seoul, Korea, May 21-26, 2001

[10] N. Wu and M. Zhou, *System Modeling and Control with Resource-*Oriented Petri Nets, CRC Press, Oct. 2009.

ACADA:

Access Control-driven Architecture with Dynamic Adaptation

Óscar Mortágua Pereira, Rui L. Aguiar
Instituto de Telecomunicações
DETI, University of Aveiro
Aveiro, Portugal
{omp,ruilaa}@ua.pt

Maribel Yasmina Santos
Centro Algoritmi
University of Minho
Guimarães, Portugal
maribel@dsi.uminho.pt

Abstract— **Programmers of relational database applications use software solutions (Hibernate, JDBC, LINQ, ADO.NET) to ease the development process of business tiers. These software solutions were not devised to address access control policies, much less for evolving access control policies, in spite of their unavoidable relevance. Currently, access control policies, whenever implemented, are enforced by independent components leading to a separation between policies and their enforcement. This paper proposes a new approach based on an architectural model referred to here as the Access Control-driven Architecture with Dynamic Adaptation (ACADA). Solutions based on ACADA are automatically built to statically enforce access control policies based on schemas of Create, Read, Update and Delete (CRUD) expressions. Then, CRUD expressions are dynamically deployed at runtime driven by established access control policies. Any update in the policies is followed by an adaptation process to keep access control mechanisms aligned with the policies to be enforced. A proof of concept based on Java and Java Database Connectivity (JDBC) is also presented.**

Keywords-access control;software architecture; adaptive systems.

I. Introduction

Software systems have increasingly played a key role in all dimensions of our existence as humans, such as transport operators, financial movements, e-health, e-governance and national/international security. They are responsible for managing sensitive data that needs to be kept secure from unauthorized usage. Access control policies (ACP) are a critical aspect of security. ACP are aimed at preventing unauthorized access to sensitive data and is usually implemented in a three phase approach [1]: security policy definition; security model to be followed; and, finally, security enforcement mechanism. Security policies define rules through which access control is governed. The four main strategies for regulating access control policies are [2, 3]: discretionary access control (DAC), mandatory access control (MAC), Role-based access control (RBAC) and credential-based access control. Security models provide formal representations [4-8] for security policies. Security enforcement mechanisms implement the security policy formalized by the security model. ACP exist to keep sensitive data safe, mostly kept and managed by database management systems. Among the several paradigms, the relational database management systems (RDBMS) continue to be one of the most successful one to manage data and, therefore, to build database applications. Beyond RDBMS, software architects use other current software solutions (CuSS), such as JDBC [9], ODBC [10], JPA [11], LINQ [12], Hibernate [13], ADO.NET [14] and Ruby on Rails [15] to ease the development process of business tiers of database applications. Unfortunately, CuSS were devised to tackle the impedance mismatch issue [16], leaving ACP out of their scope. Current mechanisms to enforce ACP to data residing in a RDBMS consist of designing a separate security layer, following one of two different approaches: traditional and PEP-PDP:

1) The traditional approach is based on a security software layer developed by security experts using RDBMS tools. ACP architecture vary from RDBMS to RDBMS but comprise several entities, such as users, roles, database schemas and permissions. They are directly managed by RDBMS and are completely transparent to applications. Their presence is only noticed if some unauthorized access is detected. Basically, before being executed, SQL statements are evaluated by RDBMS to check their compliance with the established ACP. If any violation is detected, SQL statements are rejected, otherwise they are executed.

2) The PEP-PDP approach consists in a security software layer with two main functionalities: the policy decision point (PDP) and the policy enforcement point (PEP), as defined in XACML [17] and used in [18], see Figure 1. The PEP intercepts users requests for accessing a resource protected by an ACP (Figure 1, 1) and enforces the decision to be performed by PDP on this access authorization. PDP evaluates requests to access a resource against the ACP to decide whether to grant or to deny the access (Figure 1, 2). If authorization is granted, the action is sent by the PEP to RDBMS to be executed (Figure 1, 3) and, if no other

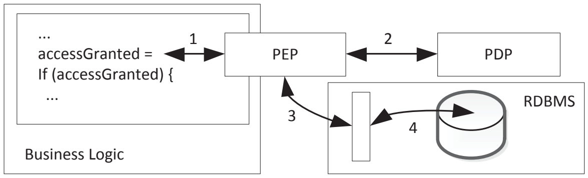


Figure 1. PEP-PDP approach.

access control exists, the action is executed by the RDBMS (Figure 1, 4). PDPs are designed and configured by security experts and exist as independent components. PEPs are intentionally inserted in key points of the source code to enforce PDP decisions.

Both approaches impose a sharp separation between ACP and the mechanisms responsible for their enforcement. This fragility is also apparent in CuSS: security layers exist to enforce ACP and CuSS exist to ease the development process of database applications. This separation of roles entails four important drawbacks regarding the usage of current CuSS:

1) For programmers who use CuSS, this separation demands a complete mastering on the established ACP and on their dependency on database schemas. This mastering is very difficult to be sustained when the complexity of ACP increases, usually coupled by an increase in complexity of databases schemas.

2) Programmers who use CuSS are free to write any CRUD expression opening a security gap. CRUD expressions may be in compliance with the formalized ACP but violating security rules that are not possible to be formalized. ACP are suited to control the access to schema objects but not to control the information SQL statements may get from them. If a user has the permission to, for example, read a set of columns from one or more tables, it is not possible to prevent any SQL statement from reading that data. Select statements may select raw data or may use aggregate functions, for example, to select critical statistical information, opening a possible security gap.

3) Whenever ACP are updated, the correspondent access control mechanisms have to be updated in advance. There is no way to automatically translate ACP into access control mechanisms of CuSS.

4) Some ACP need to be hard-coded to manage runtime constraints. There is no way to automatically update these scattered and hidden hard-coded access control mechanisms in current CuSS.

To tackle the aforementioned drawbacks we propose a new architecture for CuSS, herein referred to as Access Control-driven Architecture with Dynamic Adaptation (ACADA). Software solutions cease to be of general use and become specialized solutions to address specific business areas, such as accountability, warehouse and customers. They are automatically built from a business architectural model, enforcing ACP defined by a security expert. ACP are statically enforced by typed objects driven by schemas of Create, Read, Update and Delete (CRUD) expressions. Then, CRUD expressions are deployed at runtime in accordance with the established ACP. Any modification in the ACP is followed by an adaptation process to keep access control mechanisms aligned with the policies to be enforced. A proof of concept based on Java, JDBC [9] and SQL Server 2008 is also presented.

This paper is organized as follows: section II presents the motivation and related word; section III presents the proposed approach; section IV presents a proof of concept and, finally, section V presents the final conclusion.

II. MOTIVATION AND RELATED WORK

CuSS have been devised to improve the development process of business logic mainly for tackling the impedance mismatch [16]. From them, two categories have had a wide acceptance in the academic and commercial forums: 1) Object-to-Relational mapping (O/RM) tools [19, 20] (LINQ [12], Hibernate [13], Java Persistent API (JPA) [11], Oracle TopLink [21], CRUD on Rails [15]) and 2) Low Level API (JDBC [9], OBDC [10], ADO.NET [14]). Other solutions, such as embedded SQL [22] (SQLJ [23]), have achieved some acceptance in the past. Others were proposed but without any general known acceptance: Safe Query Objects [24] and SQL DOM [25].

Listing 1 shows the usage of four CuSS (JDBC, ADO.NET, JPA and LINQ) for updating the attribute *totalValue* returned by the query "*select clientId, ttlValue from Orders where date=2012-01-31*". Programmers are completely free to edit any CRUD expression (CRUD expressions are encoded inside strings), to execute it (line 2, 9, 20, 27) and to update the attribute *ttlValue* (line 4-5, 14-16, 21-24, 28-29). There is no sign of any ACP: neither for the CRUD expression being executed nor for the updated attribute. Beyond updating *totalValue*, nothing prevents programmers from writing source code to update any other attribute. Programmers have no guidance either on the established ACP or on the underlying database schema. Only after writing and running the source code, programmers become aware of any ACP violation or any database schema nonconformity. Moreover, this same source code may be

```
 1 // JDBC - Java
 2 rs=st.executeQuery(sql);
 3 rs.next();
 4 rs.updateFloat(“ttlValue”, newValue);
 5 rs.updateRow();
 6
 7 //ADO.NET – C#
 8 SqlDataAdapter da=new SqlDataAdapter();
 9  da.SelectCommand=new SqlCommand(sql,conn);
10  SqlCommandBuilder cb=new SqlCommandBuilder(da);
11 DataSet ds=new DataSet();
12 da.Fill(ds,"Orders");
13 DataRow dr=ds.Tables["Orders"].Rows[0];
14 dr["ttlValue"]=totalValue;
15 cb.GetUpdateCommand();
16 da.Update(ds,"Orders");
17
18 // JPA - Java
19 Query qry=em.createNamedQuery(sql,Orders.class);
20 Orders o=(Orders)qry.getSingleResult();
21 em.getTransaction().begin();
22 o.setTtlValue(value);
23 em.persist(o);
24 em.getTransaction().commit();
25
26 //LINQ – C#
27 Order ord=(from o in Orders select o).Single();
28 ord.ttlValue=value;
29 db.SubmitChanges();
```

Listing 1. Examples using CuSS

```
select o.clientId,SUM(o.ttlValue) as ttlValue
 from Orders as o
 where o.date between '2012-01-01' and '2012-01-31'
group by o.clientId
order by o.clientId desc
```

Listing 2. CRUD expression with aggregate function.

used to execute an infinite number of different CRUD expressions requiring the same ACP, such as the one shown in Listing 2. There is no way to avoid this type of security violation. Even if a PEP was used, it would not solve any of the hurdles previously presented. An example of the need for evolving ACP, is the designation of a secretary Susanne to be temporally allowed to update clients' *ttlValue*. Her role has to be changed but roles of other secretaries are to be kept unchanged. ACP foresee this possibility by using the delegation concept. The problem is the lack of preparedness of CuSS to accommodate this situation. Source-code needs to be modified to accommodate the new permission for Susanne. The situation will further deteriorate if the permission is to be only granted while she is within the facilities of the company. The use of hard-coded mechanisms to enforce ACP entails maintenance activities on source-code of client-side components of database applications whenever ACP evolve. CuSS and current access control mechanisms are not prepared to seamlessly accommodate and enforce these evolving ACP.

To address these issues several solutions have been proposed.

SELINKS [18] is a programming language in the type of LINQ and Ruby on Rails which extends Links [26]. Security policies are coded as user-defined functions on DBMS. Through a type system named as Fable, it is assured that sensitive data is never accessed directly without first consulting the appropriate policy enforcement function. Policy functions, running in a remote server, check at runtime what type of actions users are granted to perform, basically controlling more efficiently what RDBMS are currently able to do, and this way not tackling the need to master ACP and database schemas. Moreover, if ACP evolve there will be no way to automatically accommodate the modifications in the client-side components.

Jif [27] extends Java with support for information access control and also for information flow control. The access control is assured by adding labels that express ACP. Jif addresses some relevant aspects such as the enforcement of security policies at compile time and at runtime. Anyway, at development time programmers will only be aware of inconsistencies after running the Jif compiler. In spite of its valuable contribution, Jif does not address the announced goals of this research.

Rizvi et al. [28] uses views to filter contents of tables and simultaneously to infer and check at runtime the appropriate authorization to execute any query. The process is transparent for users and queries are rejected if they do not have the appropriate authorization. This approach has some disadvantages: 1) the inference rules are complex and time consuming; 2) security enforcement is transparent, so users do not know that their queries are run against views; 3) programmers cannot statically check the correctness of queries which means they are not aware of either the ACP or the underlying database schema.

Morin et al. [29] uses a security-driven model-based dynamic adaptation process to address simultaneously access control and software evolution. The approach begins by composing security meta-models (to describe access control policies) and architecture meta-models (to describe the application architecture). They also show how to map (statically and dynamically) security concepts into architectural concepts. This approach is mainly based on establishing bindings between components from different layers to enforce security policies. Authors didn´t address the key issue of how to statically incorporate the established security policies in software artifacts.

Differential-privacy [30] has had significant attention from the research community. It is mainly focused on preserving privacy from statistical databases. It really does not directly address the point here under discussion. The interesting aspect is Frank McSherry's [31] approach to address differential-privacy: PINQ - a LINQ extension. The key aspect is that the privacy guarantees are provided by PINQ itself not requiring any expertise to enforce privacy policies. PINQ provides the integrated declarative language (SQL like, from LINQ) and simultaneously provides native support for differential-privacy for the queries being written.

III. ACADA: PROPOSED APPROACH

In this section a new architecture, ACADA, is proposed for CuSS. We first introduce an overview for the proposed approach. Then we introduce some relevant aspects of CuSS from which ACADA will evolve. Then, CRUD Schemas are presented as key entities of ACADA. Finally, ACADA is presented.

A. Overview

ACADA is an architecture for software solutions used in business tiers of database applications. Each software solution derived from ACADA, herein known as Access Control-driven Component with Dynamic Adaptation

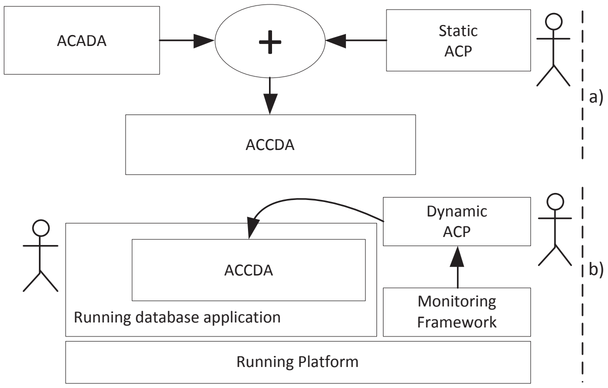


Figure 2. Proposed approach for the adaptation of ACCDA a) static composition and b) dynamic adaptation.

(ACCDA), is customized to address a specific need of a business area, such as accountability, warehouse and sales. Then, at runtime, they are dynamically adapted to be kept aligned with the established ACP. This approach combines a static composition of ACCDA and a dynamic adaptation to the running context as shown in Figure 2. During the static composition, Figure 2 a), static parts of ACP are used to build an ACCDA based on an architectural model (ACADA). Static parts of ACP comprise the information needed to build the business logic to manage CRUD expressions. Any modification in the static parts compels to a new static composition. During the dynamic adaptation, see Figure 2 b), CRUD expressions are dynamically assigned and unassigned to running ACCDA in accordance with ACP defined for each user. This process is continuous and may have as input data from a monitoring framework and from security experts. Security experts modify ACP, for example, to allow secretary Susanne to update clients' *ttlValue* and, therefore, to use the necessary business logic and necessary CRUD expressions. Monitoring framework updates ACP, for example, only to allow Susanne to update clients' *ttlValue* while she is within the facilities of the company.

B. Relevant Aspects of CuSS

To proceed with a more detailed presentation it is advisable to learn and understand CuSS and the context in which CRUD expressions are executed. Identifying a common base for current approaches is a key aspect to devise ACADA. The following shared functionalities are emphasized:

1) To promote reusability of CRUD expressions, parameters may be used to define runtime values. Parameters are mainly used to define runtime values for clause conditions and for column lists. Listing 3 shows an Update CRUD expression with four parameters: *a*, *b* and *c* are columns and *d* is a condition.

```
update table set a=?, b=?, c=? where d=?
```

Listing 3. CRUD expression with parameters.

2) If CRUD expression type is *Insert*, *Update* or *Delete*, its execution returns a value indicating the number of affected rows.

3) Data returned by CRUD expressions of type *Select* is managed by a local memory structure (LMS) internally created by CuSS. Some LMS are readable only and others are readable and modifiable. Modifiable LMS provide additional functionalities to modify their internal content: update data, delete data and insert new data. These actions, are equivalent to CRUD expressions and the results are committed into the host RDBMS.

Thus, CRUD expressions are used at two levels: at the application level and at the LMS level. At the application level CRUD expressions are explicitly used, while at the LMS level CRUD expressions are implicitly used. In both cases, to guarantee compliance with established ACP and with database schemas, CRUD expressions need to be emanated from the established ACP and from database schemas and not from programmers' will. In reality, CRUD expressions and LMS are the key assets of CuSS to interact with RDBMS. They are the entities used to read data from databases and to alter the state of databases.

C. Crud Schemas

CRUD expressions and LMS are two key entities of ACADA. They are the entities used to interact with databases and, therefore, the privileged entities through which ACP may be enforced. To this end, ACADA formalizes CRUD expressions and LMS using a schema herein known as CRUD schema. A CRUD schema is a set of services needed to manage the execution of CRUD expressions and the associated LMS (only for Select CRUD expressions). It comprises four independent parts: a) a mandatory type schema – the CRUD type - query (Select) or execute (Insert, Update or Delete); b) an optional parameter schema – to set the runtime values for the conditions used inside SQL clauses, such as the *"where"* and *"having"* clauses and runtime values for column lists (only for Insert and Update CRUD expressions); c) mandatory result schema for Insert, Update and Delete CRUD expressions – to handle the number of affected rows during the CRUD expressions execution and, finally, d) a mandatory LMS schema for Select CRUD expressions – to manage the permissions on the LMS.

Table I shows a possible definition for the permissions on an LMS derived from the CRUD expression *Select a,b,c,d,e from table*. This access matrix [32] like representation, defines for each attribute of this LMS, which LMS functionalities (read, update, insert, delete) are authorized. *delete* action is authorized in a tuple basis and, therefore, it is executed as an atomic action for all attributes.

TABLE I. TABLE OF PERMISSIONS IN A LMS

	a	b	c	d
Read	yes	no	yes	yes
Update	no	yes	no	yes
Insert	yes	yes	no	no
delete	yes			

D. ACADA Presentation

Figure 3 presents a class diagram for an ACADA model, for building ACCDA. Figure 3 a) presents the entities used to define CRUD schemas. Figure 3 b) presents the final class diagram of ACADA. There are six types of entities: ILMS, ICrudSchema, Manager, IFactory, IConfig and CrudSchema:

ILMS (used for Select CRUD expressions only) defines the permissions on LMS, following the approach presented inTable I: IRead defines the readable attributes, IUpdate defines the updatable attributes, IInsert defines the insertable attributes and IDelete defines if LMS's rows are deletable. Aditionally, ILMS also uses IScroll to define the scrollable methods to be made available.

ICrudSchema is used to model the business logic for each

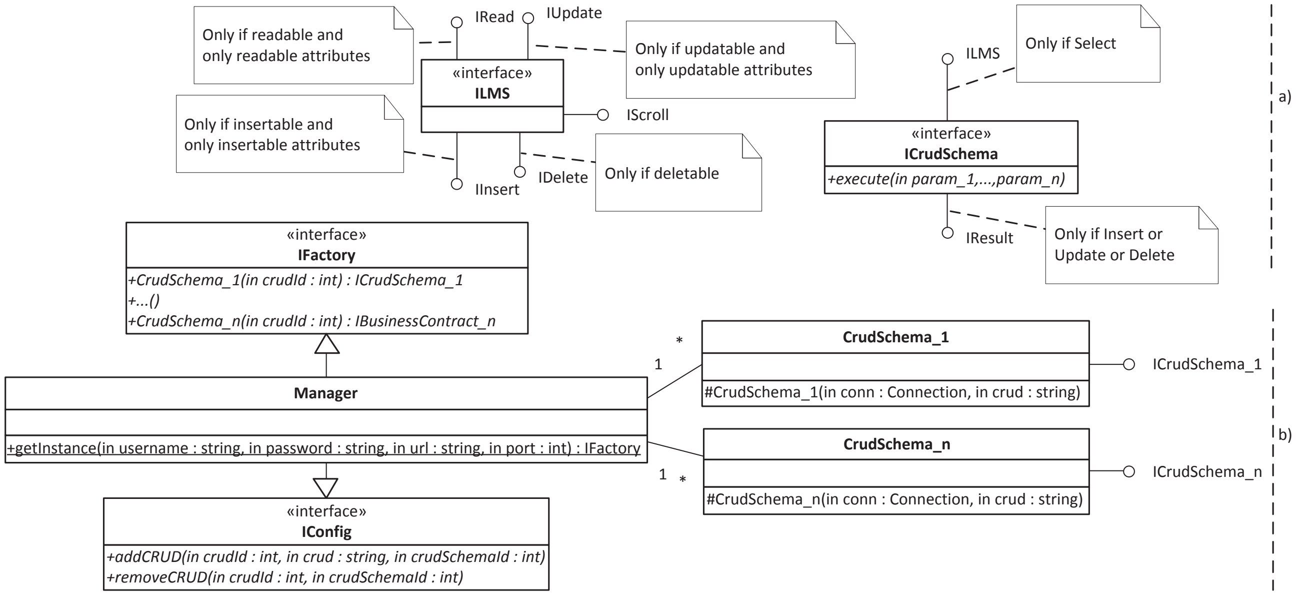

Only if readable and
only readable attributes
IRead
IUpdate
Only if updatable and
only updatable attributes
«interface»
ILMS
IScroll
Only if insertable and
only insertable attributes
IInsert
IDelete
Only if deletable
ILMS
Only if Select
«interface»
ICrudSchema
+execute(in param_1,...,param_n)
IResult
Only if Insert or
Update or Delete
a)
«interface»
IFactory
+CrudSchema_1(in crudId : int) : ICrudSchema_1
+...()
+CrudSchema_n(in crudId : int) : IBusinessContract_n
Manager
+getInstance(in username : string, in password : string, in url : string, in port : int) : IFactory
CrudSchema_1
#CrudSchema_1(in conn : Connection, in crud : string)
ICrudSchema_1
CrudSchema_n
#CrudSchema_n(in conn : Connection, in crud : string)
ICrudSchema_n
1
*
1
*
b)
«interface»
IConfig
+addCRUD(in crudId : int, in crud : string, in crudSchemaId : int)
+removeCRUD(in crudId : int, in crudSchemaId : int)

Figure 3. Class diagram for ACADA: a) entities used to define schemas of CUD expressions; b) final class diagram of ACADA.

CRUD schema instance – see ICrudSchema_1, …, ICrudSchema_n, in Figure 3 b). It comprises a mandatory method, *execute(param_1,…,param_n)*, to set the parameter schema and to execute CRUD expressions, and two optional interfaces: *ILMS* and *IResult*. IResult (for Insert, Update and Delete CRUD expressions only) implements the result schema.

CrudSchema is used to implement ICrudSchema. The arguments *conn* and *crud* are a connection object to a database and the CRUD expression to be managed, respectively. Each CrudSchema is able to manage any CRUD expression with equivalent schema. CRUD expressions with the same CRUD Schema are herein known as sibling CRUD expressions. Listing 4 presents two simple sibling CRUD expressions: both are Select, both have the same select list, none has column list or condition list parameters. This property is an opportunity to extend the adaptation capability of ACADA. In practice each CrudSchema is able to manage an infinite number of sibling CRUD expressions. Thus, any CrudSchema used by UserA is able to manage not one CRUD expression but one set of sibling CRUD expressions and the same CrudSchema may be used by UserB to manage a different set of sibling CRUD expressions.

```
    Select * from table;
    Select * from table where id=10;
```

Listing 4. 2 Sibling CRUD expressions.

Manager implements two interfaces (IFactory and IConfig) and is the entry point for creating instances of ACCDA (using *getInstance*, authentication is required). *url* and *port* are used to connect to a component responsible for the dynamic adaptation process and for the authentication of users.

IConfig is used to dynamically adapt running instances of ACCDA to users previously authenticated. The dynamic process comprises the deployment of CRUD expressions and also the required information to set the connection to RDBMS (not shown). Each CRUD expression is assigned to a CrudSchema responsible for its management. IConfig is implemented using a socket to decouple ACCDA from components responsible for managing the dynamic adaptation process.

IFactory is used to create instances of CrudSchema.Users request the access to a CrudSchema and to a CRUD expression. The access is granted or denied depending on the ACP defined by the dynamic adaptation process.

IV. Proof of Concept

In this section a proof of concept based on Java and JDBC (sqljdbc4) for SQL Server 2008, is presented. A component, herein known as ACEngine, was developed to automatically create releases of ACCDA. The biggest challenge was centered on the approach to be followed to formalize CRUD schemas to be used to define the target business area. Several approaches were considered, among them XML and standard Java interfaces. In spite of being less expressive than XML, Java interfaces proved to be an efficient and effective approach. Programmers do not need to use a different development environment, interfaces are basic entities of any object-oriented programming language and are widely used, interfaces are easily edited and maintained and, finally, CRUD schemas have also been defined as interfaces, see Figure 3. These were the fundamental reasons for having opted for Java interfaces in detriment of XML.

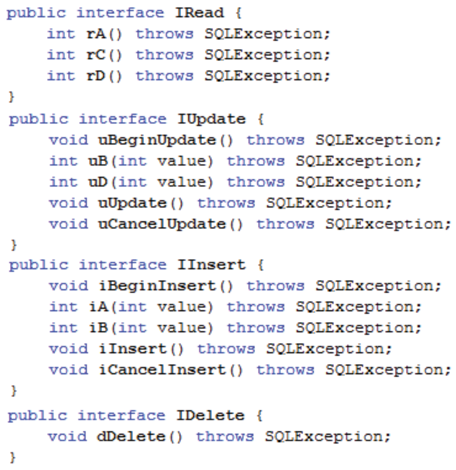
ACEngine accepts as input, for each CRUD schema, one interface extending all the necessary interfaces as defined in

ICrudSchema and shown in Figure 3. Then, through reflection, ACEngine detects which interfaces are defined and which methods need to be implemented to automatically create the source code.

A component for the dynamic adaptation process was also created. The main information is organized around users. For each user it is defined its authentication parameters (username and password), the assigned CRUD expressions and the correspondent CrudSchemas. Any modification in this information is immediately sent to users running ACCDA instances.

The example to be presented is based on the Select and on the permissions used in Table I.

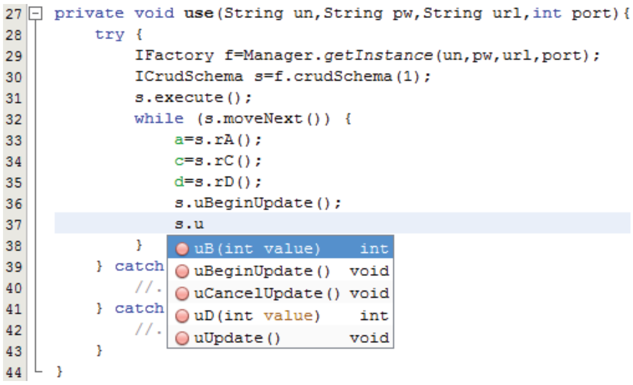
Figure 4 shows the four interfaces used to formalize the LMS's permissions, which are in agreement with Table I. CuSS use the same access methods for updating and for inserting attributes. This approach prevents the separation between update permissions and insert permissions. Therefore, to overcome this limitation, access methods of IUpdate and IInsert have been given different names. IUpdate use a prefix *u* and IInsert use a prefix *i*. Some additional methods, such as *uUpdate()* and *iBeginInsert()* are used to implement the update and insert protocols defined by JDBC for LMS.



```
public interface IRead {
    int rA() throws SQLException;
    int rC() throws SQLException;
    int rD() throws SQLException;
}
public interface IUpdate {
    void uBeginUpdate() throws SQLException;
    int uB(int value) throws SQLException;
    int uD(int value) throws SQLException;
    void uUpdate() throws SQLException;
    void uCancelUpdate() throws SQLException;
}
public interface IInsert {
    void iBeginInsert() throws SQLException;
    int iA(int value) throws SQLException;
    int iB(int value) throws SQLException;
    void iInsert() throws SQLException;
    void iCancelInsert() throws SQLException;
}
public interface IDelete {
    void dDelete() throws SQLException;
}
```

Figure 4. IRead, IUpdate, IInsert and IDelete interfaces.

Figure 5 presents the usage of ACCDA from a programmer's perspective. An attempt is done to create a new ACCDA instance (line 29). It will raise an exception if a connection to ACDynam fails or if authentication fails. Authentication is processed by ACDynam and if it succeeds, ACDynam transfers to ACCDA all the CRUD expressions, in accordance with the ACP assigned to the authenticated user. Then, an attempt is made to create an instance of a crudSchema for managing the CRUD expression identified by token 1 (line 30). As previously mentioned, programmers cannot edit CRUD expressions.They are only allowed to use CRUD expressions made available by ACDynam, overcoming this way the security gap of CuSS. If it fails (user is not authorized to execute the CRUD expression), an exception is raised. If not, CRUD expression is executed (line 31) and LMS is scrolled row by row (line 32). The dynamic adaptation is on behalf of ACDynam that, at any time, may modify the permission to use this CRUD expression. There is no need to update any source-code this way overcoming CuSS to be adapted to evolving ACP. The three readable attributes are read (line 33-35). Update protocol is started (line 36). Auto-completion window (line 38-43) shows the available methods to update attributes of LMS, relieving programmers from mastering the established ACP and database schema. This type of guided-assistance is available for all operations involving ACCDA, this way overcoming the need for mastering ACP and database schemas when using CuSS.



```
27  private void use(String un,String pw,String url,int port){
28      try {
29          IFactory f=Manager.getInstance(un,pw,url,port);
30          ICrudSchema s=f.crudSchema(1);
31          s.execute();
32          while (s.moveNext()) {
33              a=s.rA();
34              c=s.rC();
35              d=s.rD();
36              s.uBeginUpdate();
37              s.u
38          }
39      } catch
40          //.
41      } catch
42          //.
43      }
44  }
```

Figure 5. ACCDA from the programmer's perspective.

V. CONCLUSION

In this paper a new architecture (ACADA) was presented to devise solutions driven by ACP and able to be dynamically adapted to deal with evolving ACP. The adaptation process of each ACCDA release is achieved in a two phase approach: static composition and dynamic adaptation. Static composition is triggered whenever a maintenance activity is necessary in CRUD Schemas. Dynamic adaptation is a continuous process where CRUD expressions are deployed to running ACCDA instances in accordance with ACP. ACP are dynamically updated by a monitoring framework and/or by security experts.

Source code is automatically generated from an architectural model and from ACP defined by a security expert. In opposite to CuSS, programmers using ACCDA are relieved from mastering ACP and database schemas, and also from writing CRUD expressions. Security is ensured by preventing programmers from writing CRUD expressions and by controlling dynamically, at runtime, the set of CRUD expressions that each user may use. Evolving ACP are seamlessly supported and enforced by ACCDA. An independent and external component keeps the access control mechanisms of ACCDA updated at runtime by assigning and unassigning CRUD expressions. This adaptation capability of ACCDA avoids maintenance activities in the core client-side components of database applications when ACP evolve. The adaptation capacity is significantly improved by

CrudSchema design which, theoretically, is able to manage an infinite number of sibling CRUD expressions.

Summarizing, ACADA overcomes the four drawbacks of CuSS. This achievement is mostly grounded on its two phase adaptation process: static composition and dynamic adaptation.

It is expected that this work may open new perspectives for enforcing evolving ACP in business tier components of database applications.

REFERENCES

[1] P. Samarati and S. D. C. d. Vimercati, "Access Control: Policies, Models, and Mechanisms," Foundations of Security Analysis and Design, pp. 137-109, 2001.

[2] P. Samarati and S. D. C. d. Vimercati, "Access Control: Policies, Models, and Mechanisms," presented at the Revised versions of lectures given during the IFIP WG 1.7 International School on Foundations of Security Analysis and Design on Foundations of Security Analysis and Design: Tutorial Lectures, 2001.

[3] S. D. C. d. Vimercati, S. Foresti, and P. Samarati, "Recent Advances in Access Control - Handbook of Database Security," M. Gertz and S. Jajodia, Eds., ed: Springer US, 2008, pp. 1-26.

[4] D. Basin, J. Doser, and T. Lodderstedt, "Model Driven Security: From UML Models to Access Control Infrastructures," ACM Transactions on Software Engineerig and Methodology, vol. 15, pp. 39-91, 2006.

[5] R. Breu, G. Popp, and M. Alam, "Model Based Development of Access Policies," International Journal on Software Tools for Technology Transfer, vol. 9, pp. 457-470, 2007.

[6] T. Doan, S. Demurjian, T. C. Ting, and A. Ketterl, "MAC and UML for Secure Software Design," presented at the Proceedings of the 2004 ACM Workshop on Formal Methods in Sec urity engineering, Washington DC, USA, 2004.

[7] I. Ray, N. Li, R. France, and D.-K. Kim, "Using UML to Visualize Role-based Access Control Constraints," presented at the Proceedings of the ninth ACM Symposium on Access Control Models and Technologies, Yorktown Heights, New York, USA, 2004.

[8] OASIS, "eXtensible Access Control Markup Language (XACLML)," ed: OASIS Standard.

[9] M. Parsian, JDBC Recipes: A Problem-Solution Approach. NY, USA: Apress, 2005.

[10] Microsoft. (2011 Oct). Microsoft Open Database Connectivity. Available: http://msdn.microsoft.com/en-us/library/ms710252(VS.85).aspx

[11] D. Yang, Java Persistence with JPA: Outskirts Press, 2010.

[12] M. Erik, B. Brian, and B. Gavin, "LINQ: Reconciling Object, Relations and XML in the .NET framework," in A CM SIGMOD International Conference on Management of Data, Chicago,IL,USA, 2006, pp. 706-706.

[13] B. Christian and K. Gavin, Hibernate in Action: Manning Publications Co., 2004.

[14] G. Mead and A. Boehm, ADO.NET 4 Database Programming with C# 2010. USA: Mike Murach & Associates, Inc., 2011.

[15] D. Vohra, "CRUD on Rails - Ruby on Rails for PHP and Java Developers," ed: Springer Berlin Heidelberg, 2007, pp. 71-106.

[16] M. David, "Representing database programs as objects," in Advances in Database Programming Languages, F. Bancilhon and P. Buneman, Eds., ed N.Y.: ACM, 1990, pp. 377-386.

[17] OASIS. (2012 Feb). XACML - eXtensible Access Control Markup Language. Available: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml

[18] B. J. Corcoran, N. Swamy, and M. Hicks, "Cross-tier, Label-based Security Enforcement for Web Applications," presented at the Proceedings of t he 35th SIGMOD International Conference on Management of Data, Providence, Rhode Island, USA, 2009.

[19] W. Keller, "Mapping Objects to Tables - A Pattern Language," in European Conference on Pattern Languages of Progra mming Conference (EuroPLoP), Irsse, Germany, 1997.

[20] R. Lammel and E. Meijer, "Mappings Make data Processing Go 'Round: An Inter-paradigmatic Mapping Tutorial," in Generative and Transformation Techniques in Software Engineering, Braga, Portugal, 2006.

[21] Oracle. (2011 Oct). Oracle TopLink. Available: http://www.oracle.com/technetwork/middleware/toplink/overview/index.html

[22] J. W. Moore, "The ANSI binding of SQL to ADA," Ada Letters, vol. XI, pp. 47-61, 1991.

[23] Part 1: SQL Routines using the Java (TM) Programming Language, 1999.

[24] R. C. William and R. Siddhartha, "Safe query objects: statically typed objects as remotely executable queries," in 27th International Conference on Software Engineering, St. Louis, MO, USA, 2005, pp. 97-106.

[25] A. M. Russell and H. K. Ingolf, "SQL DOM: compile time checking of dynamic SQL statements," in 27th International Conference on Software Engineering, St. Louis, MO, USA, 2005, pp. 88-96.

[26] E. Cooper, S. Lindley, P. Wadler, and J. Yallop, "Links: Web Programming Without Tiers," presented at the Proceedings of the 5th International Conference on Formal Methods for C omponents and Objects, Amsterdam, The Netherlands, 2007.

[27] D. Zhang, O. Arden, K. Vikram, S. Chong, and A. Myers. (2011 Dec). Jif: Java + information flow. Available: http://www.cs.cornell.edu/jif/

[28] S. Rizvi, A. Mendelzon, S. Sudarshan, and P. Roy, "Extending Query Rewriting Techniques for Fine-grained Access Control," presented at the Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data, Paris, France, 2004.

[29] B. Morin, T. Mouelhi, F. Fleurey, Y. L. Traon, O. Barais, and J.-M. Jézéquel, "Security-Driven Model-based Dynamic Adaptation," presented at the Proceedings of the IEEE/ACM International Conference on Automated Software Engineering, Antwerp, Belgium, 2010.

[30] C. Dwork, "Differential Privacy: A Survey of Results," presented at the Proceedings of the 5th International Conference on T heory and Applications of Models of Computation, Xi'an, China, 2008.

[31] F. McSherry, "Privacy Integrated Queries: An Extensible Platform for Privacy-preserving Data Analysis," Commun. ACM, vol. 53, pp. 89-97, 2010.

[32] B. W. Lampson, "Protection," SIGOPS Operating Systems Review, vol. 8, pp. 18-24, 1974.

Connectors for Secure Software Architectures

Michael E. Shin,
Bhavya Malhotra
Department of Computer Science
Texas Tech University
Lubbock, TX 79409-3104
{michael.shin;
bhavy.malhotra}@ttu.edu

Hassan Gomaa
Department of Computer Science
George Mason University
Fairfax, VA 22030-4444
hgomaa@gmu.edu

Taeghyun Kang
Department of Computer Science
Texas Tech University
Lubbock, TX 79409-3104
th.kang@ttu.edu

Abstract

This paper describes secure software connectors encapsulating security services, which are designed separately from application business components in software architectures for business applications. Secure connectors provide application business components with security services when these components need the services. Each secure connector is structured with security relevant objects associated with security services that are needed to make software applications secure. In this paper, secure connectors are designed for different types of communication, such as either synchronous or asynchronous communication, as well as for different security services, such as authentication, authorization, confidentiality, integrity, and non-repudiation. Secure connectors can make complex applications more maintainable by separating security concerns from application concerns in the software architectures. Secure connectors are applied to the software architecture of an e-commerce application.

1. Introduction

With the widespread use of internet technologies, the threats to software applications are increasing day by day. It has become essential to design secure software architectures for applications to counter potential threats. The software architecture can be composed of components and their connectors in which connectors encapsulate the details of communication between components. However, mixing security concerns with business concerns in software architectures makes applications more complex. Therefore designing security concerns separately from the business concerns would make the applications more maintainable.

Several approaches have been developed to design secure applications by means of separation of concerns in software development. Most of the approaches have focused on making application business components secure so that the components perform security services. But less attention has been paid to connectors, which can provide security services for application business components. Security concerns can be encapsulated in software connectors, which are referred to as secure connectors, separately from application components containing application business logic.

This paper describes the secure connectors that are used to design the software architectures for secure applications. The secure connectors are designed separately from application business components by considering different communication patterns between the components as well as security services required by application components. Each secure connector encapsulates security relevant objects to provide application components with security services. Once secure connectors are made, they can be reused for different applications if they match both the required security services and required communication pattern between application components. In this paper, secure connectors are applied to the software architecture of an e-commerce application.

This paper is organized as follows. Section 2 presents existing approaches to implementing security concerns in software applications systems. Section 3 describes the secure connectors for synchronous and asynchronous communication between application components. Section 4 concludes this paper.

2. Related Work

Related work focuses on approaches to designing software architectures for secure applications.

Banerjee et. al. [1] identified several critical dimensions of software security and related them to the building blocks of software architecture. The critical dimensions for a secure software system are authentication, access control, confidentiality, integrity, availability and non-repudiation. The software components, connectors and their configurations are the architectural building blocks, which can be customized to enforce the security dimensions. However, no formal methodology has been specified in [1] to inject security into software architecture.

The security mechanisms are formalized and embedded directly into the software architecture via components in [2]. A component in [2] incorporates multilevel security, in which input and output of that component are properly labeled with security levels. The component enforces the multilevel security policy internally so that security mechanisms are intended to provide secure access control. This technique is developed for a single security model, so it is restricted to access control. Other security features are not taken into account.

Deng et. al. [3] proposed a methodology to model secure software architectures and verify whether required security constraints are assured by the composition of components of the system. This approach introduces security constraint patterns, which specify the security policies that the security system must enforce. However, this approach is lacking focus on what type and form of security policies should be used in a given context and how to implement them in a system design. The components are considered the central units for assembly and deployment. Interactions between components are captured in component interfaces.

Using connectors as the central construct, a distributed software architecture in [4] is composed of a set of components and a set of connectors that can be used to connect the components. The Unified Modeling Language (UML) [10, 11] is used to describe the component interconnection patterns for synchronous, asynchronous and brokered communications. Though different kinds of connectors are proposed in [4], there are no considerations for enforcing security into these connectors. The message communication via the connectors between components is not secure.

In [5], a connector centric approach is used to model, capture, and enforce security. The security characteristics of a software architecture are described and enforced using software connectors. An extensible architecture security contract specified by components is regulated and enforced by connectors. Connectors can decide what principals can execute the connected components. But, this approach models only access control as a security concern. The security properties like confidentially, non-repudiation and integrity are not taken into account.

Aspect-oriented design (AOD) techniques in [6] are used to encapsulate security concerns in complex systems. Different security concerns are modeled as aspects that can be woven into models of essential functionality. The mechanisms required to protect against attacks are used to identify the needing aspects and the strategy for weaving them into a design. There are many benefits of treating security concerns as aspects during design modeling, but this approach requires software engineers to know knowledge of aspect-oriented design.

In earlier work by the authors [7], an approach is described to model complex applications by modeling application requirements and designs separately from security requirements and designs using the UML notation. Security requirements are captured in security use cases and encapsulated in security objects. When a system requires security services, security use cases are extended from the non-secure business use case at extension points. However, [7] paid relatively less attention to secure application architecture where security requirements can be mapped to a secure software architecture.

In later work by the authors [8], an approach is described for modeling the evolution of a non-secure application to a secure application in terms of a requirements model and a software architecture. In the software architecture, security services are encapsulated in connectors separately from components. The security services are activated if the security requirement conditions are satisfied. However, this approach does not offer various secure connectors that are used for different communication patterns and security services.

3. Secure Connectors

The software architecture [14, 15] for concurrent and distributed applications can be designed by means of components and connectors. The components address the functionality of an application, whereas connectors deal with communication between components. Each component defines application business logic that is relatively independent of those provided by other components. A component may request services from other components, or provide services to them through connectors. A connector acts on behalf of components in terms of communication between components, encapsulating the details of inter-component communication.

Separately from application components, security services can be encapsulated in connectors between components in the software architecture for concurrent and distributed applications [8]. The original role of connectors in the software architecture is to provide the mechanism for message communication between components [14, 15]. However, in this paper, the role of connectors is extended to security by adding security services to the connectors, which are referred to as secure connectors. Secure connectors are designed by considering both the security services required by components and the type of message communication required between the components.

The security services provided for components are confidentiality, integrity, non-repudiation, access control, and authentication, as follows:

- Confidentiality security service, which prevents secret information from being disclosed to any unauthorized party, can be achieved by secure connectors encapsulating cryptosystems.
- Integrity security service, which protects against unauthorized changes to secret information, can be performed by secure connectors using message digest (MD) or message authentication code (MAC) [13].

- Non-repudiation security service protects against one party to a transaction later falsely denying that the transaction occurred. Non-repudiation security services can be realized using digital signatures [13].
- Access control security service protects against unauthorized access to valuable resources. Access control may be implemented using mandatory access control (MAC) or role-based access control [12, 16].
- Authentication security service allows an entity (a user or system) to identify itself positively to another entity. This can be achieved using a password, personal-identification number or challenge response.

Typical message communication patterns between components are asynchronous (loosely coupled) message communication and synchronous (tightly coupled) message communication [9], although there are other types of communications between components. An asynchronous message is sent from a sender component to a receiver component and is stored in a queue if the receiver is busy. The sender component can continue to send the next message to the receiver component as long as the queue is not full. In synchronous message communication, a sender component sends a message to a receiver component and waits for a response from the receiver. When a response arrives from the receiver, the sender can continue to work and send the next message to the receiver.

A distributed secure synchronous connection is provided by means of a pair of connectors, namely a secure synchronous sender connector and a secure synchronous receiver connector. The secure synchronous sender and receiver connectors act as stubs sending and receiving messages for their respective components. When the secure sender connector receives a message from the sender component, it applies the security services to the message if required by the component. The secured message is packed by the secure sender connector, which sends it to the secure synchronous receiver connector. When the receiver connector receives a secured and packed message, it checks the security of message and unpacks the message before sending it to the receiver component. Conversely, a response is sent from the receiver component to the sender component via secure connectors. If the response requires security services, the secure connectors apply the appropriate security services.

Fig. 1 depicts secure synchronous sender and receiver connectors for browsing a catalog that requires catalog access control and customer identity confidentiality security services between the Customer and Catalog application components in the business to business (B2B) electronic system. A customer browses through various WWW catalogs and views various catalog items from a given supplier's catalog. The customer may need permission to access a specific catalog, and the customer identity for access control may also need to be confidential. These security requirements are handled by two secure synchronous connectors, the secure synchronous Customer Interface connector for Customer Interface application component and the secure synchronous Catalog Server connector for Catalog Server component. The security services for catalog access control and customer identity confidentiality are encapsulated in the secure synchronous Customer Interface and Catalog Server connectors, separately from Customer Interface and Catalog Server components.

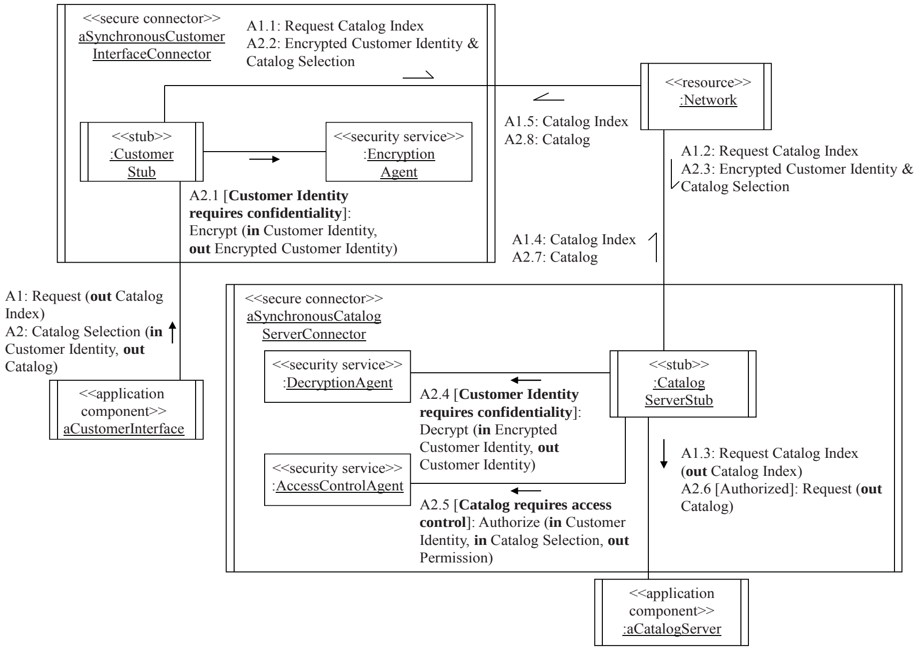


Fig. 1 Secure Synchronous Connector for Confidentiality and Access Control security services

The catalog access control and customer identity confidentiality security services are implemented in security objects in the secure synchronous Customer Interface and Catalog Server connectors in Fig. 1. For customer identity confidentiality, the secure synchronous Customer Interface connector contains the Encryption Agent security object, whereas the secure synchronous Catalog Server connector encapsulates the Decryption Agent security object (Fig. 1). Consider the following secure connection scenario in which the customer requires confidentiality: The Customer Stub in the secure synchronous Customer Interface connector requests the Encryption Agent security object to encrypt the customer identity (message A2.1 in Fig. 1). The Encryption Agent security object encrypts the customer identity with a key. The Customer Stub then sends the encrypted message to the Catalog Server (messages A2.2, A2.3). The encrypted customer identity is decrypted by the Decryption Agent security object in the secure synchronous Catalog Server connector (message A2.4 through in Fig. 1). Access to a specific Catalog is controlled by the secure synchronous Catalog Server connector that encapsulates the Access Control Agent security object (Fig. 1). The Catalog Stub requests Access Control Agent security object to authorize customer access to a specific catalog if catalog requires access control (message A2.5). Access Control Agent authorizes a customer access to a catalog item (message A2.6) depending on the organization's access control policy.

Fig. 2 depicts a secure synchronous connector that provides authentication security service for paying a selected product in the business to customer (B2C) electronic commerce system. A customer puts an item on a product cart and logs into his/her account on the website. The customer needs to be authenticated to get access to his/her account. This authentication security service can be implemented in the secure synchronous Payment Server connector, separately from Customer Interface component and Payment Server component. The secure synchronous Payment Server connector is structured with Authentication Agent and Authentication Data security objects to authenticate a customer identity. Consider the following secure connection scenario in which the customer requires authentication: The Payment Stub in the connector requests the Authentication Agent security object to authenticate a customer identity (message C4 in Fig. 2). The Authentication Agent security object verifies a customer identity using the data stored in the Authentication Data security object (message C5).

A distributed secure asynchronous connection is provided by means of a secure asynchronous sender connector and a secure asynchronous receiver connector. A secure asynchronous connector provides components with security services while it follows the component interconnection pattern for loosely coupled message communication. A secure asynchronous sender connector can encapsulate an Encryption Agent security object that encrypts a plain message before sending the message to the receiver, whereas a secure asynchronous receiver connector contains a Decryption Agent security object that decrypts a cipher message to a plain message. These security objects are executed if application components require the confidentiality security service.

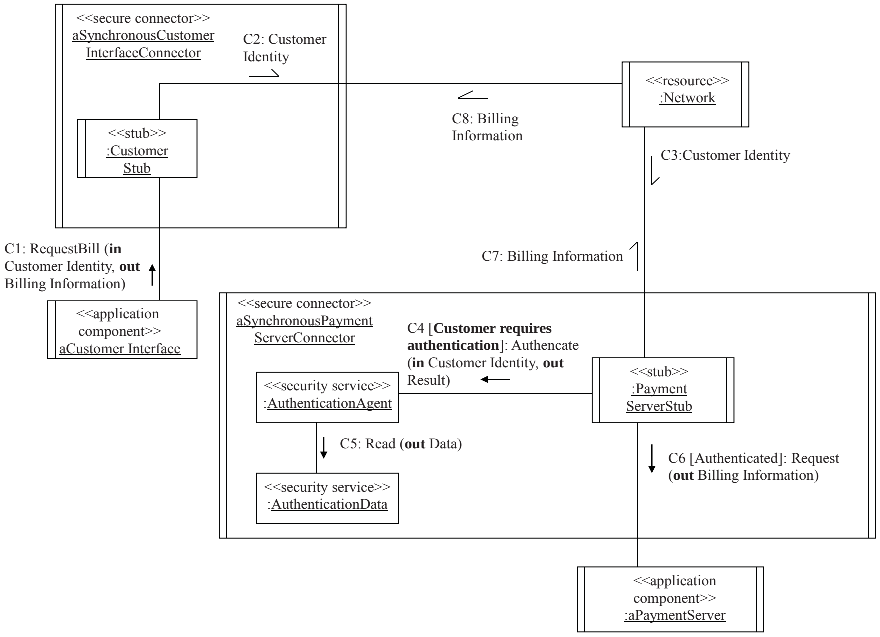


Fig. 2 Secure Synchronous Connector for Authentication security service

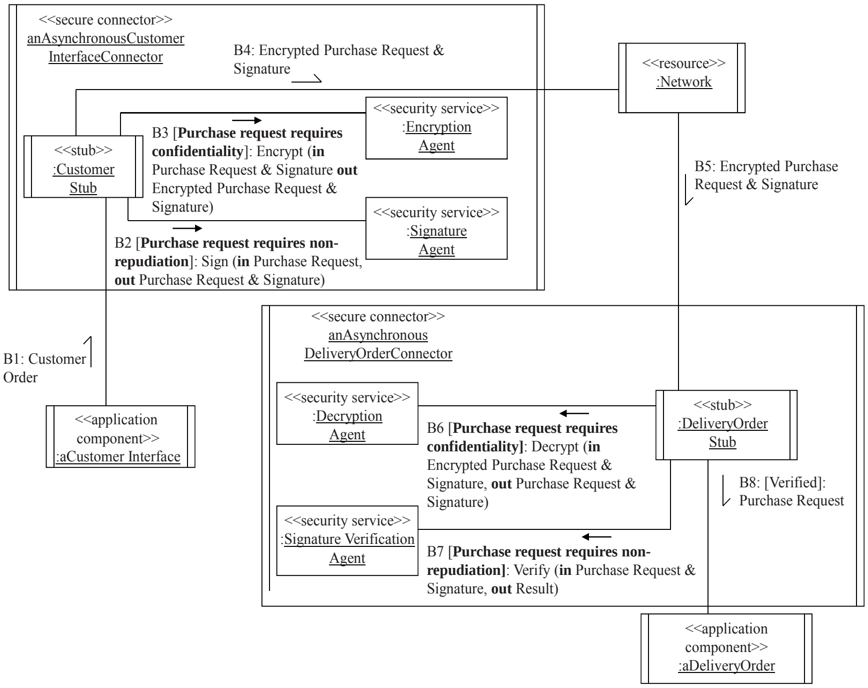

<<secure connector>>
anAsynchronousCustomer InterfaceConnector
B4: Encrypted Purchase Request & Signature
<<resource>>
:Network
<<security service>>
:Encryption Agent
<<stub>>
:Customer Stub
B3 [Purchase request requires confidentiality]: Encrypt (in Purchase Request & Signature out Encrypted Purchase Request & Signature)
<<security service>>
:Signature Agent
B2 [Purchase request requires non-repudiation]: Sign (in Purchase Request, out Purchase Request & Signature)
B5: Encrypted Purchase Request & Signature
<<secure connector>>
anAsynchronous DeliveryOrderConnector
B1: Customer Order
<<security service>>
:Decryption Agent
<<stub>>
:DeliveryOrder Stub
<<application component>>
:aCustomer Interface
B6 [Purchase request requires confidentiality]: Decrypt (in Encrypted Purchase Request & Signature, out Purchase Request & Signature)
B8: [Verified]: Purchase Request
<<security service>>
:Signature Verification Agent
B7 [Purchase request requires non-repudiation]: Verify (in Purchase Request & Signature, out Result)
<<application component>>
:aDeliveryOrder

Fig. 3 Secure Asynchronous Connector for Confidentiality and Non-repudiation security services

For an integrity security service, a secure asynchronous sender connector can have an Integrity Generation Agent security object to generate a message digest for a message being sent. The message digest is sent to the receiver along with the original message. A secure asynchronous receiver connector encapsulates the Integrity Checking Agent security object, which checks the integrity of the message by means of comparing the received message digest with the message digest generated by the receiver using the received message. These Integrity Generation and Checking security objects are performed if application components require an integrity security service.

In addition, a secure asynchronous sender connector can contain a Signature Agent security object that signs a message. The signature and original message are sent from the sender connector to the receiver connector. The secure asynchronous receiver connector encapsulates the Signature Verification Agent security object to prove the authenticity of the received signature. The signature for a message is stored at the Signature Verification Agent security object for later use. These security objects are activated if application components require a non-repudiation security service.

Fig. 3 depicts a secure asynchronous connector for confidentiality and non-repudiation security services, which are required for sending a purchase request from a customer to a supplier in the electronic commerce system. The secure asynchronous Customer Interface connector is structured with Encryption Agent and Signature Agent security objects. The secure asynchronous Delivery Order connector contains corresponding Decryption Agent and Signature Verification Agent security objects. Consider the following confidential purchase request scenario: The Customer Stub in the sender connector requests the Signature Agent security object to sign the purchase request with a key (message B2 in Fig. 3). Then the Customer Stub sends the signed purchase request to the Encryption Agent security object to encrypt the secret data with a key (message B3 in Fig. 3). The encrypted purchase request and signature are decrypted by the Decryption Agent security object (message B6 in Fig. 3), and then the signature is verified by the Signature Verification Agent security object (message B7 in Fig. 3).

4. Conclusions

This paper has described an approach to designing secure connectors that make software architectures secure for software applications. Secure connectors contain security objects to implement the security services separately from application components. These security objects are activated only if application components require the desired security services such as authentication, access control, confidentiality, integrity, and non-repudiation. Secure connectors have been designed by considering communication patterns between application components in software architecture. Secure connectors can make complex systems more maintainable by separation of security concerns from application concerns. A secure connector can be reused in different applications if it matches the security requirement and communication style between application components. To validate this approach, the secure connectors have been implemented in an electronic commerce application.

This paragraph describes future research for secure connectors. The security services provided by secure connectors described in this paper are confidentiality, integrity, non-repudiation, authentication, and authorization. Secure connectors can be extended to include an availability security service, which should be capable of preventing deliberate denial of services in a software application. In addition, security connectors can be specialized to ones that realize specific methods or algorithms. For example, a secure connector containing access control security service can be implemented with role-based access control or mandatory access control. To realize these specific methods or algorithms, a secure connector could be specialized to create the appropriate secure objects. In addition, secure connectors can be building blocks for composing secure software architectures along with application components. Future work needs more investigation on how software architectures for secure applications can be composed of secure connectors and application components.

References

[1] S. Banerjee, C. A. Mattmann, N. Medvidovic, and L. Golubchik, “Leveraging Architectural Models to Inject Trust into Software Systems,” Proceedings of the ICSE 2005 Workshop on Software Engineering for Secure Systems, St. Louis, Missouri, May, 2005.

[2] M. Moriconi, X. Qian, R. A. Riemenschneider, and Li Gong “Secure Software Architectures,” IEEE Symposium on Security and Privacy, 1997.

[3] Y. Deng, J. Wang, J. J. P. Tsai, and K. Beznosov, “An Approach for Modeling and Analysis of Security System Architectures,” IEEE Transactions on Knowledge and Data Engineering, vol.15, no.5, pp.1099-1119, Sept/Oct, 2003.

[4] H. Gomaa, D. A. Menasce, and M. E. Shin, “Reusable Component Patterns for Distributed Software Architectures,” Proceedings of ACM Symposium on Software Reusability, ACM Press, Pages 69-77, Toronto, Canada, May 2001.

[5] J. Ren, R. Taylor, P. Dourish, and D. Redmiles, “Towards An Architectural Treatment of Software Security: A Connector-Centric Approach,” Proceedings of the Workshop on Software Engineering for Secure Systems, St. Louis, Missouri, USA, May 15-16, 2005.

[6] G. Gong, I. Ray, and R. France, “Using Aspects to Design a Secure System,” Proceedings of the 8th IEEE International Conference on Engineering of Complex Computer Systems, December, 2002.

[7] H. Gomaa, M. E. Shin, “Modeling Complex Systems by Separating Application and Security Concerns” 9th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 2004), Italy, April, 2004.

[8] M. E. Shin and H. Gomaa, “Software Modeling of Evolution to a Secure Application: From Requirements Model to Software Architecture,” Science of Computer Programming, Volume 66, Issue 1, April 2007, pp. 60-70.

[9] H. Gomaa, “Software Modeling and Design: UML, Use Cases, Patterns, and Software Architectures”, Cambridge University Press, February 2011.

[10] G. Booch, J. Rumbaugh, and I. Jacobson, “The Unified Modeling Language User Guide,” Addison Wesley, 2nd Edition, 2005.

[11] J. Rumbaugh, G. Booch, and I. Jacobson, “The Unified Modeling Language Reference Manual,” Addison-Wesley, 2nd Edition, 2004.

[12] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, “Role-based access control models,” IEEE Computer, Volume 29, Issue 2, February 1996, pp. 38-47.

[13] C. P. Pfleeger, and S. L. Pfleeger, “Security in Computing,” Prentice-Hall, Inc., third edition, 2002.

[14] F. Buschmann, R. Meunier, H. Rohnert, and P. Sommerlad, "Pattern Oriented Software Architecture: A System of Patterns," John Wiley & Sons, 1996.

[15] R. N. Taylor, N. Medvidovic, and E. M. Dashofy, “Software Arcthiecture: Foundations, Theory, and Practice,” John Wiley & Sons, 2010.

[16] D. Bash, M. Clavel, and M. Egea, “A Decade of Model-Driven Security,” 16th ACM symposium on Access control models and technologies SACMAT11, Innsbruck, Austria, June 15-17, 2011.

How Social Network APIs Have Ended the Age of Privacy

Derek Doran, Sean Curley, and Swapna S. Gokhale
Dept. of Computer Science & Engineering
University of Connecticut, Storrs, CT, 06269
{derek.doran,smc08002,ssg}@engr.uconn.edu

Abstract

Online Social Networks (OSNs) have captured our imagination by offering a revolutionary medium for communication and sharing. Skeptics, however, contend that these OSNs pose a grave threat to privacy. This paper seeks to examine the veracity of this skepticism by analyzing the APIs of six popular OSNs for their propensity to violate user privacy. Our analysis lends substantial support to this skepticism by finding that OSNs: (i) facilitate an extensive collection of user information; (ii) provide default access to information of new users; (iii) do not seek comprehensive permissions; (iv) request these permissions ambiguously; and (v) offer privacy settings that enable only limited control.

1 Introduction and Motivation

Online Social Networks (OSNs) are now tightly knit into the fabric of our society. People are overwhelmingly participating in these OSNs regardless of their age, socioeconomic, and demographic status [10]; approximately 50% of all U.S. adults, and over 90% of all Gen-Y members use OSNs. Moreover, the growth in the user base is staggering, and is occurring in unconventional segments; recent data suggests a 90% increase in users over 50 years old, while only a 13% increase in users aged 18 to 29 [9].

OSNs can be designed for a variety of purposes. Some such as Facebook and Orkut offer a platform to stay connected with friends and acquaintances. Microblogging sites such as Twitter provide a modern news ticker to simultaneously stream current events and friends' updates [4]. OSNs also offer portals to share opinions and recommendations about products and brands; and users vastly affirm their trust in these peer recommendations over advertisements [1]. Many organizations thus seek to harness OSNs to raise brand awareness, build reputation, and stay relevant. Besides their commercial potential, OSNs can also serve as channels of societal transformation; the recent Egyptian political unrest may have been precipitated with the support of social media such as Facebook and Twitter [13].

The API of an OSN is a collection of public and open access methods for third parties to interact with the OSN. These APIs can be used to systematically harvest user information, and hence, commercial organizations exploit these APIs to collect user data for several purposes [2, 4, 14]. Most APIs also expose methods to enhance the usability of OSN services; the Twitter API enables posts from mobile devices, the YouTube API allows integration with TVs, and the Facebook API facilitates photo uploads directly from digital cameras. OSNs design their APIs to attract third parties to harvest user data or to integrate their social features into commercial products. Thus, through these APIs, OSNs seek to embed themselves deeper into the daily experiences of their users. These APIs thus need to be sophisticated for third parties to collect meaningful information to offer novel services. However, the higher the porosity of the APIs, the greater is the chance that third parties will misuse them to acquire unnecessary personal data and compromise user privacy. OSNs thus have to balance the conflicting priorities of providing a rich API against protecting user privacy.

OSNs balance these competing concerns through a careful consideration of three aspects in the design of their APIs. First, they build these APIs to expose a wealth of user information. However, they then cede control back to their users by offering a range of privacy settings on their accounts, and by requiring third parties to seek permissions from users before collecting their information. Users can protect their privacy through these settings and permissions by limiting the information that third parties can retrieve. Figure 1 shows how privacy settings and permissions form a protective barrier between access queries and user information.

Most users, especially certain groups such as seniors and minors, are very fascinated by the novel communication capabilities of OSNs, and do not appreciate their wide reach and impact [5]. They do not realize that their shared information may be used to their detriment by various agencies (law enforcement, insurance, and employers) and even criminals. For example, the Center for American Progress suggests that had the Egyptian movement been unsuccessful, the same OSNs that supported this movement would have provided adequate information to track down and persecute the protesters [13]. Even worse, they do not acknowledge the necessity to adjust and benefit from the privacy settings on their accounts [12]. Finally, third parties usually seek broader permissions than what are legitimately necessary for their services. Not mindful of the privacy implications, many users grant these permissions, bypassing this layer of protection as well. Thus, despite the mechanisms available to limit the exposure of information, OSN APIs can easily compromise users' privacy.

This paper presents a comparative analysis of the propensity of OSN APIs to violate user privacy. Ever since Facebook CEO Mark Zuckerberg famously defended the change to make user activity public-by-default, technologists have asserted that the "Age of Privacy" on the Web is over.[1] We seek to verify the truth in this assertion by examining the APIs of six popular and widely used OSNs. We find substantial support for this assertion because OSNs: (i) facilitate a collection of detailed user data; (ii) provide limited privacy settings and control; and (iii) use ambiguous requests to seek unnecessarily broad and general permissions.

The paper is organized as follows: Section 2 introduces OSN APIs and privacy control mechanisms. Section 3 presents our comparative analysis. Section 4 summarizes our findings and offers directions for future work.

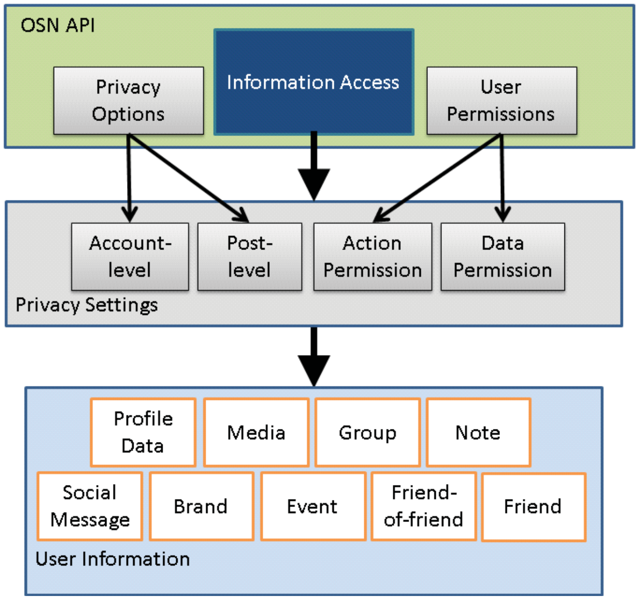


Figure 1. API access and privacy protection

2 OSN APIs: An Overview

Fundamentally, OSNs offer a platform for users to connect, interact, and share information. Each user is represented with a profile that contains the user's information. A *social feed* attached to the profile provides a forum to share information. OSN users establish connections; and updates to their social feeds are broadcast to their connections. Popular OSNs differ with respect to what constitutes user connections and the information shared via social feeds. User connections on Facebook, Orkut, and Google+ are bi-directional friendships, social feeds include status updates, photos, videos and posts by other users, and updates can be broadcast to all or a subset of users' connections. Twitter (YouTube) defines uni-directional relationships where users subscribe to receive other users' updates (video uploads).

Each OSN designs an API to allow third parties to collect information or to act on behalf of their users. These open, Web-based APIs can be queried with HTTP requests to retrieve a particular user's information. For example, the request `https://graph.facebook.com/220439` retrieves all the public information on Facebook's CTO. A request to `http://search.twitter.com/search.json?q=happy` returns recent tweets which include the word happy. Frequently, the APIs also return irrelevant information; for instance, the above Twitter call returns the name, user id, and a link to the user's photo for each tweet.

OSN users may expect that their shared information will be protected at the friendship, network, and public levels [6]. Friendship- and network-level protection prevents unintended disclosure, and users believe that these may be achieved through privacy settings. They may also presume public-level protection, because their notion of an OSN is a walled garden, impenetrable without authorization. Piercing through these privacy expectations, however, are the open access APIs. Most OSNs thus try to restore users' perception of privacy by empowering them with two forms of control: (i) configurable privacy settings; and (ii) express permission requests from third parties.

Understanding how the APIs of six popular OSNs balance these competing concerns of information access versus user privacy is the focus of this paper. We choose these OSNs because of their extreme popularity, widely varying purposes, and unique features as summarized in Table 1. We measure popularity in terms of the number of users[2] and include representative (not exhaustive) profile data.

3 Comparative Analysis

We extensively analyzed the online documentation, and aggregated the details of each OSN API along three dimensions: (i) user information that can be collected; (ii) privacy settings that can be configured; and (iii) permissions that may be requested.[3] For each dimension, we define the *coverage* metric as the percentage of the number of features that the API currently implements to the total number of features that it can *feasibly provide*, given the context and the functionality of the site. Because the total number of feasible features is determined by the union of the features across all APIs, it is inherently subject to our interpretation of the APIs. Therefore, we intend to use the coverage metric as a

[1] http://news.cnet.com/8301-17852_3-10431741-71.html

[2] http://vincos.it/social-media-statistics
http://royal.pingdom.com/2011/02/04/facebook-youtube-our-collective-time-sinks-stats

[3] API details are available at http://www.cse.uconn.edu/∼ded02007/osnapi.

OSN	Social Feed	Unique Features	Connections	Profile	# Users
Facebook	Status, media, friend posts, external sites, application alerts	Groups, events, notes, "Like" brands and Web pages, service integration	Bi-directional friendship	Name, contact info, family, education, work history, interests	845M+
Twitter	Tweets, retweets, mention tweets	Public access social feed	Uni-directional subscriber	Real name, location	200M+
Google+	Status, media, friend posts, external sites	Social circles, groups, "+1" brands and Web pages	Bi-directional friendship	Name, contact info, education, work history, interests	90M+
LinkedIn	Status, resume updates, discussion posts, application alerts	Groups, message boards, share professional recommendations	Bi-directional, professional connection	Name, work history, education, professional accomplishments, goals, skills	150M+
YouTube	Uploaded, subscribed, and favorite videos	Share video channels and playlists, public commentary on videos	Uni-directional subscriber	Age, country, occupation, education, interests	490M+
Orkut	Status, media, friend posts, application alerts	Integrated applications	Bi-directional friendship	Name, contact info, pets, family status, romance views, drinking habits	120M+

Table 1. Purpose, popularity (# of users), and features of OSNs

rough measure of the strength of an API along each dimension, and report their approximate rather than exact values. Next, we compare the APIs along these dimensions to understand their similarities and differences.

3.1 User information

We define nine categories to homogenize the disparity in the types of data that users can upload as seen from Table 1, in order to meaningfully compare the OSN APIs.

- **Profile**: Personal details such as hometown, date of birth, gender, and other information.
- **Social Message**: Text posted to users' social feeds, appearing in the news feeds of their connections.
- **Media**: Photos, videos, and music.
- **Friend**: First-degree connections.
- **Friend-of-friend**: Second-degree connections.
- **Brand Page**: A page dedicated to a branded entity, such as a company, a business, a movie, or a band.
- **Event**: Event details such as name, place, and time.
- **Group**: Membership details of users' groups.
- **Note**: A long post concerning a specific subject; typically more detailed than a social message.

The number of unique methods for each data type are: Profile (94), Social Message (10), Media (23), Friend (23), Friend-of-Friend (21), Brand Page (7), Event (7), Group (5), and Note (7). Table 2 lists the number of methods and coverage for each OSN. It shows that because Google+ implements only 9 methods for Social Messages, although it could feasibly implement 10, its coverage for this data type is 90%. If an OSN does not provide any method for a given data type, a 0 appears in the corresponding cell in the upper table (Google+, Friend-of-friend). Additionally, if a OSN does not implement any of the feasible methods for a data type, a 0 appears in the corresponding cell in the lower table (Google+, Friend-of-friend). Finally, when no methods are feasible for a given data type for a given OSN, a 0 appears in the upper table, and a "–" appears in the lower table (YouTube, Events).[4] For each data type, the number of distinct methods is less than the total number of methods across all OSNs; 92 distinct methods collect profile data, but a total of 209 profile access methods exist across all OSNs.

All APIs offer high coverage for profile data. This is expected from Facebook, Orkut, and LinkedIn because their primary purpose is sharing of personal information. Surprisingly, YouTube makes many user details available, although its main objective is open, public sharing. Twitter, Facebook, and YouTube APIs can retrieve nearly every attribute of media and social messages, leading to very high coverage. Although Google+ does not provide access to media, this is planned.[5] Most APIs only expose names and profile pictures (if any), of first- and second- degree connections. Ta-

[4] Tables 3 and 5 have the same interpretation of 0 and -.

[5] http://siliconfilter.com/google-gets-an-api-for-photos-and-videos/

ble 2 shows that all OSNs except LinkedIn provide limited coverage of connections, otherwise third parties can retrieve user data without their consent as long as their friends' data is accessible. The LinkedIn API, however, strongly covers second-degree connections. This accessiblility may be deliberate to support LinkedIn's main objective of mining connections to establish professional and business relationships. Thus, the LinkedIn API can be abused to collect significant user information without directly targeting them.

Number of methods

	Fb	T	G+	LI	YT	Ork
Profile	44	14	33	40	25	52
Message	9	9	9	5	6	6
Media	21	12	0	0	6	10
Friend	2	1	1	23	0	1
Friend-of-friend	0	1	0	21	0	0
Brand	7	0	6	0	0	0
Event	7	0	0	0	0	0
Group	6	1	3	0	0	0
Note	7	0	0	2	0	0

Coverage (%)

	Fb	T	G+	LI	YT	Ork
Profile	80	70	70	90	85	90
Message	90	90	90	100	85	60
Media	90	100	0	-	85	65
Friend	20	100	15	100	-	15
Friend-of-Friend	0	35	0	100	-	0
Brand	100	-	85	-	-	-
Event	100	-	-	-	-	-
Group	100	100	60	-	-	-
Note	100	-	-	100	-	-

Table 2. User information – # and coverage

3.2 Privacy Settings

OSN users can expect to control the spread and visibility of their information through privacy settings. Each setting is associated with a *privacy option* which indicates the pieces of data it covers. Moreover, a privacy option is exercised by assigning a *scope*, which identifies the audience of the data. Next, we study the privacy options and scopes of the APIs.

3.2.1 Privacy Options

Privacy options may apply either to a collection of users' data at the account level or to a specific data item such as a social message, a photo, or a video. *Account-level* options are broad, whereas, *item-level* options are narrow. Item-level options offer finer, second layer of protection on top of account-level options. For example, account-level options may allow public access to all Facebook wall posts, but an item-level option can restrict the visibility of a specific post. We identified 35 distinct options; 20 are account-level, and 15 are item-level. Table 3 lists these options and their coverage for each OSN.

Facebook and LinkedIn define a large number of account-level options to give users substantial control over how they can be searched and reached. LinkedIn users can decide which fields of their resume are publicly viewable, and who can send them a message. Similarly, Facebook users can choose the distance between themselves and other users who can send them friend requests, and whether their profile can be indexed by a search engine. Google+ and Orkut have weaker account-level options, and users can neither control who can share information with them nor can they decide how their profile can be searched. Orkut's limited account-level options is concerning, because it seeks very personal information as listed in Table 1. Finally, YouTube and Twitter can implement numerous account-level options given the type of information users share, and arguably these options may even be necessary. However, the lack of these account-level options may be intentional, because the purpose of these OSNs is open, public sharing.

Number of options

	Fb	T	G+	LI	YT	Ork
Account	11	0	6	15	3	4
Item	9	2	3	2	3	4

Coverage (%)

	Fb	T	G+	LI	YT	Ork
Account	70	0	50	85	25	35
Item	70	30	40	50	60	55

Table 3. Privacy options – # and coverage

Facebook and Google+ offer strong item-level options, and users can limit the audience of a wall post, and identify a group who can see an update. Other OSNs offer very few item-level options and implementation of some additional, feasible ones can easily enhance user privacy on these OSNs. For example, Twitter can implement important item-level options such as reviewing of tweets that mention a user, and restricting the reach of tweets to specific receivers.

3.2.2 Privacy Scopes

Very few users accept the notion that OSN activity is public-by-default, private-by-effort; 84% Facebook profiles set to full, default, public access [7] confirm this belief. Most users are also easily confused and frustrated by too many options and scopes and are therefore discouraged from changing their default settings [11]. Savvy users, however, need comprehensive options and scopes to adequately manage their privacy. This conflict poses a tradeoff between the number of options and scopes and their usability.

Table 4 lists the privacy scopes for each OSN; a (x)✓ indicates (un)availability of a scope. Only Twitter (YouTube) users can control the reach of their tweets (videos) to a particular individual; shown by a '✓' against their private scope. *User groups* scope, defined by Facebook, Google+, and Orkut, is similar but richer compared to the private scope because it allows the selection of a specific group rather than an individual. Facebook, LinkedIn, and Orkut offer additional scopes based on degrees of separation between the owner of the data and its viewers. For example, Facebook users can limit their photos to just friends, but make wall posts visible to friends-of-friends as well. LinkedIn users can share publicly, with only first-degree connections or up to third-degree connections. Sharing with third-degree connections may be equivalent to public sharing because most OSN users are separated by less than four degrees [3].

	Fb	T	G+	LI	YT	Ork
Up to 1st degree	✓	x	x	✓	x	✓
Up to 2nd degree	✓	x	x	x	x	x
Up to 3rd degree	x	x	x	✓	x	x
Public	✓	✓	✓	✓	✓	✓
Private	x	✓	x	x	✓	x
User Groups	✓	x	✓	x	x	✓

Table 4. Privacy scopes

3.3 User permissions

Permissions authorize third parties to access information that would otherwise be out of bounds because of users' privacy settings. All six OSNs use the OAuth protocol [8] to manage these permissions. This protocol represents a permission as an access token, which includes the specific information that is covered by the request, and a user-defined/OSN-specified expiration time. A dialogue displays a summary of this information to the user. Upon receiving user's consent, the dialogue requests the OSN to create this access token, which is then handed off to the requesting third party. Each OSN defines its permissions and customizes the dialogue presentation and information summary. We study these two attributes of user permissions across the APIs.

3.3.1 Permission types

We aggregate the permissions that third parties may request into two types: *data permissions* and *action permissions*. Data permissions grant access to users' social data, while action permissions allow third parties to perform tasks such as uploading media, changing a profile field, and posting to social feeds on behalf of users. Each type of permission has different implications; data permissions enable third parties to harvest user information, whereas, action permissions allow third parties to impersonate users. We identified 44 unique permissions; 29 data and 15 action. Table 5 shows the number of permissions and their coverage for each OSN.

Number of permissions

	Fb	T	G+	LI	YT	Ork
Data	28	7	15	0	0	1
Account	13	6	2	0	0	1

Coverage (%)

Data	95	80	80	-	-	20
Account	85	100	65	-	-	100

Table 5. Permission types – # and coverage

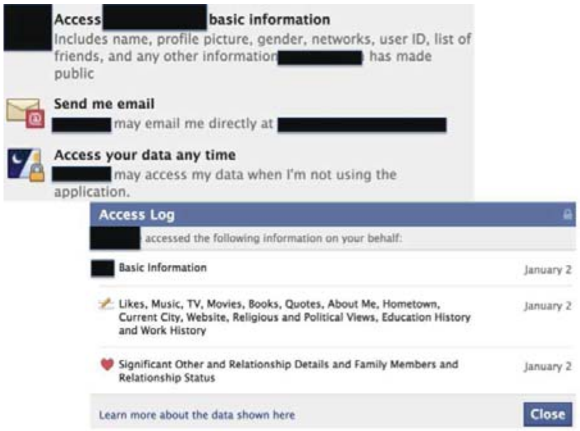
Facebook, Twitter, and Google+ exhibit high coverage for both types of permissions. On Facebook, third parties can request comprehensive permissions including authorizations to access data when users are offline, and to read the stream of updates to social feeds of users' connections. Although Twitter's purpose is public sharing, users can still protect their tweets through explicit permissions. Retweets and mention tags, however, can expose these tweets to third parties with access to the tweets of users' connections.

Orkut seeks a generic token which grants broad access to all the profile information, which does not have a friends-only privacy option. Third parties need to seek additional permissions to extend this generic token to post to users' social feeds or to access their photos. LinkedIn and YouTube users also cannot grant detailed data and action permissions, but instead grant a generic token to access all information that is not protected by a private scope. Thus, on LinkedIn the generic access token provides third parties with the potential to collect all personal information, and on YouTube it allows third parties to upload videos, and even browse lists of favorite and recommended videos. YouTube's generic access is worrisome because of its weak privacy control coverage as seen in Table 3. LinkedIn, however, limits its generic token by curtailing access to specific profile fields through comprehensive account-level privacy options.

3.3.2 Permission presentation

OSNs may seek broad data and action permissions from their users to enable them to enter into favorable agreements with third parties. Users, however, may be unwilling to grant such broad permissions, and OSNs may try to circumvent this reluctance through ambiguous permission dialogues.

Figure 2 displays a typical Facebook dialogue for an application that desires access to a user's profile for registration and login to an educational site for learning languages. The application requests permission to retrieve all "basic" Facebook data. The figure also shows that this application retrieved hometown, current city, religious views, work history, and even relationship details from the user's profile. Most of this basic information, however, is unnecessary for registration/login or to teach languages. This collection is additionally disturbing because many users probably do not expect their consent to expose so many private details.



[redacted] **Access [redacted] basic information**
Includes name, profile picture, gender, networks, user ID, list of friends, and any other information [redacted] has made public

Send me email
[redacted] may email me directly at [redacted]

Access your data any time
[redacted] may access my data when I'm not using the application.

Access Log

[redacted] accessed the following information on your behalf:

[redacted] Basic Information	January 2
Likes, Music, TV, Movies, Books, Quotes, About Me, Hometown, Current City, Website, Religious and Political Views, Education History and Work History	January 2
♥ Significant Other and Relationship Details and Family Members and Relationship Status	January 2

Learn more about the data shown here

Figure 2. Permission dialogue, accessed data

Except Twitter, the dialogues of all OSNs provide sketchy details.[6] Facebook and Google+ provide coarse summaries; Google+ users can retrieve details, but only after clicking on the summary. Most users will either not know or care to take this extra step without explicit instruction. LinkedIn and Orkut ambiguously caution users that their information will be shared with third parties and seek assent to terms of service linked from the dialogue. Finally, YouTube dialogues provide no details about the data that third parties can retrieve once users provide their consent.

4 Conclusions and Future Work

This paper reveals that OSN APIs expose significant user data, and substantially enable third parties to act on behalf of their users. Their privacy options are not comprehensive and can be exercised only with limited scopes. Permissions are often confusing and inconsistent, and their ambiguous presentation may mislead users into revealing excessive information. OSNs thus tip the scale towards information exposure versus privacy protection, lending ample support to critics' skepticism regarding user privacy.

Our future work seeks to quantify the privacy content in the information posted by different demographic groups. We also propose to develop tools and educational materials to raise public awareness about how third parties can exploit APIs to collect user data and how users can protect themselves by adopting proper measures.

References

[1] Personal recommendations and consumer opinions posted online are the most trust forms of advertising globally. The Nielsen Company Press Release, July 2009.

[2] A. Agarwal, B. Xie, I. Vovsha, O. Rambow, and R. Passonneau. Sentiment analysis of twitter data. In *Proc. of the Workshop on Languages in Social Media*, pages 30–38, 2011.

[3] L. Backstrom, P. Boldi, M. Rosa, J. Ugander, and S. Vigna. Four Degrees of Separation. Technical report, arXiv:1111.4570v3 [cs.SI], 2012.

[4] M. Cha, A. Mislove, and K. Gummadi. A Measurement-driven Analysis of Information Propagation in the Flickr Social Network. In *Proc. of 18th Intl. Conference on World wide web*, 2009.

[5] D. Boyd. Why youth (heart) social network sites: The role of networked publics in teenage social life. *Youth, Identity, and Digital Media*, pages 119–142, 2008.

[6] A. P. Felt and D. Evans. Privacy Protection for Social Networking APIs. In *Workshop on Web 2.0 Security and Privacy*, 2008.

[7] M. Gjoka, M. Kurant, C. T. Butts, and A. Markopoulou. Practical Recommendations on Crawling Online Social Networks. *IEEE Journal on Selected Areas in Communications*, 29(9):1872–1892, Oct. 2011.

[8] E. Hammer-Lahav, D. Recordon, and D. Hardt. The OAuth 2.0 Protocol. http://tools.ietf.org/html/draft-ietf- oauth-v2-10, 2010.

[9] P. S. R. A. International. Pew Internet & American Life Poll: Social Side of the Internet. Roper Center for Public Opinion Research Study USPEW2010-IAL12 Version 2, Nov. 2010.

[10] P. S. R. A. International. Pew Internet & American Life Poll: Spring Change Assessment Survey 2011. Roper Center for Public Opinion Research Study SPEW2011-IAL04 Version 2., Apr. 2011.

[11] H. R. Lipford, A. Besmer, and J. Watson. Understanding Privacy Settings in Facebook with an Audience View. In *Proc. of 1st Conference on Usability Psychology and Security*, 2008.

[12] Y. Liu, K. Gummadi, B. Krishnamurthy, and A. Mislove. Analyzing facebook privacy settings: user expectations vs. reality. In *Proc of ACM SIGCOMM Conference on Internet Measurement*, pages 61–70, 2011.

[13] P. Swire. Social Networks, Privacy, and Freedom of Association. Technical report, Center for Americal Progress, Feb. 2011.

[14] B. Viswanath, A. Mislove, M. Cha, and K. Gummadi. On the Evolution of User Interaction in Facebook. In *Proc. of 2nd ACM Workshop on Online Social Networks*, 2009.

[6] Images of these dialogues are available at: http://cse.uconn.edu/∼ded02007/osnapi

Computer Forensics: Toward the Construction of Electronic Chain of Custody on the Semantic Web

Tamer Fares Gayed, Hakim Lounis
Dept. d'Informatique
Université du Québec à Montréal
Montréal, Canada
gayed.tamer@courrier.uqam.ca
lounis.hakim@uqam.ca

Moncef Bari
Dept. d'Éducation et Pédagogie
Université du Québec à Montréal
Montréal, Canada
bari.moncef@uqam.ca

***Abstract*—The tangible Chain of Custody (CoC) document in computer forensics is the record of all information about how digital evidences are collected, transported, analyzed, and preserved for production. Chain of Custody plays a vital role in the digital forensic investigation process because it is used to prove the integrity of the evidences from seizure through production in court. The CoC integrity is the demonstration that the digital evidences have been properly copied, transported, and stored. With the advent of the digital age, the tangible CoC document needs to undergone a radical transformation from paper to electronic data (*e*-CoC), readable and consumed by machines and applications. Semantic web is a flexible solution to represent the tangible CoC document knowledge. It provides semantic markup languages for knowledge representation, supported by different provenance vocabularies to ensure the trustworthiness and the integrity of this represented knowledge. The semantic web is a web of data that are published and consumed according to the web aspects known as the data linked principles. This paper proposes the construction of *e*-CoC through a semantic web framework based on the same principles used to publish data on the web.**

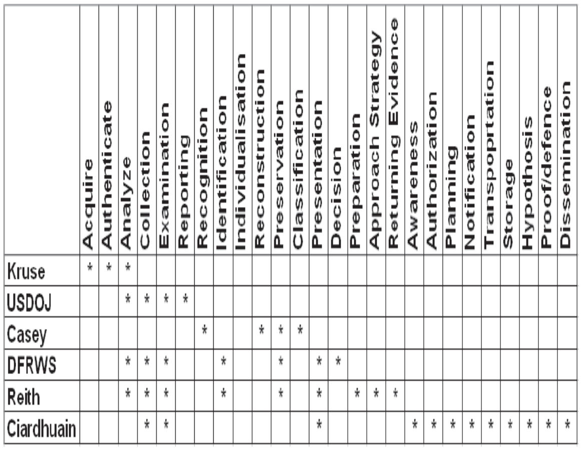
Keywords-Chain of Custody; Semantic Web; Semantic Markup Language; Provenance Vocabularies; Resource Description Framework.

I. INTRODUCTION

Computer/Digital forensic is a growing field. It combines computer science concepts including computer architecture, operating systems, file systems, software engineering, and computer networking, as well as legal procedures. At the most basic level, the digital forensic process has three major phases: Extraction, Analysis, and Presentation. Extraction (acquisition) phase saves the state of the digital source (ex: laptop, desktop, computers, mobile phones) and creates an image by saving all digital values so it can be later analyzed. Analysis phase takes the acquired data (ex: file and directory contents and recovering deleted contents) and examines it to identify pieces of evidence, and draws conclusions based on the evidences that were found. During presentation phase, the audience is typically the judges; in this phase, the conclusion and corresponding evidence from the investigation analysis are presented to them [1].

Nevertheless, there exists others forensic process models, each of them relies upon reaching a consensus about how to describe digital forensics and evidences [2][17]. The next table (Table-I) shows the current digital forensic models. Each row of the table presents the name of the digital forensic process model, while the columns present the processes included in each of these models.

TABLE I. DIGITAL FORENSIC PROCESS MODELS [17]



	Acquire	Authenticate	Analyze	Collection	Examination	Reporting	Recognition	Identification	Individualisation	Reconstruction	Preservation	Classification	Presentation	Decision	Preparation	Approach Strategy	Returning Evidence	Awareness	Authorization	Planning	Notification	Transpoprtation	Storage	Hypothosis	Proof/defence	Dissemination
Kruse	*	*	*																							
USDOJ			*	*	*	*																				
Casey							*			*	*	*														
DFRWS			*	*	*			*			*		*	*												
Reith			*	*	*			*			*		*		*	*	*									
Ciardhuain				*	*								*					*	*	*	*	*	*	*	*	*

Like any physical evidence, digital evidence needs to be validated for the legal aspects (admissibility) in the court of law. In order for the evidence to be accepted by the court as valid, chain of custody for digital evidence must be kept, or it must be known who exactly, when, and where came into contact with the evidence at each stage of the investigation [3].

The role of players (first responders, investigators, expert witnesses, prosecutors, police officers) concerning CoC is to (im)prove that the evidence has not been altered through all phases of the forensic process. CoC must include documentation containing answers to these questions:

- Who came into contact, handled and discovered the digital evidence?
- What procedures were performed on the evidence?
- When the digital evidence is discovered, accessed, examined or transferred?
- Where was digital evidence discovered, collected, handled, stored and examined?
- Why the evidence was collected?

- How was the digital evidence collected, used and stored?

Once such questions (i.e. known as 5Ws and the 1H) are answered for each phase in the forensic process, players will have a reliable CoC which can be then admitted by the judges' court.

On the other hand, the main aim of the semantic web is to publish data on the web in a standard structure, and manageable format [8]. Tim Berner Lee outlined the principles of publishing data on the web. These principles known as Linked Data Principles (i.e. LD principles):

- Use URI as names for things.
- Use HTTP URIs so that people can look up those names.
- When someone looks up a URI, provide useful information using the standards (RDF, SPARQL).
- Include RDF statements that link to other URIs so that they can discover related things.

This paper proposes the creation of electronic chain of custody (*e*-CoC) using a semantic web based framework that represent and (im)prove the classical/traditional paper-based CoC during the cyber forensics investigation. Semantic web will be a flexible solution for this task because it provides semantic markup languages such as Resource Description Framework (RDF), RDF Scheme (RDFS), and Web Ontology Language (OWL) that are used to represent knowledge.

In addition, the semantic web is rich with different provenance vocabularies [10], such as Dublin Core (DC), Friend of a Friend (FOAF), and Proof Markup Language (PML) that can be used to (im)prove the CoC by answering the 5Ws and the 1H questions.

Furthermore, the principles used to publish data on the web can also be applied on the represented knowledge. Thus, each CoC generated from each forensic phase in a forensic process can be represented and interlinked together using the web technology stack (i.e. URI and HTTP).

The remainder of this document is organized as follows: section 2 presents the state of the art related to this work. Section 3 provides the different challenges met by the tangible CoC documents. Section 4 gives a brief presentation about the semantic web, and the principles used to publish data on the web. In section 5, the proposed solution is explained, and finally, last section provides the conclusion.

II. State Of The Art

The state of the art related to this work can be summarized over three dimensions. The first dimension is the works on (im)proving the CoC. In [22], a conceptual Digital Evidence Management Framework (DEMF) was proposed to implement secure and reliable digital evidence CoC. This framework answered the who, what, why, when, where, and how questions. The 'what' is answered using a fingerprint of evidences. The 'how' is answered using the hash similarity to changes control. The 'who' is answered using the biometric identification and authentication for digital signing. The 'when' is answered using the automatic and trusted time stamping. Finally, the 'where' is answered using the GPS and RFID for geo location.

Another work in [23], discusses the integrity of CoC through the adaptation of hashing algorithm for signing digital evidence put into consideration identity, date, and time of access of digital evidence. The authors provided a valid time stamping provided by a secure third party to sign digital evidence in all stages of the investigation process.

Other published work to (im)prove the CoC is based on a hardware solution. SYPRUS Company provides the Hydra PC solution. It is a PC device that provides an entire securely protected, self contained, and portable device (i.e. connected to the USB Port) that provides high-assurance cryptographic products to protect the confidentiality, integrity, and non-repudiation of digital evidence with highest-strength cryptographic technology [15]. This solution is considered as an indirect (im)proving of the CoC as it preserves the digital evidences from modification and violation.

The second dimension concerns knowledge representation. An attempt was performed to represent the knowledge discovered during the identification and analysis phase of the investigation process [26]. This attempt uses the Universal Modeling Language (UML) for representing knowledge. It is extended to a unified modeling methodology framework (UMMF) to describe and think about planning, performing, and documenting forensics tasks.

The third dimension is about the forensic formats. Over the last few years, different forensic formats were provided. In 2006, Digital Forensics Research Workshop (DRWS) formed a working group called Common Digital Evidence Storage Format (CDEF) working group for storing digital evidence and associated metadata [12]. CDEF surveyed the following disk image main formats: Advanced Forensics Format (AFF), Encase Expert Witness Format (EWF), Digital Evidence Bag (DEB), gfzip, ProDiscover, and SMART.

Most of these formats can store limited number of metadata, like case name, evidence name, examiner name, date, place, and hash code to assure data integrity [12]. The most commonly used formats are described here.

AFF is defined by Garfinkel et al. in [27] as a disk image container which supports storing arbitrary metadata in single archive, like sector size or device serial number. The EWF format is produced by EnCase's imaging tools. It contains checksums, a hash for verifying the integrity of the contained image, and error information describing bad sectors on the source media.

Later, Tuner's digital evidence bags (DEB) proposed a container for digital crime scene artifacts, metadata, information integrity, and access and usage audit records [5]. However, such format is limited to name/value pairs and makes no provision for attaching semantics to the name. It attempts to replicate key features of physical evidence bags, which are used for traditional evidence capture.

In 2009, Cohen et al. in [4] have observed problems to be corrected in the first version of AFF. They released the AFF4 user specific metadata functionalities. They described the use of distributed evidence management systems AFF4 based on an imaginary company that have offices in two different countries. AFF4 extends the AFF to support multiple data sources, logical evidence, and several others (im)provements such the support of forensic workflow and the storing of arbitrary metadata. Such work explained that the Resource Description Framework (RDF) [7] resources can be exploited with AFF4 in order to (im)prove the forensics process model.

III. CoC Challenges

Chain of custody is a testimony document that records all information related to the evidences (digital/physical) in order to ensure that they are not altered throughout the forensic investigation. It contains information about who handled the evidence, why there was a change of possession and how each person safeguards it. Failure to record enough information related to the evidence may lead to its exclusion from the legal proceedings.

The continuous growing of devices and software in the field of computing and information technology creates challenges for the cyber forensics science in the volume of data (i.e. evidences) being investigated. It also increases the need to manage, process, and present the CoC in order to minimize and facilitate its documentation.

The second issue is related to the interoperability between digital evidence and its CoC documentation. Last works concentrated mainly on the representation and correlation of the digital evidences [24][25] and as an indirect consequence, the (im)proving of the CoC by attempting to replicate the key features of physical evidence bags into Digital Evidence Bags (DEB) [5]. However, the documentation of CoC for digital evidences remains an exhausted task. Knowledge communication between the digital evidence and the information documentation about the evidence, apart from natural language, can create some automation and minimize human's intervention.

The third issue concerns the CoC documents. They must be affixed securely when they are transported from one place to another. This is achieved using a very classical way: seal them in plastic bags (i.e. together with physical evidence if it exists, such as hard disk, USB, cables, etc.), label them, and sign them into a locked evidence room with the evidences themselves to ensure their integrity.

The fourth issue is about the judges' awareness and understanding that the digital evidences are not enough to evaluate and take the proper decision about the case in hand. One solution is to organize a training program to educate the juries the field of Information and Communication Technology (ICT) [6]. From the point of view of the authors, this will not be an easy task to teach juries the ICT concepts. The other solution is to provide a descriptive *e*-CoC using forensic and provenance metadata that the juries can query and so, find the answers to their questions through these metadata.

The last issue is that the problem is not only to represent the knowledge of the tangible CoC in order to solve the issues mentioned above, but also to express information about where the CoC information came from. Juries can find the answers to their questions on the CoC, but they need also to know the provenance and origins of those answers. Provenance of information is crucial to guarantee the trustworthiness and confidence of the information provided. This paper distinguishes between forensic information and provenance information. Forensic information is responsible to answer the 5Ws and 1H questions related to the case in hand, while provenance information is responsible to answer questions about the origin of answers (i.e. what information sources were used, when they were updated, how reliable the source was).

IV. The Web Of Data

Semantic web is an extension of the current web, designed to represent information in a machine readable format by introducing knowledge representation languages based in XML. The semantic markup language such as Resource Description Framework (RDF), RDF Schema (RDFS), and the web ontology language (OWL) are the languages of the semantic web that are used for knowledge representation.

According to the W3C recommendation [7], RDF is a foundation for encoding, exchange, and reuse of structured metadata. RDF supports the use of standards mechanisms to facilitate the interoperability by integrating separate metadata elements (vocabularies) defined by different resource description communities (e.g. Dublin Core).

RDF consists of three slots: resource, property/predicate, and object. Resources are entities retrieved from the web (e.g. persons, places, web documents, picture, abstract concepts, etc). RDF resources are represented by uniform resource identifiers (URI) of which URLs are a subset. Resources have properties (attributes) that admit a certain range of value or attached to be another resource. The object can be literal value or resources.

In 2006, Berners Lee outlined set of rules for publishing data on the web using the LD principles. These rules explain that the data (content/resources) should be related one another just as documents are already. The semantic web extended these web principles from document to data and provided the RDF framework that allows data to be shared on the web.

With the linking data principles, entities/resources are identified with the HTTP URIs that can be resolved over the Web. Data itself are represented using the RDF generic model where each element (i.e. subject, predicate, and object) in this model can be URI for unnamed entities. The property/predicate in the RDF model describes how the subject is related to the object and it is defined in vocabularies. The late can also be represented as RDF data and be published as linked data. The HTTP URIs enables applications to retrieve and consume the data. RDF data are related to each others thought RDF links where the subject is a URI of one data source and the object is a URI in the name-space of another data source. Thus, RDF graphs retrieved by resolving URI references are called the linked data object.

The example in next figure (figure-1) [28] shows that the interlink of three datasets and the object of an RDF graph (dbpedia:berlin) can be the subject of another RDF graph related through by predefined predicate name-spaces (i.e. URI: http://dbpedia.org/resource/Berlin).

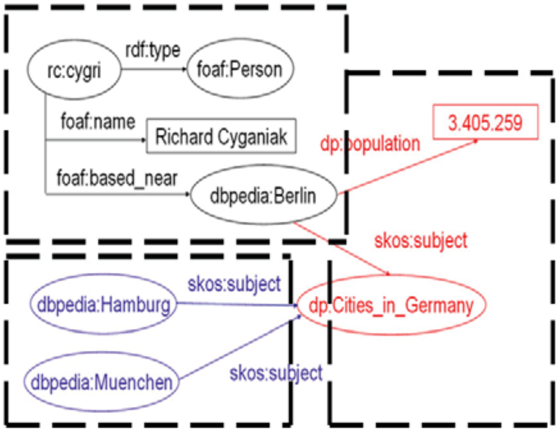


Figure 1. Example RDF Linked Data Example [28]

The Linking Open Data (LOD) project is the most visible project using this technology stack (URLs, HTTP, and RDF) and converting existing open license data on the web, into RDF according to the LOD principles [9].

The LOD project created a shift in the semantic web community and changed the researches wheel of the semantic web. Pre-2007, the concern was concentrated on defining the semantics and the capability to capture the meanings on the semantic web (i.e. languages to represent them, logics for reasoning with them, methods and tools to construct them), but post-2007, the researches on the semantic web are mainly concentrated on the web aspects (i.e. how to publish and consume the data on the web).

Moreover, Semantic web provides provenance vocabularies that enable providers of web data to publish provenance related metadata about their data. Provenance information about a data item is information about the history of the item, starting from its creation, and including information about its origins. Provenance information of Web data must comprise the aspect of publishing and accessing data on the Web. Providing provenance information as linked data requires vocabularies that can be used to describe the different aspects of provenance [11][10][13][14].

V. The Proposed Framework

The solution framework is about the use of the semantic web to represent the CoC using RDF and (im)prove its integrity through different built in provenance vocabularies. Thus, the CoC forensic information and its provenance metadata will be published and consumed on the web.

There exist various vocabularies to describe provenance information with RDF data. The popular standard metadata that can be used in different contexts is the Dublin core metadata terms defined in the RDFS schema [19]. The main goal of consuming this provenance metadata is to assess the trustworthiness and quality of the represented knowledge.

As mentioned in section 2, digital evidence can be stored in open or proprietary formats (e.g. CDEF, AFF, EWF, DEB, gfzip, ProDiscover, and SMART). These formats store forensic metadata (e.g. the sector size and device serial number). The most advanced format for representing the digital evidence is the AFF4 which is an extension of the AFF to accommodate multiple data sources, logical evidence, arbitrary information, and forensic workflow [16].

The framework proposed in Figure-2 shows that the tangible CoC can be created manually from the output of forensic tools (e.g. AFF4 or any other format). AFF4 can be modeled into RDF. Human creates the CoC according to a predefined form determined by the governmental/commercial institution and fill the forms from the forensic information synthesized from the forensic tools. Experience can be used, if necessary, to prune or add some forensic metadata not provided on the current output format.

This framework is generalized to all phases of the forensic investigation. As we have different forensic models with different phases, the framework can be adapted for different phases of different models (see table I). A summary of different process models can be found in [2][17]. Each phase for specific forensic process model has its own information: forensic metadata, forensic algorithm, player who came into contact, etc.

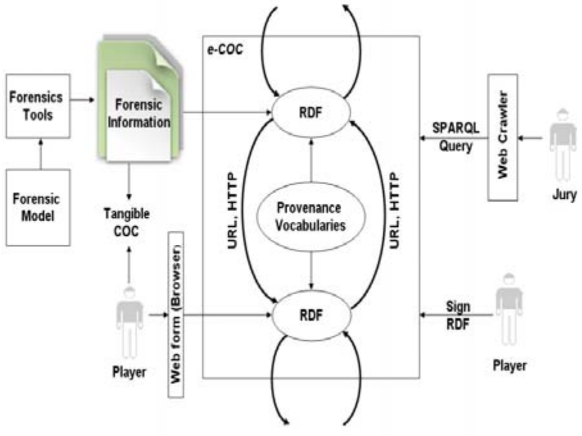


Figure 2. Framework for representing and (im)proving CoC using semantic web

The AFF4 can be directly represented using the RDF. Researchers have proposed several solutions on the use of AFF4 and RDF resources to (im)prove digital forensics process model or software. The tangible CoC associated to each phase/digital evidences can be also represented in RDF. Players of each phase can enter the necessary information through a web form interface which is then transformed to RDF triple using web service tool (e.g. triplify) [18]. The RDF data are supported by different build in provenance vocabularies like DC [19], FOAF [20], and Proof Markup Language (PML) [21]. For example the provenance terms used by the DC are:

dcterms:contributor, dcterms:creator, dctemrs:created, dcterms:modified, dcterms:provenance, which can give the juries information about who contributed, created, modified, the information provided to the court.

FOAF provides classes and properties to describe entities such as persons, organizations, groups, and software agents.

The Proof Markup Language describes justifications for results of an answering engine or an inference process.

CoC representation for each phase in RDF data can be linked with another, using the same principle of the LOD project (i.e. RDF graph/statement can be linked and be navigated using the semantic technology stack: URLs, HTTP, and RDF). Digital evidence may be also integrated with its CoC information. After representing all information related to the digital evidence and its associated CoC, the player who comes into work in this phase can finally sign his RDF data. Finally, we will have an interlinked RDF based on the LOD principle which represents the whole *e*-CoC of the case in hand.

Juries can use application based on the same idea of the web crawler; they can not only navigate over the interlinked RDF graph/statement, but also, run query through a web application over the represented knowledge using SPARQL query language, and find the necessary semantic answers about their forensic and provenance questions.

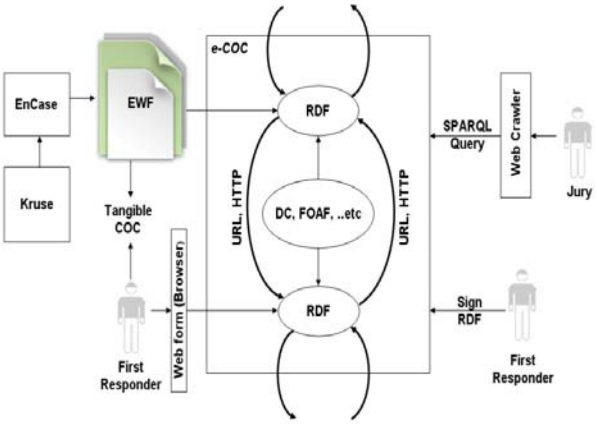


Figure 3. The application of the Framework for the authentication phase in the Kruse DFPM

An example on this framework is shown in the last figure (Figure-3). The authenticate phase of the Kruse DFPM (i.e. this model consists of 3 phases: acquire, authenticate and analyze-see Table-I). The authentication in digital forensics is usually done by comparing data of the original MD5 hash with the copied MD5 hash.

The digital format is the EWF as mentioned in section 2 (i.e. produced by EnCase's imaging tools) contains checksums, a hash for verifying the integrity of the contained image, and error information describing bad sectors on the source media. Players that come into role in the authentication phase of the Kruse DFPM can be one of the following prosecution, defense, or first responder.

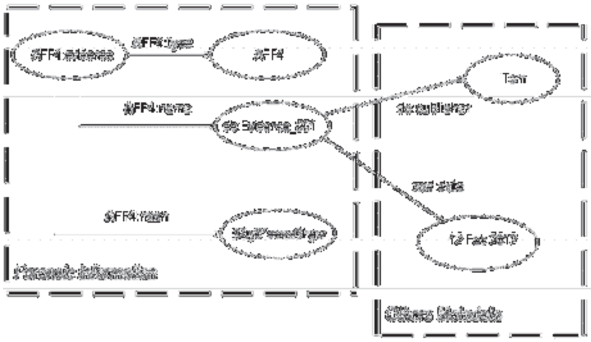


Figure 4. Interlinked Forensic RDF and Metatada

Figure-4 reflects the same idea as figure-1. In this figure, the forensic information is represented using RDF graph. The example used here is using the AFF4; it is more advanced than the EW4 and also can be synthesized from different forensic toolkits. We imagined according to the proposed framework that the forensic information/digital evidence can be supported with other build in vocabularies metadata. Last figure shows that the Dublin core is used to specify the publisher of this evidence and another build in scheme like the XML schema was also used to specify the date of publication.

The proposed framework can provide solutions to the previously mentioned issues. Representation of the tangible CoC knowledge in RDF facilitates the management and processing, because it is a machine readable form (first issue). It is also interoperable; digital evidence representation and its CoC description can be unified together under the same framework (i.e. RDF). Also, each player comes into role can secures (i.e. using cryptographic approaches) and signs his RDF data (i.e. using digital signature) to ensure the integrity and identity, respectively (second issue and third issue). On the other hand, juries can consume and navigate over the interlinked RDF data which present the whole and detailed information about the history of evidence from its collection to its presentation in the court (fourth issue). Provenance vocabularies can also be used to provide extra and descriptive metadata beyond the forensic metadata provided by the forensic tools (last issue).

VI. Conclusion

This paper proposes the construction of a semantic web based framework to represent and (im)prove tangible chain of custody using RDF and provenance vocabularies. This framework will be based on the same principles used to publish data on the web. The first phase of this framework will be the definition and analysis of all related information (metadata) for each phase in a selected forensic process (i.e. source will be the human experience and forensic tools output). Secondly, we will focus on the conversion and representation of tangible CoCs information into interlinked RDF (*e*-CoCs). This representation will contain forensic and provenance metadata (built-in/custom) related to the case in hand. The last phase will be the construction of a web interface that let the juries consume and query these interlinked RDF data in order to answer all questions related to the CoCs of evidences and their provenances.

References

[1] Erin Kenneally. Gatekeeping Out Of The Box: Open Source Software As A Mechanism To Assess Reliability For Digital Evidence. Virginia Journal of Law and Technology. Vol 6, Issue 3, Fall 2001.

[2] Michael W. Andrew "Defining a Process Model for Forensic Analysis of Digital Devices and Storage Media" Proceedings of the 2nd International Workshop on Systematic Approaches to Digital Forensic Engineering SADFE 2007

[3] Ćosić, J., Bača, M. Do we have a full control over integrity in digital evidence life cycle, Proceedings of ITI 2010, 32nd International Conference on Information Technology Interfaces, Dubrovnik/Cavtat, pp. 429-434, 2010

[4] COHEN, M.; GARFINKEL, S.; SCHATZ, B. Extending the advanced forensic format to accommodate multiple data sources, logical evidence, arbitrary information and forensic workflow. Digital Investigation, 2009. S57-S.

[5] Turner, P. Unification of Digital Evidence from Disparate Sources (Digital Evidence Bags). In 5th DFRW. 2004. New Orleans

[6] Judges' Awareness, Understanding and Application of Digital Evidence, Phd Thesis in computer technology in Education, Graduate school of computer and information sciences, Nova Southeastern University, 2010

[7] RDF: Model and Syntax Specification. W3C recommendation, 22 Februrary 1999, www.w3.org/TR/REC-rdf-syntax-19990222/1999

[8] The semantic web, Linked and Open Data, A Briefing paper By Lorna M. Campbell and Sheilla MacNeill, June 2010, JISC CETIS

[9] Christian Bizer, Tom Heath, Tim Berners-Lee: Linked Data - The Story So Far. Int. J. Semantic Web Inf. Syst. 5(3): 1-22 (2009)

[10] Olaf Hartig: Provenance Information in the Web of Data. In Proceedings of the Linked Data on the Web (LDOW) Workshop at the World Wide Web Conference (WWW), Madrid, Spain, Apr. 2009

[11] Olaf Hartig and Jun Zhao: Publishing and Consuming Provenance Metadata on the Web of Linked Data. In Proceedings of the 3rd International Provenance and Annotation Workshop (IPAW), Troy, New York, USA, June 2010

[12] CDESF. Common Digital Evidence Format. 2004 [Viewed 21 December 2005]; Available from: http://www.dfrws.org/CDESF/index.html

[13] [Olaf Hartig and Jun Zhao: Using Web Data Provenance for Quality Assessment. In Proceedings of the 1st Int. Workshop on the Role of Semantic Web in Provenance Management (SWPM) at ISWC, Washington, DC, USA, October 2009 Download PDF

[14] Olaf Hartig, Jun Zhao, and Hannes Mühleisen: Automatic Integration of Metadata into the Web of Linked Data (Demonstration Proposal). In Proceedings of the 2nd Workshop on Trust and Privacy on the Social and Semantic Web (SPOT) at ESWC, Heraklion, Greece, May 2010

[15] Solvinghe digital Chain of Custody Problem, SPYRUS, Trusted Mobility Solutions, © Copyright 2010

[16] M. I. Cohen, Simson Garfinkel and Bradley Schatz, Extending the Advanced Forensic Format to accommodate Multiple Data Sources, Logical Evidence, Arbitrary Information and Forensic Workflow, DFRWS 2009, Montreal, Canada

[17] MD Köhn, JHP Eloff and MS Olivier, "UML Modeling of Digital Forensic Process Models (DFPMs)," in HS Venter, MM Eloff, JHP Eloff and L Labuschagne (eds), Proceedings of the ISSA 2008 Innovative Minds Conference, Johannesburg, South Africa, July 2008 (Published electronically)

[18] http://triplify.org/Overview

[19] http://dublincore.org/

[20] http://www.foaf-project.org/

[21] P. P. da Silva, D. L. McGuinness, and R. Fikes. A Proof Markup Language for Semantic Web Services. Information Systems, 31(4-5):381–395, June 2006

[22] Ćosić, J., Bača, M. (2010) A Framework to (Im)Prove Chain of Custody in Digital Investigation Process, Proceedings of the 21st Central European Conference on Information and Intelligent Systems, pp. 435-438, Varaždin, Croatia

[23] Ćosić, J., Bača, M. (2010) (Im)proving chain of custody and digital evidence integrity with timestamp, MIPRO, 33. međunarodni skup za informacijsku i komunikacijsku tehnologiju, elektroniku i mikroelektroniku, Opatija, 171-175

[24] Schatz, B., Mohay, G. And Clark, A. (2004) 'Rich Event Representation for computer Forensics', Proceedings of the 2004 Asia Pacific Industrial Engineering and Management System

[25] Schatz,B., Mohay, G. and Clark, A.(2004) 'Generalising Event Forensics Across Multiple domains' Proceedings of the 2004 Australian Computer Network and Information Forensics Conference (ACNIFC 2004), Perth, Australia.

[26] Bogen, A. and D.Dampier. Knoweldge discovery and experince modeling in computer forensics media analysis.In international Symposium on Information and Communication Technologies.2004: Trinity College Dublin

[27] Garfinkel, S.L., D.J. Malan, K.-A. Dubec, C.C. Stevens, and C. Pham, Disk Imaging with the Advanced Forensics Format, library and Tools. Advances in Digital Forensics (2nd Annual IFIP WG 11.9 International Conference on Digital Forensics), 2006

[28] http://www4.wiwiss.fu-berlin.de/bizer/pub/linkeddatatutorial/

A Holistic Approach to Software Traceability

Hazeline U. Asuncion
Computing and Software Systems
University of Washington, Bothell
Bothell, WA 98011 USA
hazeline@u.washington.edu

Richard N. Taylor
Institute for Software Research
University of California, Irvine
Irvine, CA 92697 USA
taylor@ics.uci.edu

Abstract—**Software traceability is recognized for its utility in many software development activities. Nonetheless, applying traceability in practice is generally difficult due to interrelated factors such as high overhead costs, the presence of heterogeneous artifacts and tools, and low user motivation. In addition, many traceability techniques tend to only address difficulties from either the economic, technical, or social perspective, but not all three. To holistically tackle these three dimensions, we created a traceability technique, Architecture-Centric Traceability for Stakeholders (ACTS). The ACTS technique connects distributed and varied artifacts around concepts represented by the architecture, enables stakeholders to trace to artifacts that are of interest to them, and captures traceability links prospectively in the background as stakeholders perform development tasks. Our case studies of the ACTS tool usage reveal that it is possible to simultaneously address economic, technical, and social difficulties, suggesting that such an approach can aid the adoption of traceability in industry.**

Keywords-documentation; software traceability; software architecture; open hypermedia

I. Introduction

In a typical software development setting, numerous artifacts are produced throughout the software lifecycle. Software traceability aims to identify and explicitly link together related artifacts to support various software engineering tasks, and is useful in system comprehension, impact analysis, and system debugging [18, 26]. Despite these benefits, implementing novel traceability methods within an industry setting is often very difficult [20]. Manual approaches are tedious and error-prone; as a result, software engineers have an aversion to traceability tasks [18]. Automated techniques often require human intervention [15].

We posit that the difficulties with transitioning traceability techniques to real-world settings stem from a narrow understanding of the challenges around traceability. In fact, many interacting factors hinder traceability, such as high costs [18], complex interrelationships between artifacts [2], heterogeneity of artifacts and tools [14], and varied stakeholder interests [13, 29]. These factors, which reflect economic, technical, and social perspectives, must all be addressed to realize the benefits of traceability.

This paper presents a novel technique to traceability that aims to holistically address interacting challenges from these three perspectives. Our technique, Architecture-Centric Traceability for Stakeholders (ACTS) advances the state-of-the-art in traceability by (1) connecting distributed and varied artifacts to concepts represented by the *architecture,* (2) enabling *stakeholders* to trace to artifacts that are of interest to them, and (3) capturing traceability *prospectively, in situ,* as software engineers perform development tasks.

In previous work, we discussed traceability challenges from the three perspectives within the specialized context of one organization [6]. ACTS provides a generalized and automated technique for holistically addressing traceability challenges, using open hypermedia and rules to generate trace links. More recently, we combined architecture-centric traceability with a machine learning method known as topic modeling [5]; however, that work did not focus on holistic traceability. In this work, we focus on the rationale and methodology behind ACTS and perform case studies that suggest that our approach has utility from economic, technical, and social perspectives.

In the next section, we discuss the importance of analyzing the challenges to traceability from the three perspectives and survey existing techniques. Section 3 presents the main elements of ACTS. Section 4 covers the ACTS tool support. Section 5 discusses three case studies of tool usage, and Section 6 concludes with future work.

II. Problem Analysis and Existing Techniques

Traceability challenges can be viewed from economic, technical, and social perspectives, which we describe below.

A. Economic Perspective

The economic perspective focuses on the cost of supporting traceability. Costs can be in terms of labor hours [18], documentation, and user training time [6]. High cost is one of the major hindering factors to traceability [18]. Techniques to minimize cost include examining the trade offs between cost and quality or between cost and benefit [12].

B. Technical Perspective

The challenges from the technical perspective include the complexity of tracing due to the heterogeneity of artifacts and tools [14, 21], the combinatorial explosion of the artifact space [20], and the continuous and independent evolution of artifacts.

There are several approaches that tackle these technical challenges. Techniques to address the heterogeneity of artifacts include natural language processing [8], model transformation [19], and translation techniques [2]. Techniques to address the heterogeneity of artifacts and tools include

This material is based in part upon work supported by the National Science Foundation under Grant No. 0917129.

shared repository and specialized code [6]. Open hypermedia adapters can also be used to create traceability links across tool boundaries [3]; we build upon this technique to automatically capture traceability links. Information retrieval techniques may be used to address the explosion of the artifact space [9, 11, 15]. Finally, centrally managing all the artifacts in one tool [24] may be used to address traceability link maintenance.

C. Social Perspective

The social perspective has equally important challenges that arise from the interaction of stakeholders with traceability, such as differing expectations and low user motivation. It is recognized that people play a crucial part in determining the quality of traceability links [4, 15]. Because of different stakeholder expectations [13], it is difficult to create a general traceability tool that caters to these different stakeholders. This difficulty can be addressed by developing customized tools [6, 26]. In addition, software engineers generally have low motivation to perform traceability tasks [18]. Providing incentives to engineers or integrating traceability tasks into the development process may address these challenges [4, 6, 25].

D. Perspectives Interplay

We argue that the challenges from the economic, technical, and social perspectives are interrelated. As an example, let us consider the following scenario. Because of the high cost of manually capturing traceability links, an organization may resort to automated techniques to capture traceability links (e.g., [11]). However, since automated techniques cannot achieve 100% accuracy [9], human analysts must still check the recovered traces, which can be tedious [15]. Since the number of traces to be examined can be high [20], manual checking could lead to additional overhead costs. Thus, simply addressing the challenges from one perspective is inadequate.

III. ADDRESSING THE CHALLENGES WITH ACTS

To holistically address the traceability challenges, we present Architecture-Centric Traceability for Stakeholders (ACTS). The key elements of ACTS are centering the links to the architecture, enabling stakeholders to trace artifacts that are of interest to them, and capturing links prospectively. We focus on addressing the challenges of the cost of capturing traceability links (economic), ability to link across heterogeneous tools and artifacts (technical), and low user motivation in performing traceability tasks (social).

A. Architecture-Centric Traceability

To address the technical challenges of tracing across heterogeneous artifacts, we chose to center the traceability links to the architecture, since it provides higher level concepts than the code [12] and it has a more rigorous representation than requirements [15]. There is also an inherently close relationship between the architecture and other artifacts in software development: e.g., between requirements and architecture [22], between architecture and source code [23], and between architecture and test cases [17]. Moreover, the architecture itself contains information that aids in understanding the system and its related artifacts. The software architecture provides a comprehensive view of the system, enabling engineers to better understand the system in its entirety as well as in its individual computational units. This understanding lends itself to understanding the traceability links from the architecture to the various software artifacts. It also aids in understanding the links across different system versions and across software product lines [16]. Links centered on the architecture can be updated to reflect system evolution.

We define software architecture broadly, as the set of principal design decisions about a software system [28]. This definition implies that every software system has an architecture—some set of design decisions that were made in its development. Principal design decisions can be expressed as the system's structure, functional behavior, interaction, and non-functional properties; this paper focuses on decisions as expressed in the system's structure, e.g., the functional units, the mode of communication between the functional units, and the configuration of these units. These principal design decisions provide product-based concepts by which traceability can be anchored and supported.

B. Stakeholder-driven Traceability

To address the social challenge of low motivation, we enable different stakeholders to both capture and use the traceability links they capture, which we call stakeholder-driven traceability. Stakeholders are individuals who have a vested interest in capturing relationships between software artifacts. Oftentimes, the stakeholder who captures the traceability links is not the same stakeholder who uses the links, causing a separation between the work and the incentive to perform the work [4]. We believe, however, that it is imperative for stakeholders who invested time in capturing the traceability links to also be able to use the links.

To achieve stakeholder-driven traceability, the tool support must satisfy the following requirements. First, the tool must cater to varied stakeholder interests in capturing traceability links. For example, a requirements analyst may be interested in tracing high level to low level requirements while a maintenance engineer may be interested in tracing architecture to source code. Thus, the tool support must decouple the heuristics of determining the traceability links from the tool itself, to allow stakeholders to specify which artifacts to trace. Secondly, the tool must enable stakeholders to use the traceability links they captured. To be usable, the links should be accessible (i.e., traced artifacts are viewable within the context of its native editor), updateable, and maintainable.

C. Tracing Links Prospectively

To address the economic challenge of overhead cost, we capture traceability links automatically in the background, *in situ* while artifacts are generated or edited. This prospective technique is complementary to retrospective techniques which recover links after the fact [9, 11, 15]. The online capture of links can ensure that important information is not neglected or overlooked due to lack of resources or time [29]. The idea of tracing prospectively is not new [24], but it was only recently that this approach became feasible [19]. Empirical evidence reveals that traceability links captured in this manner are often useful [27].

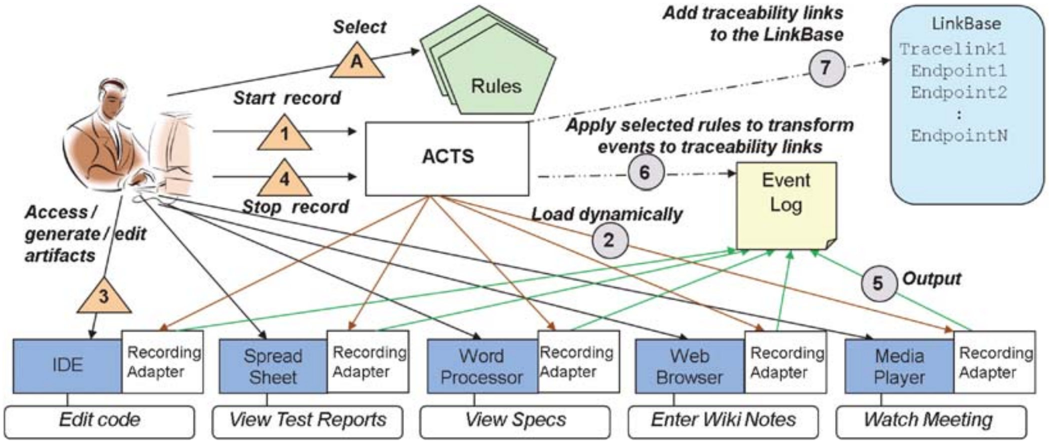


Figure 1. Prospectively capturing traceability links

Figure 1 shows an overview of the process of prospectively capturing traceability links—numbers denote steps, triangled steps denote user actions, and circled steps denote automated tool support. A user may select which rules to apply prior to any recording session (Step A). The user initiates a recording session in the trace tool (1). The trace tool invokes appropriate tool-specific recorders (2, brown arrows) whenever the user opens specific artifacts. As the user performs development tasks and accesses, generates, or edits artifacts (3), each recorder captures the user interaction events. Each event captured is associated with the resource path and perhaps a location within the resource. When the user ends the recording session of the trace tool (4), the adapters output the captured events to a common event log (5, green arrows). The trace tool orders the events sequentially. Rules may be automatically applied to transform the event log into traceability links (6). Finally, the new traceability links are added to the linkbase (7).

IV. Tool Support

We implemented ACTS within ArchStudio, an architecture-centric development environment integrated with Eclipse [10][1]. We focused on relating artifacts to the structural representation of the architecture and we designed our system to achieve the goals of supporting stakeholder-driven traceability and prospectively capturing links across heterogeneous artifacts. Detailed usage scenarios are in [7].

A. Using Open Hypermedia Concepts

A main difficulty with current traceability approaches is tracing across heterogeneous artifacts and tools [14]. Previous efforts to integrate off-the-shelf (OTS) software development tools required the framework to adjust to the assumptions of the OTS tools [25] or required OTS developers to adhere to the environment's framework [1]. In contrast, our approach loosely integrates existing independently developed tools into a traceability framework via open hypermedia techniques such as adapters and first class traceability links (see [7]).

To support user-customized link capture, we designed the tool to have an explicit extension point for incorporating custom adapters. The adapters, which utilize a third party tool's API, are external to the trace tool and are independent of each other. In addition to the concept of rendering adapters that are used in a hypermedia system [3], we created recording and maintenance adapters. Recording adapters capture user interactions within the context of the tools. Rendering adapters render the artifacts within the context of the third-party tool when a link is navigated. Finally, maintenance adapters detect if a traced artifact has been deleted, modified, or moved.

Capturing links across heterogeneous artifacts is supported through tool-specific recorders. The following tools are supported: Eclipse, Mozilla Firefox, MS Office (Word, Excel, Powerpoint), and Adobe Acrobat. Currently, only the artifacts launched within these tools can be prospectively captured; however, adapters can also be implemented for other tools.

To support queries and updates, we use first class n-ary traceability links. We currently use a xADL file [28] which is an XML-based architecture description language to store our linkbase; xADL was extended with a traceability schema. A traceability link consists of a set of two or more endpoints that share a common relationship.

B. Using Rules

We use rules to analyze a potentially large set of user interaction events and to automate the capture of traceability links. Rules are used to filter noise by ignoring user interactions based on a criterion (e.g., "ignore files *ProjectX*", "ignore Save events"). Rules are also used to transform the user interactions to traceability links by using a criterion (e.g., patterns of events or primary trace artifact). To decouple heuristics from the trace tool, rules are external to ACTS. Rules are implemented as XSL Transformations (XSLT) on XML, and Xalan-Java acts as our rule engine. A rule is usually represented by a single XSLT file.

Rules may be applied in the background or interactively. To apply rules in the background, the user selects the rules prior to a recording session. The rules are automatically applied after each recording session. To apply rules interactively, ACTS prompts the user to select a rule after each recording session. The status of link transformation is displayed and additional rules may be applied. In both cases, traceability links are added to the linkbase after the rule application.

[1] For info on the tool, see http://faculty.washington.edu/hazeline/acts/

TABLE I. COMPARISON OF THE ACCEPTABILITY OF OVERHEAD TIME BETWEEN THE TWO GROUPS

Question	Interactive	Background	P-Value
Please rate the acceptability of the time spent on recording links	Ave: 2.4 ± 0.9	Ave: 2.2 ± 0.8	0.341
Please rate the acceptability of time spent on applying rules	Ave: 3.0 ± 1.3	N/A	N/A
Note: Scale of 1 to 5, where 1= "Highly Acceptable" and 5="Unacceptable"			

TABLE II. LEVEL OF DISTRACTION IN LINK CAPTURE WITH ACTS

Question	Interactive	Background	P-Value
Did you find the turning on and off of the recording button distracting?	Ave: 2.8 ± 1.2	Ave: 3.9 ± 1.1	0.027
Did you find the rules to be distracting?	Ave: 2.2 ± 1.0	N/A	N/A
Note: Scale of 1 to 5, where 1= "Very Distracting" and 5="Not at all"			

I. EVALUATION

To assess the extent to which ACTS addresses challenges from the three perspectives, we ran case studies in various contexts: lab settings, a proprietary project, and an open source project. We focused on the following questions:

Q1. Is the cost associated with capturing a traceability link acceptable to users? *(Economic and social)*
Q2. Are architecture-centric traceability links usable? *(Technical and social)*
Q3. Would users use the tool in future projects? *(Social, economic and technical)*

A. Case studies in laboratory setting

Case studies in a laboratory setting allow us to perform a screen capture of the user actions with the tool for later analysis. We solicited feedback from 18 participants, 15 of whom are PhD students in either Computer Science or Informatics at the University of California, Irvine. In addition, seven of these participants have held industry positions (e.g., programmer, architect, manager, and intern), allowing them to assess the feasibility of the tool based on their roles in industry.

The participants were asked to use the ACTS tool in the following task: capture traceability links while editing a structural design in ArchStudio and while viewing or editing documentation files. Half of the users were asked to interactively apply rules and half applied the same rules in the background. All users were also asked to retrieve the captured links. Prior to the task, users were given a brief tutorial on how to use the tool (about 5 minutes). After the task, users were then asked to provide feedback via a questionnaire and semi-structured interview. Based on the screen capture, we also measured the time for capturing and retrieving traceability links.

Q1: Both groups concur that time spent is acceptable (see Table 1). An independent one-tailed t-test shows that there is no significant difference between the ratings of both groups, suggesting that both groups view the overhead as equally acceptable. Users commented that the tool "make[s] the linking job easier" and "[t]he tool saved me lots of time. Thanks!"

Using the recorded screen capture, we also measured the time it takes for users to capture traceability links and to retrieve the captured links. The box plot in Figure 2 shows the distribution of overall times between users who applied the rules interactively and users who applied the rules in the background. The box plot shows that most of the users are more efficient when applying rules in the background. A one-tailed independent t-test comparison between the two groups yields a p-value of 0.005, strongly suggesting that applying rules in the background is more efficient than interactive application.

Meanwhile, the participants in the interactive rules group felt that the manual application of rules was less acceptable (average rating of 3 in Table I). These users felt that turning the record button on and off was "somewhat distracting" while users who applied the rules automatically in the background found it less distracting (see Table II). An independent one-tailed t-test between the two groups on this question shows p = 0.027, indicating there is a significant difference between the views of the groups. A user from the interactive rules group described turning the recording on and off to be "tedious."

Q2: Linking artifacts to the architecture (or design) is a useful feature to some of our participants. Five participants (three with industry experience) identified this feature as the feature they liked about the tool. One of these participants, an architect/programmer, reasoned that "[i]n practice [it is] necessary to link architecture to documents (requirements specs). [T]his makes the documents more useful."

Q3: Over half of the participants indicated interest in continued usage of the tool for their own software development projects. Nine participants answered "yes", and three additional participants answered "yes" with conditions, such as if the user is a system architect using ArchStudio or if the user needed to link the documentation with the design. Three additional participants answered "maybe".

Most of the participants liked the ability to link design elements to documentation, the ability to link to specific points within the documentation, and the ease of navigating to the documentation from design. One participant liked the ability to view the artifacts in their native tool: "Simplifies the process, everything is integrated into one workspace."

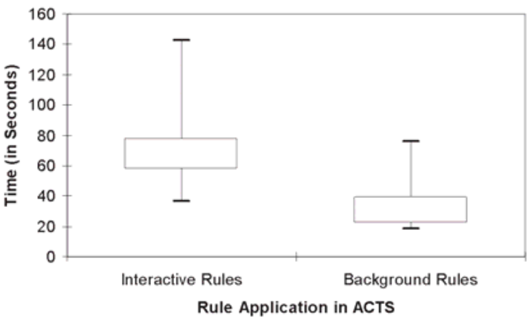


Figure 2: Time to capture and retrieve traceability links

B. Case study in a proprietary project

The tool was also used in a company where capturing and managing traceability is crucial. The tool was used by an engineering specialist who was in charge of evaluating software artifacts and capturing traceability links to them.

Q1: The user was concerned with the amount of overhead needed to create the rules that will filter the traceability links. Since the user may be switching between different projects, the user may be visiting artifacts that should not be traced to the current project. Because the rules are currently expressed as XSLT, it required more overhead than was acceptable to the user. "I need an environment that allows me to make snap decisions about individual potential links, or whole groups of links, quickly and on the fly."

Q2: Centering links on the architecture was of limited value because the organization has a requirements-centric approach to traceability and it currently has no centralized architecture document for a project (i.e., the architecture is spread out over multiple documents and different formats). Moreover, traceability capture starts before an architecture is created (e.g., among different levels of requirements document).

Q3: The user has expressed reservations with using ACTS, because of its architecture-centric approach and because of usability issues, such as the background capture of links. Again, because the user may be switching between different projects and because the user wanted to ensure that traceability links are indeed captured, the user wanted a capability to simply "point" to the artifact to capture the traceability links, instead of automatically capturing links in the background.

C. Case study in an open source project

Finally, to assess how the ACTS tool works in the context of a group of users, the tool was used to support a team of developers adding functionality to the ArchStudio project. A team of four undergraduate students with industry experience (architect, programmer, technical support) were instructed to capture links between the architecture and online references during the ten weeks of development. They were also provided a pre-defined set of rules to use.

Q1: Three of the four students indicated that the time spent in recording and using the pre-defined rules was acceptable. Using the same scale as in Table I, two students rated it as 1 and one student rated it as 2.

Q2: The students indicated that linking to the architecture was useful. Three students found it useful in the following tasks: learning about the system, gathering requirements, designing, implementing new features, and testing. Since they coordinated their tasks among themselves, three students used the traceability links to help answer questions from their teammates. When asked how often they used their traceability links to perform these tasks, 3 students responded "Majority of the time" while one responded "One fourth of the time".

Q3: All the students stated that they would continue using the tool. They pointed out that "[t]he main hassle is the installation and setup," but once they learned how to use the tool, they found that it supported their development tasks. The students commented, "Thanks. It's very useful for me" and "Excellent tool."

D. Discussion

The case studies demonstrated that in development settings where an architecture is predominantly used, the ACTS technique was able to address difficulties from the three perspectives, such as the cost in capturing links (with pre-specified rules), supporting traceability capture across different tools, and increasing the motivation to perform traceability tasks (indicated by the willingness to use the tool again).

Q1: There is a concern with regards to the overhead involved in creating the rules, as expressed by the engineering specialist in the proprietary project case study. Because XSLT requires an initial learning curve, we previously planned that a "traceability administrator" would be in charge of creating the rules and simply providing them to the users, as we have done in the lab usage trials and in the open source case study. It turns out that there are users who would like to create their own rules, and thus, making the rules accessible to the general user is an important requirement to be addressed in future work.

During this study, we also encountered an unexpected result—that users are willing to spend more time capturing traceability links as long as they are in control of the traceability task. In fact, some lab participants asked for the ability to manually map artifacts to design via "drag and drop." One participant even suggested displaying a dialog box to manually confirm the links to add after each recording session: "I'd like to have more control over the links. I'd like to have checkboxes to manually pick the one link I wanted." One participant also wanted to be able to manually enter labels for the captured traceability links. This finding is consistent with our engineering specialist who works in a setting where it is important not to have missing links. Thus, higher cost may be acceptable to users as long as they have more control.

Q2: Our approach takes the assumption that there is an architectural model whereby all the other artifacts can be linked. The lab usage trial as well as open source case study, where an architectural model exists, has demonstrated that capturing traceability links work well. In fact, the traced artifacts can support development tasks as demonstrated in the open source project case study. However, in settings where there is no dominant architectural model, as in the proprietary project case study, our approach provides limited value. A placeholder or "bare-bones" architectural model could be used until a substantive model is developed.

Q3: Future tool usage was largely dependent on the user experience during the trial usage and the context of usage. Many of the lab users and the undergraduate team saw that they could benefit from ACTS. Two of the lab users, an architect and a manager, opined that ACTS could be useful in the context of their work: documents become more useful when linked to the architecture and traceability is useful during company audits. The undergraduate team was also able to save time since they could organize their references around the architectural model. We also see that if ACTS does not match the context of usage, as in the proprietary project, it is unlikely to be used in that setting.

While the lab participants like the idea of being able to automatically capture links to documentation, they expressed several usability issues regarding the current tool implementation. Some participants prefer more visual feedback on the status of traceability capture (e.g., a better visual indicator that the recording is taking place). One user would like more "on-the-fly" directions to know what actions to take while recording. These usability issues can be addressed by further tool development.

Most of the literature on automating traceability techniques focus on increasing the accuracy of traceability links captured to minimize the cost (i.e., the time spent on capturing traceability links [15]). By holistically analyzing the problem, we learn that software engineers' aversion to traceability is not solely because traceability tasks are time consuming, but because of the interaction of cost with social challenges (e.g., engineers capture traceability links because of an external mandate [4, 6]) or the interaction of cost with the technical challenges (e.g., captured traceability links quickly become outdated). Time spent in capturing traceability links becomes less of an issue when software engineers have a direct benefit [4, 6]. This is confirmed by our finding where some users are also willing to spend more time for a greater level of control.

II. Conclusion

This paper examined the traceability challenges and showed that the complexity of the problem stems from multiple interacting factors that arise from the economic, technical, and social perspectives. ACTS is a novel technique that holistically tackles these challenges by unifying distributed and varied artifacts around the architecture, by catering to stakeholder interests, and by prospectively capturing links. The ACTS tool support is extensible—existing OTS tools can be integrated via hypermedia adapters and heuristics can be specified via external rules. Our case studies have shown that ACTS can address interacting challenges from different perspectives.

There are several avenues of future work. ACTS can potentially be adapted to settings where an architectural model may be distributed among different artifacts or may not play a central role in development. More work is also needed to understand other interactions between the three perspectives. For example, it is important to know how to balance individual versus organizational priorities in capturing traceability links. We anticipate that these issues can be addressed and that a holistic approach like ACTS can aid in transitioning state of the art traceability techniques into practice.

Acknowledgment

The authors are grateful to G. Mark, S. Hendrickson, Y. Zheng, and E. Dashofy for useful discussions, and C. Leu, J. Meevasin, H. Pham, D. Purpura, and A. Rahnemoon for tool development.

References

[1] The Jazz Project. http://jazz.net.

[2] K.M. Anderson, S.A. Sherba, and W.V. Lepthien. Towards large-scale information integration. In *Proc of ICSE*, 2002.

[3] K.M. Anderson, R.N. Taylor, and E.J. Jr. Whitehead. Chimera: Hypermedia for heterogeneous software development environments. *TOIS*, 18(3):211–245, 2000.

[4] P. Arkley and S. Riddle. Overcoming the traceability benefit problem. In *Proc of 13th Int'l Conf on Requirements Engineering (RE)*, 2005.

[5] H.U. Asuncion, A.U. Asuncion, and R.N. Taylor. Software traceability with topic modeling. In *Proc of ICSE*, 2010.

[6] H. U. Asuncion, F. François, and R.N. Taylor. An end-to-end industrial software traceability tool. In *Proc of ESEC/FSE*, 2007.

[7] H.U. Asuncion and R.N. Taylor. *Software and Systems Traceability*, chapter Automated Techniques for Capturing Custom Traceability Links across Heterogeneous Artifacts. Springer-Verlag, 2012.

[8] J.A. Camacho-Guerrero, A.A. Carvalho, M.G.C. Pimentel, E.V. Munson, and A.A. Macedo. Clustering as an approach to support the automatic definition of semantic hyperlinks. In *Proc of Conf on Hypertext and Hypermedia*, 2007.

[9] J. Cleland-Huang, R. Settimi, E. Romanova, B. Berenbach, and S. Clark. Best practices for automated traceability. *Computer*, 40(6):27–35, 2007.

[10] E. M. Dashofy, H. Asuncion, S.A. Hendrickson, G. Suryanarayana, J.C. Georgas, and R.N. Taylor. Archstudio 4: An architecture-based meta-modeling environment. In *Proc of ICSE*, volume Res Demo, 2007.

[11] A. De Lucia, M. Di Penta, and R. Oliveto. Improving source code lexicon via traceability and information retrieval. *TSE*, 37(2):205 –227, 2011.

[12] A. Egyed, S. Biffl, M. Heindl, and P. Grünbacher. Determining the cost-quality trade-off for automated software traceability. In *Proc of Int'l Conf on Automated Software Engineering (ASE)*, 2005.

[13] O. Gotel and A. Finkelstein. Modelling the contribution structure underlying requirements. In *Proc of 1st Int'l Workshop on RE: Foundations for Software Quality*, 1994.

[14] J. Hayes and A. Dekhtyar. Grand challenges for traceability. Tech Rep COET-GCT-06-01-0.9, Center of Excellence for Traceability, 2007.

[15] J. H. Hayes, A. Dekhtyar, and S. K. Sundaram. Advancing candidate link generation for requirements tracing: The study of methods. *TSE*, 32(1):4–19, 2006.

[16] S. A. Hendrickson and A. van der Hoek. Modeling product line architectures through change sets and relationships. In *Proc ICSE*, 2007.

[17] P. Inverardi, H. Muccini, and P. Pelliccione. Automated check of architectural models consistency using SPIN. In *Proc ASE*, 2001.

[18] M. Jarke. Requirements tracing. *Com. of the ACM*, 41(12), 1998.

[19] F. Jouault. Loosely coupled traceability for ATL. In *Proc of the ECMDA Workshop on Traceability*, 2005.

[20] J. Leuser. Challenges for semi-automatic trace recovery in the automotive domain. In *Proc of Workshop on TEFSE*, 2009.

[21] M. Lindvall and K. Sandahl. Practical implications of traceability. *Software - Practice and Experience*, 26(10):1161–80, 1996.

[22] B. Nuseibeh. Weaving together requirements and architectures. *Computer*, 34(3):115–117, March 2001.

[23] P. Oreizy, N. Medvidovic, and R.N. Taylor. Architecture-based runtime software evolution. In *Proc of ICSE*, 1998.

[24] F.A.C. Pinheiro and J.A. Goguen. An object-oriented tool for tracing requirements. *Software*, 13(2):52–64, 1996.

[25] K. Pohl, K. Weidenhaupt, R. Dömges, P. Haumer, M. Jarke, and R. Klamma. PRIME–toward process-integrated modeling environments. *TOSEM*, 8:343–410, October 1999.

[26] B. Ramesh and M. Jarke. Toward reference models for requirements traceability. *TSE*, 27(1):58–93, 2001.

[27] J. Singer, R. Elves, and M.-A. Storey. Navtracks: Supporting navigation in software maintenance. In *Proc of ICSM*, 2005.

[28] R. N. Taylor, N. Medvidovic, and E.M. Dashofy. *Software Architecture: Foundations, Theory, and Practice*. John Wiley & Sons, 2010.

[29] A. von Knethen and B. Paech. A survey on tracing approaches in practice and research. Tech Report IESE-Report Nr. 095.01/E, Fraunhofer Institut Experimentelles Software Engineering, Fraunhofer Gesellschaft, 2002.

Pointcut Design with AODL

Saqib iqbal
Department of Informatics
University of Huddersfield,
Huddersfield, HD1 3DH, United Kingdom
s.iqbal@hud.ac.uk

Gary Allen
Department of Informatics
University of Huddersfield,
Huddersfield, HD1 3DH, United Kingdom
g.allen@hud.ac.uk

Abstract—**The designing of pointcuts is a crucial step in Aspect-Oriented software development. Pointcuts decide the places where aspects interact with the base system. Without designing these pointcuts properly, the weaving process of aspects with the base system cannot be modelled efficiently. A good design of pointcuts can ensure proper identification of join points, clear representation of advice-pointcut relationships and overall efficiency of the weaving process of the system. The existing approaches do not design pointcuts separately from their parent aspects, which hinders in identifying pointcut conflicts before the implementation of the system. This paper provides a set of graphical notations to represent join points, pointcuts, advices and aspects. A graphical diagram has been proposed that shows the relationships between pointcuts and their relevant advices. The paper also provides a technique to represent and document pointcuts along with their related advices and corresponding base elements in a tabular way. The technique can help in resolving two of the most complicated problems of pointcut modelling, the fragile pointcut problem and the shared join point problem.**

Keywords-component; Aspect-Oriented Design, Pointcut Design, Pointcut Modelling, Aspect-Oriented Design Language

I. INTRODUCTION

The handling of concerns is extremely important for critical, distributed and complex systems. Each concern has to be identified, specified and designed properly to avoid inconsistencies which can lead to serious consequences if the system is cr itically sensitive. Object-oriented design approaches have been the pick of the design techniques for such systems during the last three decades. Unified Modelling Language [2] emerged in the 1990s, and rapidly became accepted as the standard analysis and design approach for object-oriented development of systems. Unfortunately, object-oriented approaches started showing problems in capturing concerns that are scattered in nature and whose implementation overlaps other concerns and/or system units. Such concerns are known as crosscutting concerns. Examples of crosscutting concerns include system security, logging, tracing and persistence. The implementation of these concerns resides within the implementation of other concerns or classes, which results in inconsistencies and modification anomalies.

Aspect-Oriented Programming (AOP) [1] was introduced to rectify this problem. AOP introduced a new construct, called an aspect, besides the traditional object-oriented classes. The aspect is i dentified, specified and designed separately, and is woven into the base system at run-time wherever it is required. AOP was proposed as an implementation solution to the crosscutting problem. That is the reason why initial developments in AOP were mainly in the field of implementation. AspectJ, AspectWerkz and JBoss AOP are among a n umber of implementation technologies which were proposed immediately after AOP came into existence. With the passage of time research was extended to earlier phases of development as well, resulting into development of aspect-oriented requirement engineering and design approaches. Although modularity of aspect-oriented systems has been ensured with the introduction of such techniques, the cohesive nature of aspects with the base system has not been addressed properly in the existing approaches. Pointcuts, which are responsible for identifying join points in the system where aspects are invoked, are highly dependent on the base program's structural and behavioral properties. This characteristic makes pointcuts fragile with respect to any changes in the base program, and results in a problem called *Fragile Pointcut Problem* [6,7]. Another problem related to pointcuts is *Shared Join Point Problem* [4], where more than one advice from a single or multiple aspects are supposed to be executed at a single join point. A proper design is required to resolve the precedence of advices so that they run in a pre-determined order. This paper addresses these two pointcut problems and proposes a diagrammatic and tabular approach to help in rectifying both issues without compromising the consistency of the system. The tabular approach promises better documentation of pointcuts and enables modifications to be made in a consistent manner.

The rest of this paper is structured as follows: Sections 2 and 3 desc ribe the Fragile Pointcut Problem and the Shared Join Point Problem respectively. Section 4 describes the proposed pointcut-advice diagram and Pointcut Table, and section 5 provides discussion and conclusion of the paper.

II. THE FRAGILE POINTCUT PROBLEM

Pointcuts are considered fragile because their definitions are cohesively coupled with the constructs in the base system. Upon modification in the base system, pointcuts' semantics are bound to change [6,7]. Pointcuts are defined by join points which are specific points in the base system. Once a join point is modified in the base system, the relevant pointcut is altered to adopt that change or lose that particular join point. This fragile nature of pointcuts forces designers to reflect changes in the pointcut definitions when they make any modification to the base system. Join points are formed not only on structural characteristics of the system but also on behavioural properties of functional units of the system. Therefore, any type of change to structural or behaviour property of the base system would require related pointcuts to be altered [3]. The fragility of pointcuts leads to two core problems: unintended join point capture problem, which arises when a join point is accessed which does not exist anymore because of modifications to the source code; and accidental join point miss problem, which happens when a join point is not captured which was originally supposed to be captured, again due to an alteration to the source code of the base program [3].

III. THE SHARED JOIN POINT PROBLEM

Aspects superimpose their behaviours at well-defined join points in the base system. A shared join point is a point where multiple aspects interact and superimpose their advices. The problem arises in deciding which aspect should run first. This is a critical decision because execution of one aspect's advice can change the attributes which are supposed to be used by another aspect's advice. Figure 1, taken from [4], illustrates the problem.

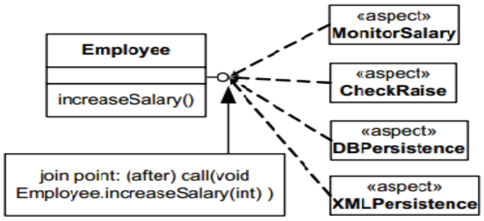


Figure 1. 'Employee' class and its superimposed aspects (taken from [4]).

This problem is considered to be an implementation-level problem and has been addressed by renowned aspect-oriented programming techniques. For example, AspectJ [10] provides a *declare precedence* keyword to order advices, Composition Filters [8] provides *Seq* operator to declare precedence, and JAC [9] determines the order by implementing wrappers in the classes which are filed in a wrapper file in an execution sequence.

IV. DESIGNING POINTCUTS IN AODL

Pointcuts rely heavily on the lexical structure of the base program. The definition of pointcuts (group of join points) contains direct reference to the syntax of base elements. This tightly coupled nature makes it hard for programmers to make any changes to base program without having knowledge of pointcuts and vice versa. Aspect-Oriented Design Language (AODL) [5] presents a diagrammatic approach to the design of pointcuts and a tabular way of documenting their definitions. This kind of well-documented representation of aspects in the design phase makes it easier for designers as well as programmers to evolve either aspects or the base program. Before moving onto the proposed models, we introduce AODL briefly in the following section.

A. Aspect-oriented Design Language

Aspect-Oriented Design Language (AODL) [5] is a UML-like design language that has been developed by the authors to design aspects, aspectual elements, and object-aspect relationships. AODL offers a unified framework to design both aspects and objects in one environment. The constructs of aspects are denoted by specialized design notations and structural and behavioral representations are done with the help of design diagrams. The details about the semantics of the notations can be found in [5]. Figure 2 shows design notations adopted by AODL.

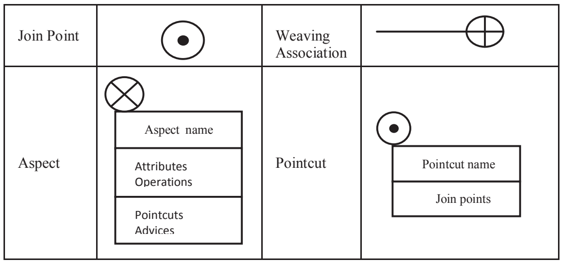


Figure 2. AODL Design Notations

AODL proposes a three phase design for aspects, constituent elements and the relationships between aspects and objects, details can be found in [5]. In the first phase, join points are modeled with the help of two diagrams, one for the structural identification of join points, known as a Join point Identification diagram, and the second for the behavioral representation of join points, known as a Join Point Behavioural diagram. In the second phase, aspects are designed with the help of an Aspect Design Diagram. And in the final phase, aspects are composed with the base model with the help of two diagrams, an Aspect-Class Structure Diagram and an Aspect-Class Dynamic Diagram. Complete details about the semantics of the language and its usage can be found in [5].

B. Pointcut-Advice Diagram

Pointcuts are highly dependent on the base objects through join points. Similarly, advices are tightly coupled with their corresponding pointcuts. To represent these cohesive relationships, we need to show related pointcuts, advices and base objects in a diagrammatic model. An Aspect Design Diagram (shown in Figure 3) contains the properties and behavior of an aspect. The diagram connects with the base classes through use of the crosscuts stereotype. Pointcuts are represented along with their corresponding advices and occurrence attributes (before, after and around) in a pointcut-advice diagram.

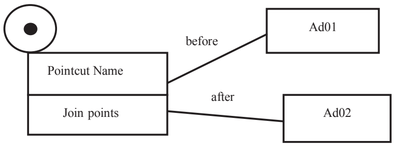


Figure 3. Pointcut-Advice Diagram

C. Pointcut Table

Pointcuts are predicates which are set on defined points (join points) in the execution of a program. To define and document pointcuts properly ensures consistency of the program. AODL has proposed a pointcut table (shown in Table 1) to document pointcuts along with their related advices, aspects and base classes. The table defines pointcuts in vertical columns by indicating the join points of the base system horizontally. The columns of the table provide list of aspects and complete definition of their pointcuts along with their related advices. The rows, on the other hand, show the base system attributes, methods and execution points where join points have been identified. The execution order of advices on a single join point is declared in the last column, named Order. The table has been tested and verified to represent all types of legitimate AspectJ pointcuts, as defined in [11].

Table 1: Pointcut Table

	Aspect A		**Aspect B**		**Order**
	AdA1 (Before)	**AdA2 (After)**	**AdB1 (Before)**	**AdB2 (Around)**	
Class A	this				
Attribute					
constructor	execution				
Method1	execution		execution		AdA1, AdB1
Method2					
getX()					
Class B					
Method1	call		within		AdB1, AdA1
Method2		exception(type)			
Pointcut Definition	**this**(A) && **exec**(MA1) && **call** MB1)	**exception**(type)	**call**(A1) OR **within**(MB1)		
Pointcut	**P1**	**P2**	**P3**	**P4**	
Pointcut Trigger	**!(P2)**		**cflow(P1)**	P1 && P2	
Complete Definition	**this**(A) && **exec**(MA1) && **call** (MB1) &&**!**(P2)	**exception**(type)	**call**(A1) OR **within**(MB1) && cflow(P1)	P1 && P2	

In case of system being too complex and containing a number of aspects, the table can be broken into multiple tables and a specific number of aspects can be contained in each table. This way each table will contain pointcut information about a specific number of aspects only (say 3 or 5) and the readability of the system will improve.

D. *Implications of the Approach*

The existing aspect-oriented design approaches do not provide means to design pointcuts separately from their parent aspects. The composition of aspects heavily depends on consistency of pointcuts because they define the join points where aspects are woven into the system. The authors felt that (i) the pointcuts should be designed properly with the help of designated design notations and design diagrams, (ii) the pointcuts should be documented properly so that their features and relationships are specified efficiently before being implemented, and (iii) the pointcuts should be ordered at the modelling level so that problems such as the fragile pointcut problem and the shared join point problems are handled before implementation.

The authors of the paper do not claim that the proposed diagrammatic and tabular approach resolve both the problems completely. It is, however, suggested that adopting this type of approach can help in resolving a number of problems especially inconsistencies and conflicts involving pointcuts.

E. *Example*

To make the proposed approach more understandable, we will implement the *Observer Pattern* [12] as implemented by [13]. An abstract aspect Observer is extended by two aspects, *Observing1* and *Observing2* as shown in Figure 4. The following sections implement this example using a Pointcut-Advice Diagram and a Pointcut Table.

```
OBSERVER PATTERN
abstract aspect Observer {
void notify() { ... }
abstract pointcut p();
abstract pointcut c();
after(): p() {notify();}
after(): c() {notify();}
}
aspect Observing1 extends Observer {
pointcut p(): call(void Buffer.put(int));
pointcut c(): call(void Buffer.get());
}
aspect Observing2 extends Observer {
pointcut p():within(Buffer) && call(* put*(*);
pointcut c():within(Buffer) && call(* get*(*));
}
```

Figure 4: Observer Pattern Implementation (Taken from [13])

Pointcuts are designed with the help of pointcut-advice diagrams, where each pointcut is represented along with its corresponding advice. As shown in Figure 5a, pointcuts p() and c() of the *Observing1* aspect have been represented with the help of a pointcut diagram which shows the definition of the pointcut (set of join points) and the advices with the occurrence attribute (which is after for both the pointcuts). Similar diagram has been drawn in the Figure 5b for *Observing2* aspect.

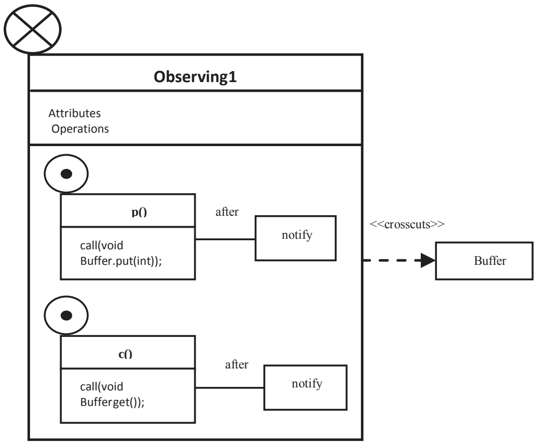


Figure 5a: Aspect Design Diagram for *Observing1*

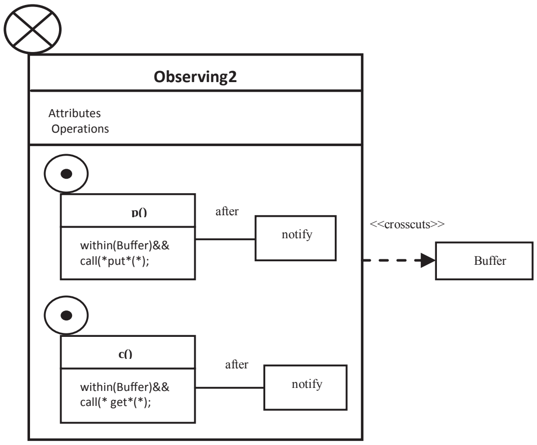


Figure 5b: Aspect Design Diagram for *Observing2*

Table 2 shows the pointcut table for the Observer Pattern. It documents all of the information about the pointcuts of a particular aspect, along with the corresponding base methods and attributes, in a tabular way. Besides documenting all the information about a pointcut, the table also shows the order in which advices are executed on a particular join point. For instance, in the first row of the table we have two advices, AdOb2_1 and AdOb2_2, which are supposed to be executed on a join point "within" which is defined on objects of the Buffer class. The table also provides the Order column, where we can show which advice should be executed first, which in the case of Table 2 shows that AdOb2_1 will execute before AdOb2_2.

Table 2: Pointcut Table for Observer Pattern

	Observing1		Observing2		Order
	AdOb1_1 (After)	AdOb1_2 (After)	AdOb2_1 (After)	AdOb2_2 (After)	
Class Buffer			within	within	AdOb2_1, AdOb2_2
Attributes					
put()			call		
put(int)	call		call		AdOb1_1, AdOb2_1
get()		call		call	AdOb1_2, AdOb2_2
get(int)				call	
Pointcut Definition	call(void Buffer.put(int))	call(void Buffer.get())	witin(Buffer) && call(*put*(*))	witin(Buffer) && call(*get*(*))	
Pointcut Name	P()	c()	p()	c()	
Pointcut Trigger					
Complete Definition	call(void Buffer.put(int))	call(void Buffer.get())	witin(Buffer) && call(*put*(*))	witin(Buffer) && call(*get*(*))	

V. FRAGILE POINTCUT AND SHARED JOIN POINT PROBLEM

The method to document pointcuts, shown in Table 1 and Table 2, reduces the fragile nature of pointcuts. The table provides complete information about a pointcut, which helps in modifying the pointcut without allowing inconsistencies. This kind of tabular documentation helps remove modification anomalies when join point definitions are altered in the base system.

The pointcut table also provides a column named Order to declare the precedence of advices which have execution clashes with each other. Advices are grouped in order of their execution on a join point which is shown in that particular row. The table therefore provides an opportunity for designers to set the precedence on the execution of advices during the design phase in order to avoid clashes in execution.

It is again stressed that the authors do not claim that the proposed approach resolves all types of pointcut conflicts including the fragile pointcut problem and shared join point problem. It is, however, asserted that this approach can help in designing pointcuts at modelling level and it can help in identifying and resolving some key issues of pointcuts before they are implemented.

VI. DISCUSSION AND CONCLUSION

Aspect-Oriented Development unifies separately defined aspects with objects of the base system through well-defined join points. The composition between aspects and objects depends heavily on the identification of join points and their proper grouping in the form of pointcuts. The cohesive nature of pointcuts is defined by definition of join points which are susceptible to change if an alteration is made in the base program. The resultant pointcuts are inconsistent with the system and result in either missing some join points or capturing join points which are not intended by the designer. To avoid such problems, proper documentation and proper design of a pointcut becomes vitally important. This paper has proposed a diagrammatic approach to the design of pointcuts along with their related aspects, advices and base objects. The diagram does not only help structural design of a pointcut but also helps in representing the relationships between an aspect and its corresponding advices. The paper also proposes a tabular approach to the documentation of pointcuts during the design phase, so that all of the corresponding definitions of a pointcut are defined along with its characteristics and parent entities (aspects and classes). The pointcut table promises the rectification of two of the most common problems of pointcut design, the fragile pointcut problem and the shared join point problem during the design phase.

The future research will strive to develop pointcut composition models to identify aspect interferences at the pointcut level. Tool support will also be provided to automate development of pointcut models and generation of code.

REFERENCES

[1] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J. M. Loingtier, and J. Irwin. "Aspect-Oriented Programming," In: Proceedings of ECOOP 1997, Jyväskylä, Finland, June 9-13, 1997, pp. 220-242.

[2] "Unified Modelling Language", OMG, http://www.uml.org/. Accessed on 09 Dec. 2011.

[3] A. Kellens, K. Gybels, J. Brichau, and K. Mens. A model-driven pointcut language for more robust pointcuts. In Proc. Workshop on Software Engineering Properties of Languages for Aspect Technology (SPLAT), 2006.

[4] I. Nagy, L. Bergmans, and M. Aksit. "Composing aspects at shared join points". In Proceedings of International Conference NetObjectDays (NODe), Erfurt, Germany, Sep 2005.

[5] S. Iqbal and G. Allen, *Designing Aspects with AODL.* International Journal of Software Engineering. 2011, ISSN 1687-6954 (In Press)

[6] C. Koppen and M. Stoerzer. Pcdiff: Attacking the fragile pointcut problem. In First European Interactive Workshop on Aspects in Software (EIWAS), 2004.

[7] M. Stoerzer and J. Graf. Using pointcut delta analysis to support evolution of aspect-oriented software. In 21st IEEE International Conference on Software Maintenance (ICSM), pages 653–656, 2005.

[8] "Compose* portal", http://composestar.sf.net, Accessed on 24 Jan, 2012.

[9] "Java Aspect Component", http://jac.ow2.org/, Accessed on 24 Jan, 2012.

[10] Eclipse, AspectJ, http://www.eclipse.org/aspectj/, Accessed 05 Dec. 2010

[11] Pointcuts, Appendix B. Language Semantics, http://www.eclipse.org/aspectj/doc/next/progguide/semantics-pointcuts.html, Accessed on 30 Jan, 2012.

[12] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Professional Computing Series. Addison-Wesley, Reading, Ma, USA, 1995.

[13] W. Cazzola, S. Pini, and M. Ancona. Design-Based Pointcuts Robustness Against Software Evolution. In Proceedings of RAM-SE'06 Workshop, Nantes, France, July 2005

Feature modeling and Verification based on Description Logics

Guohua Shen, Zhiqiu Huang, Changbao Tian, Qiang Ge
College of Computer Science and Technology
Nanjing University of Aeronautics and Astronautics
Nanjing, China
{ghshen, zqhuang}@nuaa.edu.cn

Wei Zhang
School of Electronics Engineering and Computer Science
Peking University
Beijing, China
zhangw@sei.pku.edu.cn

Abstract Most of the current domain engineering methods have adopted the feature model as a domain requirements capturing model. But these methods lack the semantic description of the feature model and the relationship between features. This has led to the redundancy and confusion in feature model representation between different domain engineering methods. In this paper, a description logics-based feature model, including the feature class and the constraints of the features, is presented. A group of rules for constraints are proposed, which are used to verify the consistency and completeness of feature model instances. Then, combining with a real software domain, the modeling process of the feature model with description logic and its verification are discussed systematically.

Keywords *feature model; requirements engineering; software reuse; model verification; description logics*

I. INTRODUCTION

Software Product Line (SPL) is an important way to reuse software domain assets. SPL practices guide organizations towards the development of products from existing assets rather than the development of separated products one by one from scratch. Thus, features can be shared among all software products in domain. The feature model has been widely adopted as a domain requirements capturing model by most of the current domain methods, such as FODA [1], FORM [2], FeatuRSEB [3], PuLSE [4] and SPL [5].

Description logics (DLs) are a family of languages for representing knowledge and reasoning about it. There are several studies proposing the usage of DL to analyze feature models, such as [6,7,8]. Intuitively, a domain-specific feature model is a concrete instance of meta-model. However, these methods do not rigorously differentiate between the feature meta-model and feature models (i.e., instances of meta-model). New concepts, roles and constraints are created for every domain feature model, which causes additional efforts and bloating in the numbers of concepts, roles and constraints. Actually, the feature meta-model is domain-independent and built for only once, while feature models are domain-dependent and must be instantiated for each domain by domain engineers.

II. FEATURE MODEL

A. *Feature*

The features define both common aspects of the domain as well as differences among all products of a SPL. A Feature is a distinctive characteristic of a product, and it may refer to a requirement, a component or even to pieces of code [9].

A feature model is a compact representation of all potential products of a SPL. Feature models are used to model SPL in terms of features and relations among them.

B. *Feature-Oriented model*

The feature model is an abstraction of commonality and variability in a software family across a domain. Most proposals organize feature in hierarchy, and the parent feature is composed of children features. Figure 1 describes the feature model of the graph editor domain.

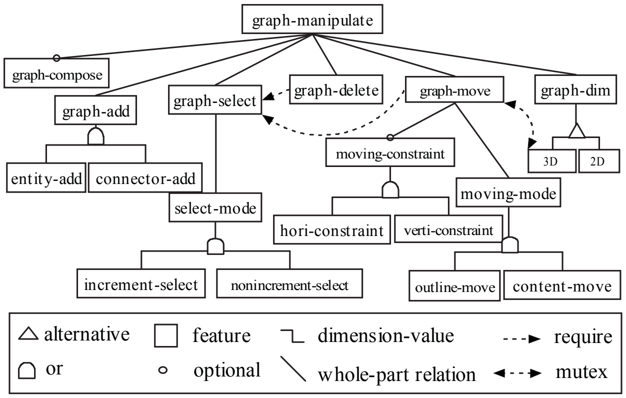


Figure 1. Feature model of graph editor.

The feature model describes the variability in two ways: i) the feature is optional; ii) the feature is a variation point (vp-feature, also called dimension feature [10]), which attaches some value features. For example, the feature *moving-constraint* is optional for its parent feature *graph-move*, and the feature *moving-mode* is a dimension feature with two different values: *content-moving* and *outline-moving* (see Figure 1).

C. *Feature Model Tailoring*

Feature modeling is an activity of development for reuse. We customize a specific software product specification through tailoring.

The system can be changed at certain time in the system's lifecycle. Selecting some variant of the feature model is called binding the variant. Every variation point has at least one associated binding time. There are three types of binding time: *reuse-time*, *compile-time* and *run-time*.

Every feature has the binding states like *bound*, *removed* and *undecided*. A *bound* feature means that if its pre-conditions are satisfied, this feature will be executed. A *removed* feature means that it will never be bound again. An *undecided* feature means that it is currently not in the *bound* state, but still has the chance to be bound or removed in later binding-times. In addition, we use "tailoring" to denote the action of changing a feature's binding state from *undecided* to *removed*, use "binding" to denote the action of changing a feature's binding state from *undecided* to *bound* [11].

III. SEMANTIC FEATURE MODELING

A. *Description Logics*

Description logics play an important role in Semantic Web since they are the basis of the OWL[1] (Web ontology language). DLs offer considerable expressive power, while reasoning is still decidable [12].

Concepts and roles are basic elements in DLs. The syntax and semantics of a description logic are summarized in table 1, where *C* and *R* are concepts and roles respectively.

TABLE I. SYNTAX AND SEMANTICS FOR DLS CONSTRUCTORS

Constructor	Syntax	Semantics
top concept	$\top$	Δ^I
bottom concept	$\bot$	$\varnothing$
concept negation	$\neg C$	$\Delta^I \backslash C^I$
intersection	$C \sqcap D$	$C^I \cap D^I$
union	$C \sqcup D$	$C^I \cup D^I$
existential restriction	$\exists R.C$	$\{a \in \Delta^I \mid \exists b.(a,b) \in R^I \wedge b \in C^I\}$
value restriction	$\forall R.C$	$\{a \in \Delta^I \mid \forall b.(a,b) \in R^I \rightarrow b \in C^I\}$
set	$\{a_1,\ldots,a_n\}$	

A knowledge base *K* of DLs is constituted by the Tbox *T* and the Abox *A*, denoted as *K*=(*T*, *A*), where the TBox introduces the terminology (that is, concepts), as well as axioms like general concept inclusion (GCI) and general role inclusion (GRI); while the ABox contains assertions about named individuals in terms of this vocabulary [12]. In TBox, there are concepts definition and axioms. Inclusion axioms are of the form "$C \sqsubseteq D$". GCI and GRI axioms express the inclusion relation between concepts and roles respectively. For example, axiom "*Woman* $\sqsubseteq$ *Person*" indicates that *Woman* is a sub-concept of *Person*.

In ABox, each assertion states a fact. Assertions are divided into two types: concept assertions (written as "*C*(*x*)") and role assertions (written as "*r*(*x*,*y*)"). For example, the assertion "*Woman*(*mary*)" states that *mary* is an individual (instance) that belongs to the concept *Woman*, while "*hasMother*(*peter*, *mary*)" states that *peter* has mother *mary*.

B. *Semantic feature modeling based on DLs*

Our DLs-based feature model is composed of two levels: the feature meta-model and models. The feature meta-model is domain-independent, and defined in TBox. Feature models are domain-dependent and built in ABox. The domain engineers instantiate models according to the meta-model.

1) *Feature class*

In DLs, the feature in meta-model is expressed as a concept, and its definition is as follows. The concept *Feature* has role *hasBindTime*, whose range is the concept *BindTime*, and has role *hasState*, whose range is the concept *BindState*.

Feature::= $\top \sqcap$hasBindTime.BindTime $\sqcap$ hasState.BindState

The binding time consists of *reuse-time*, *compile-time* and *run-time*. So the concept *BindTime* is defined by using the set constructor as:

BindTime ::= {reuse-time, compile-time, run-time}

The binding state consists of *bound*, *removed*, *undecided* and *conflict*. A new state conflict is appended, which is used for consistency verification (see conflict-rule in detail). So the concept *BindState* is defined by using the set constructor as:

BindState ::= {bound, removed, undecided, conflict}

A dimension feature has variable behavior. The dimension feature concept *DimFeature* is the sub-class of concept *Feature*, and it is associated with some value features *DimValue*.

DimFeature::=Feature $\sqcap \exists$ hasValue.DimValue

The concept *DimValue* is the sub-class of concept *Feature* too. The sub-class relation is defined in GCI axiom as:

DimValue $\sqsubseteq$ Feature.

Figure 2 depicts the concepts of feature meta-model.

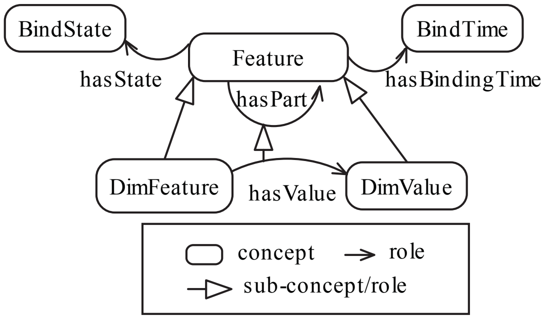


Figure 2. Semantic feature meta-model (concepts)

2) *Relations between features*

The features are organized in hierarchy. For example, the feature *graph-manipulate* is composed of *graph-add*, *graph-select* and *graph-move*. The whole-part relation is described as role *hasPart*, whose domain and range are both *Feature*.

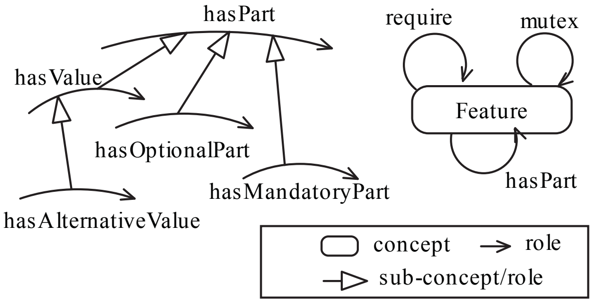


Figure 3. Semantic feature meta-model (roles and constraints).

There are two kinds of whole-part relations: optional and mandatory. These two relations are defined as role *hasOptionalPart* and *hasMandatoryPart*. Both of them are sub-role of *hasPart*, written as following GRI axioms:

[1] http://www.w3.org/TR/owl-guide/

$\text{hasOptionalPart} \sqsubseteq \text{hasPart}$

$\text{hasMandatoryPart} \sqsubseteq \text{hasPart}$

The role *hasValue*, which is sub-role of role *hasPart*, describes the relation between concept *DimFeature* and *DimValue*. The sub-role relation is defined in GRI axiom as:

$\text{hasValue} \sqsubseteq \text{hasPart}.$

Figure 3 depict the roles, and their inclusion relation.

C. Constraints of the feature model

Several methods proposed their own constraints of the model. E.g., the semantic between vp-feature and variant features in FeatuRSEB [3] is somewhat similar to the one between dimension feature and value features in FODM [10]. However, the "require", "mutual exclusion" in FODA [1], and "OR", "XOR" constraints in FeatuRSEB are redundant and easy to cause confusion among different methods. In this paper, we define roles *require* and *mutex*, to describe the dependency and mutual exclusion constraints respectively (see Figure 3).

We define a role *hasAlternativeValue*, which is sub-role of role *hasValue*, to indicate the mutual exclusion constraints.

$\text{hasAlternativeValue} \sqsubseteq \text{hasValue}.$

We use DLs to describe the semantic of constraints between features. The rules are defined in DLs knowledge base.

Alternative-Rule: $\forall f1 \forall f2 \forall f3\ \text{DimFeature}(f1) \wedge \text{DimValue}(f2) \wedge \text{DimValue}(f3) \wedge \text{hasAlternativeValue}(f1,f2) \wedge \text{hasAlternativeValue}(f1,f3) \rightarrow \text{mutex}(f2,f3)$

Mutex-Rule: $\forall f1 \forall f2\ \text{Feature}(f1) \wedge \text{Feature}(f2) \wedge \text{hastate}(f1,\text{bound}) \wedge \text{mutex}(f1,f2) \rightarrow \text{hasState}(f2, \text{removed})$

The alternative-rule indicates that *f2* and *f3* are mutually exclusive.

The mutex-rule means that *f1* and *f2* are instances of *Feature*, and they are mutually exclusive, if *f1* is bound, then *f2* must be removed.

The role *hasMandatoryPart* expresses the dependency constraint implicatively. That is, feature instance *f1* has a mandatory child feature *f2* means that *f2* depends on *f1* (see require rule1). If *f2* is bound, then *f1* must be bound (see require rule2). If *f1* is removed, then *f2* must be removed (see require rule3).

Require-Rule1: $\forall f1 \forall f2\ \text{Feature}(f1) \wedge \text{Feature}(f2) \wedge \text{hasMandatoryPart}(f1, f2) \rightarrow \text{require}(f2, f1)$

Require-Rule2: $\forall f1 \forall f2\ \text{Feature}(f1) \wedge \text{Feature}(f2) \wedge \text{hasState}(f2,\text{bound}) \wedge \text{require}(f2, f1) \rightarrow \text{hasState}(f1,\text{bound})$

Require-Rule3: $\forall f1 \forall f2\ \text{Feature}(f1) \wedge \text{Feature}(f2) \wedge \text{hasState}(f1,\text{removed}) \wedge \text{require}(f2, f1) \rightarrow \text{hasState}(f2,\text{removed})$

D. Reasoning about model for its verification

There are some basic inference issues considered in the reasoning about consistency and completeness of feature model. In order to reason about consistency, we define the state conflict by using the following conflict rule, which indicated that a feature instance *f1* has the two states *bound* and *removed* at the same time, then *f1* has the state *conflict*.

Conflict-Rule: $\forall f1\ \text{Feature}(f1) \wedge \text{hasState}(f1,\text{bound}) \wedge \text{hasState}(f1,\text{removed}) \rightarrow \text{hasState}(f1,\text{conflict})$

Definition 1. consistency

For knowledge base K=(T,A), the ABox A, in which the feature model is described, is consistent with respect to (w.r.t.) the TBox T if there is no state *conflict*.

Definition 2. completeness

For K=(T,A), the ABox A, in which the feature model is described, is complete w.r.t. the TBox T if all the assertions necessary are included.

IV. CASE STUDY

We take the graph editor system as an example, and analyze its feature modeling and verification based on DLs. The graph editor is used for manipulating and displaying graphs in domains such as CAD/CAM. It is a relatively simple, easy to understand domain system.

A. Semantic model of graph editor

Though the notations of the graphs have different semantic, we can abstract features, which can be shared among all graph software.

In graph editor, graph manipulation is a key function; it controls the lower level functions such as graph addition, selection, deletion, moving and composition. These lower level functions are bound in reuse time, and graph composition is optional. Figure 1 depicts its typical feature model.

B. Reasoning about feature model of graph editor

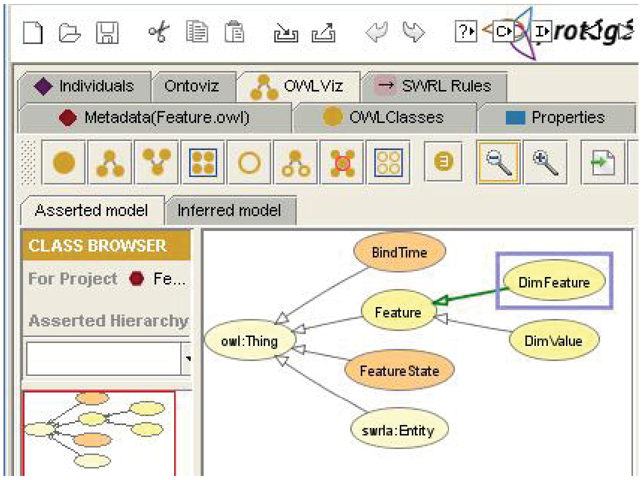


Figure 4. Feature meta-model definition in Protégé.

We adopt Protégé3.4.4[2] as OWL editor, Jena[3] as ontology API. Jena is a Java framework for ontology application. It provides a programmatic environment for OWL, query language SPARQL[4] and includes a rule-based inference engine and SPARQL query engine.

First, we create the feature meta-model according to the DLs-based modeling method (see Section 3.2) in Protégé. That is, we define the concepts, roles and constraints (see Figure 4).

Second, we use rule language to describe all rules.

[2] http://protege.stanford.edu/
[3] http://jena.sourceforge.net/
[4] http://www.w3.org/TR/rdf-sparql-query/

Third, we present the feature model of the graph editor by adding assertions (refer to the tutorial[5] for use of protégé).

Last, we reason about feature model to verify its consistency and completeness by using reasoner and SPARQL.

We get feature instances whose state is conflict by following SPARQL: "SELECT ?x WHERE {?x hasState conflict}"

Case 1: Suppose ABox *A*={*Feature*(*graphManipulate*), *Feature*(*graphDelete*), *Feature*(*graphSelect*), *hasState*(*graphDelete*, *bound*), *hasState*(*graphSelect,removed*), *require*(*graphDelete*, *graphSelect*)}. Using SPARQL, we find two conflict feature instances *graphDelete* and *graphSelect* according to require-rule2, require-rule3 and conflict-rule.

Case 2: Suppose ABox *A*={ *Feature*(*graphMove*), *DimFeature* (*movingMode*) , *hasState*(*movingMode,bound*), *hasMandatoryPart* (*graphMove*, *movingMode*), *DimFeature*(*graphDim*), *DimValue*(*2D*), *DimValue*(*3D*), *hasState*(*3D,bound*), *hasAlternativeValue* (*graphDim,2D*), *hasAlternativeValue*(*graphDim,3D*), *mutex* (*graphMove*, *3D*)}. Using SPARQL, we find three conflict feature instances *3D*, *graphMove* and *movingMode* according to mutex-rule, require-rule3 and conflict-rule. Feature instances *3D*, *graphMove* are conflict because they are mutually exclusive. The feature instance *movingMode* is conflict, because new assertion "*hasState*(*movingMode,removed*)" is derived according to require-rule3, the assertion is inconsistent with "*hasState*(*movingMode,bound*)".

By using reasoner, we can verify the consistency of feature model, or achieve new assertions to make the feature model more complete.

V. RELATED WORK

Current methods have some limitations. The relationship between features and constraints among them are similar but different. For example, the variation-variant features (in FeatuRSEB [3]) are equivalent to dimension-value features (in FODM [10]); and XOR-relation (in FeatuRSEB) and alternative-relation (in Czarnecki's proposal [13]) are similar. Furthermore, most proposals depict feature models in a grapy way by using slightly different notations. They are essential and easy to comprehension. However, they lack semantic, and this has led to the redundancy and confusion.

Ontology-Oriented requirement analysis (OORA) [14] method enhances the object-oriented analysis and design. In FODM, propositional logic is selected separately to model and verify constraint relations. Kaiya *et al* [15] proposed a method of software requirements analysis based on ontologies, where inference rules is established to detect incompleteness and inconsistency in a specification. Peng *et al* [8] proposed an ontology-base feature meta-model supporting application-oriented tailoring. Some constraints rules are introduced to validate constraints by ontology reasoning. Benavides *et al* [9] used constraint programming to model and reason on a SPL. The model need to be extended to support dependencies such as requires or excludes relation. To the best of our knowledge [16], there are only small numbers of proposals that treat automatic reasoning about feature models to verify the constraints. Many ontology-based approaches do not differentiate between feature meta-model and feature models.

VI. CONCLUSIONS

In this paper, we propose a DLs-based method to model feature: describing feature meta-model with concepts, roles, axioms and rules in TBox, while describing feature model with assertions in ABox. We can reason about the feature model to verify the consistency and completeness. A case study in graph editor domain shows that this method will be beneficial.

The main features of this method are as follows: i) our feature model is compatible with the model that adopted by most of domain engineering methods. ii) The explicit semantic clarifies the similarity and differences among these methods. iii) Concrete feature models are instantiated in ABox, so it is convenient to perform running-time verification. Still there are some weaknesses: some non-functional features are not taken into considerations; how to elicit feature in a domain depends on expertise experience.

ACKNOWLEDGMENT

This research was supported by the National High-Tech R&D Program of China (No. 2009AA010307), and by Fundamental Research Funds for the Central Universities (No. NS2012022). We thank Prof. David Parnas, Jie Chen for their helpful suggestions.

REFERENCES

[1] K.C. Kang, S.G. Cohen, J.A. Hess, W.E. Novak, and A.S. Peterson. Feature-Oriented Domain Analysis Feasibility Study [R]. Technical Report(SEI-90-TR-21), CMU, 1990.

[2] K.C. Kang, S. Kim, J. Lee, K. Kim, E. Shin, and M. Huh. FORM: A Feature-Oriented Reuse Method with Domain-Specific Architecture [J]. Annals of Software Engineering, vol. 5, pp. 143-168, Sept. 1998.

[3] M.L. Griss, J. Favaro, and M. d'Alessandro. Integrating Feature Modeling with the RSEB [C] //Proc of ICSR 1998, pp. 76-85,1998.

[4] 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[C] //Proc of SSR'99, May 1999.

[5] P. Clements, L.M. Northrop. Software Product Lines: Practices and Patterns [M]. Addison-Wesley, 2001.

[6] H. Wang, Y.F. Li, J. Sun, H. Zhang, and J. Pan. Verifying Feature Models using OWL. Journal of Web Semantics, 5:117–129, June 2007.

[7] S. Fan and N. Zhang. Feature model based on description logics. In Knowledge-Based Intelligent Information and Engineering Systems, 10th Intl. Conf., KES, Part II, volume 4252 of Springer–Verlag, 2006.

[8] X. Peng, W.Y. Zhao, Y.J. Xue, Y.J. Wu. Ontology-Based Feature Modeling and Application-Oriented Tailoring [C] //Proc of ICSR 2006. pp. 87-100, LNCS, 2006.

[9] D. Benavides, P. Trinidad, and A. Ruiz-Cortes. Automated Reasoning on Feature Models. //Proc of CAiSE 2005. pp.491-503, LNCS, 2005.

[10] W. Zhang, H. Mei. A Feature-Oriented Domain Model and Its Modeling Process[J]. Journal of Software, 2003,14(8): 1345-1356(in Chinese).

[11] W. Zhang, H.Y.Zhao, H. Mei, A Propositional Logic-Based Method for Verification of Feature Models, ICFEM 2004 (LNCS, 3308) , pp. 115-130,2004.

[12] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, P. Patel-Schneider. The description logic handbook: theory, implementations and applications [M]. Cambridge University Press, 2003.

[13] K. Czarnecki and U.W. Eisenecker. Generative Programming: Methods, Techniques, and Applications. Addison–Wesley, may 2000.

[14] R.Q. Lu, Z. Jin. Domain Modeling-Based Software Engineering: A Formal Approach [M]. Kluwer Academic Publishers, pp. 1-347, 2000.

[15] H.Kaiya, M.Saeki. Using Domain Ontology as Domain Knowledge for Requirements Elicitation [C] //Proc of RE'06, pp. 189-198, IEEE C.S., 2006.

[16] D. Benavides, S. Segura, A. Ruiz-Cortés: Automated analysis of feature models 20 years later: A literature review. Information Systems 2010,35(6):615-636

[5] http://protege.stanford.edu/doc/users.html#tutorials

A Context Ontology Model for Pervasive Advertising: a Case Study on Pervasive Displays

Frederico Moreira Bublitz[1], Hyggo Oliveira de Almeida[2], and Angelo Perkusich[2]
fredbublitz@uepb.edu.br, hyggo@dsc.ufcg.edu.br, perkusic@dee.ufcg.edu.br
[1]State University of Paraiba, [2]Federal University of Campina Grande
Campina Grande, Brazil

Abstract

Context awareness is a key concept for Pervasive Advertising. However, the literature indicates that there is a lack of high level abstraction models that enable applications establish a common vocabulary to share contextual information. In this paper we present an ontology model for pervasive advertising. The main aspects of this model is that it is comprehensive and extensible. Its feasibility is demonstrated in a case study on a pervasive display scenario.

1. Introduction

As a new channel for communication, Pervasive Computing [1] offers the opportunity of delivering advertisements at anytime and anywhere. Once these advertisements can be delivered through personal devices, this paradigm enables to achieve a level of audience that was never imagined before. More than a new channel for delivering advertisements, the most important aspect of Pervasive Advertising is that it enables the delivering of contextualized advertisements [2].

An advertisement is contextualized if it is delivered in accordance with the context of the user to became more appropriated and adapted to the situation. In pervasive computing paradigm, context is defined as any information that applications can use to provide relevant services or information to an entity [3]. In case of advertising applications, the context is defined as any information used to determine the relevance of an advertisement for a user. For example, how an application knows if an advertisement of a happy hour promotion is relevant to you? To answer this question the application must know about your preferences, location, activity, friends, and so on.

The contextual information can be obtained from distinct and heterogeneous sources. This includes the information that the user can explicitly provides, and the information that can be obtained from a device, either by sensors or inferred from other information. In this way, it is necessary to provide an unified and high level abstraction model to enable applications to "understand" the meaning of the collected information and also to establish a common vocabulary to share this information [4].

As can be noticed, to be aware of the context is important for effective delivery of contextualized advertisements. However the literature indicates that there is a lack of high level abstraction contextual models for Pervasive Advertising. The reason for this absence of effective solutions occurs because it is almost impossible to define *a priori* what kind information should be used to represent the context, once it is closely related the product to be announced. For example, the body temperature of a person may be a relevant information in order to deliver an advertisement in health care domain, but it can be unnecessary for determine the relevance of an announcement of a car.

Notice that this problem is equivalent to the problem of create a model to represent context in Pervasive Computing. Several works ware developed over this problem, but a deeper analysis reveals that this is still an opened problem. The existent approaches can be comprehensive, but it is possible to notice that the kind of information modeled in these works is strongly coupled to some domain of application. Other works uses a synthetic set of information that can be used in most domains of applications, but this is not efficient in practice [5], see more details in Section 2.

In view of the above, in this paper we present a high level abstraction model for context in Pervasive Advertising, described in Section 3. The main features of this model are: (i) it is comprehensive; and (ii) it is loose coupled. That is, it contemplates many domains of information associated with the products and can be easily extended to contemplate new concepts.

The feasibility of the model is demonstrated through a case study on a pervasive display scenario, detailed in Section 4. In this scenario, the content of a public display is selected according to the context of the group of consumers in the vicinity of the display. The information about whom is in the vicinity of the display is acquired by the detection of their devices through a symbiotic relation among them.

2. Related Work

Due to the importance of context for Pervasive Computing as a whole, many works has been carried out with the intention of represent the contextual information from the environment. In a general way, these works can be grouped into two categories: (i) the comprehensive works - where the authors try to gather the maximum of information for represent context, but always according with the scope of some application being developed [6, 7, 8, 9]; (ii) the synthetic works - which try to find a minimal set of information that can be used in most applications [10, 11, 12].

The problem in adopt these works in Pervasive Advertising is that when the developer adopt a comprehensive model, add a new concept, or a task ontology, is not possible, once there is a strong coupling between the existent classes. Still, if the developer tries to adopt a work that uses the synthetic approach, the minimal modeling will force him to extend the solution for contemplate each kind of product that will be announced, and eventually, when many concepts are added to the model, the developer will find the same problem from comprehensive approach.

The main problem with these works is that they cant be applied to the Pervasive Advertising domain, and adapt them to this domain is a hard task due to the high level of coupling of the information. We also detach that still there are not a comprehensive model created for the Pervasive Advertising [4], which be able to represent specif information of this domain such as the target audience, and the message of the ad.

3. The Context Model

To be aware of context is a key concept in order to effectively develop applications in the scope of pervasive computing. A good modeling formalism reduces the complexity of developing context-aware applications by improving their maintainability and evolvability. Considering that the growing trend of use the multiagent approach for implement Pervasive Advertising systems, the need of built-in semantics and expressiveness and the need of computational guarantee, we decided to adopt *OWL DL Ontologies* for modeling the contextual information.

The conception of a model for Pervasive Advertising is not an easy task. Mainly because (i) this model must be comprehensive, in order to address the different needs of contextual information of applications; (ii) this model must be loosely coupled, in order to be extensible, in other words, the incorporation of new concepts still absent shall be easy. Notice that there is a trade off between the need of low coupling and comprehensiveness. Once, raising the level of comprehensiveness increases the coupling, and vice-versa.

To solve this impasse, we did not try to avoid coupling, it is impossible, but we change the way the coupling occurs to make it weaker. The key to achieve the result is the association of the contextual information with the entities capable of provide these information. These entities are the *users* and the *devices*, once they are the only entities capable of produce information.

After identified the sources, it is possible to associate other concepts to them. In this way, the coupling with these entities is not a problem anymore, once the information about them are necessary for any application that uses context in Pervasive Computing paradigm. Following this line of thought, our model was created using a gradual approach, and is divided into three layers, they are: the **Kernel**, **Pervasive**, and **Advertising** layers following described.

3.1. Kernel Layer

In this layer are represented the entities capable of provide information in scope of Pervasive Computing, they are the *User* and *Device*. The Figure 1 (yellow boxes) illustrates how the context can be represented in terms of users and devices. It is also possible to notice in this figure that a user can holds many devices. In this layer there is yet the representation of location, once this information is directly associated to the user and device.

It is also common to think in the environment as a source of information. But, actually the environment is only a representation of a *location*, a place. In this way the information about the environment can be obtained through the devices that are in a specific location. This means that all the information can be obtained either from the user or the device. For example, the information about the gender of a person can be directly provided by him or herself. Or the information about the temperature of a place can be obtained by a sensor (device).

Other aspect is the specification of the *Device Profile* that can be a hardware or a software profile. The *Hardware Profile* describes specific characteristics of a device, such as memory and processor power, battery level, and screen size. The *Software Profile* follows the same line of though, including the characteristics of software of a device like browser support and characteristics.

3.2. Pervasive Layer

The Pervasive layer gathers the information that can be relevant for most applications from different domains. That

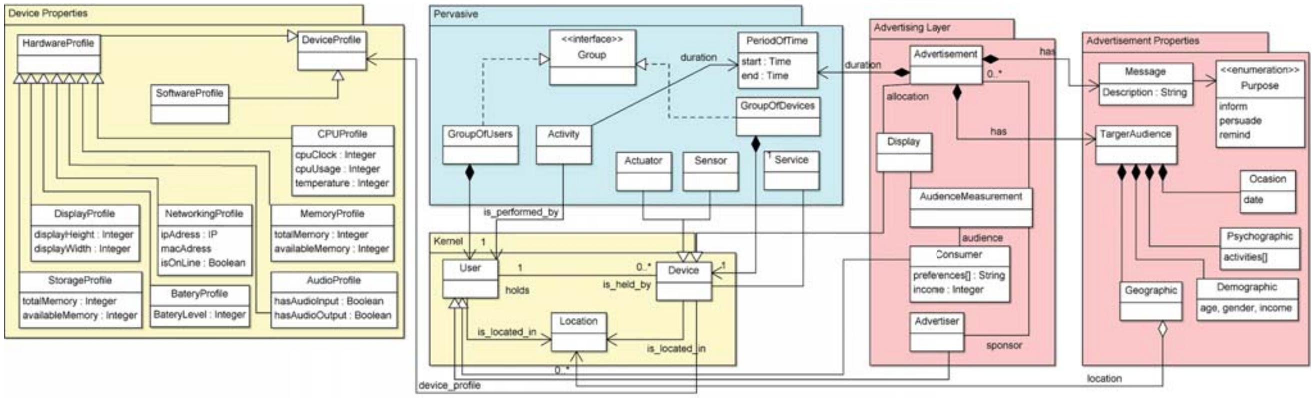


Figure 1. UML view of the model. The yellow boxes represent the Kernel layer and the device properties; the green box represents the Pervasive layer; and the red boxes represent the Advertising layer the advertising properties. This model view is simplified containing only the most important classes and attributes.

is, we believe that is possible to model the contextual information according to aspects that are useful for many application domains instead of focuses in specific domains of applications. For example, the services available in the location is an information useful for applications of different domains.

Following this point of view, we can think that is possible to classify the context in classes, as follows:

Group of Users: People usually establish relationships with other people, either by sharing the same location (e.g., workplace, city) or because they have common interests. In this work, a *group of users* consists of people who have something in common such as location, and interests. Through this concept is yet possible to deal with the concept of *social networking*. Social Network is a form of representation of emotional or professional relationships of human beings among themselves or among their groups of mutual interests. The same idea can be applied to the **devices**, in which a device can form a group with other devices that have characteristics in common. For example, a device can form a group with the devices that are accessible to a particular user or with the devices needed to supply a particular service.

Service: in a pervasive environment, it is common that devices act as services providers. This service provision is related to service-oriented architecture (SOA). By focusing on functionality, the SOA architecture allows heterogeneous applications can establish relations between them. Therefore, it is important for a great number of applications executing in pervasive environments be aware of the services that each of the available devices provides.

Sensor and Actuator: these are special kinds of devices, and their representation is useful when it is necessary to deal with the idea of **environment**. In this representation the environment is a kind of derivate information, which can be obtained when combined with the *location*.

Time and Period: the notion of time is useful for applications once it introduces the concept of period. This is yet useful yet to establish a chronological order of the facts, a **history**.

Activity: the activity of a user is an information very useful for application that intend to deal with personalization of services.

In the Figure 1 (green box) these concepts and their relationships are represented. Notice that it is possible to exist others classes useful for pervasive applications, such as *Event*. However, in practical terms it is impossible to cover all the classes, for this reason we did not make an exhaustive list. We decide to focuses on the most useful classes, once the incorporation of new classes can be easily done by extending the model. What make this model easy of extending is the way the classes are coupled, notice that none of the classes on kernel layer make reference to the others layers. This means that a modification in the Pervasive layer does not affect the kernel layer.

3.3. Advertising Layer

The advertising layer was created to contemplate specifics characteristics of Pervasive Advertising. In Figure 1 (red box) the main concepts related with the advertisement task are presented. Following we give a description of them.

When developing a communication and promotion program, marketers must take into account [13]:

- The *Message* of the advertising campaign: all advertisement must transmit a message to its audience during a defined time period (*duration*). This message is propagated with a specific objective that can be classified by primary purpose - whether the aim is to *inform*, *persuade*, or *remind*.
- The *Target Audience*: it is important for the success of a marketing campaign, to define for who it will serve. This can be done by diving the marketing into segments of customers (Marketing Segmentation). In this sense, consumers can be grouped and served in various ways based on *Geographic*, *Demographic*, *Psychographic*, and *Behavioral* factors.

The model also contemplates a special kind of device, the *Display*, which distinguishes the devices capable of transmit a message for consumers. Notice yet, that there is a distinction between the *Advertiser* and *Consumer*, which are both users. This distinction is important to allow application to measure the audience.

4. Case Study

Currently, there are environments (e.g., universities, airports and malls) in which public devices (e.g., LCD televisions) are positioned at strategic points such as elevators, hallways and showcases. These devices are used with the purpose of promoting products and services, although acting as a form of entertainment to the people who frequent such environments. This form of ad serving is also known as *Digital Signage*.

Although these devices can be able to achieve a lot of consumers, advertisers have little information from users who are present in the environment, making difficult the ads to be more relevant to those users. For example, in a university is common that things related to education be announced in these displays. It is also common that classmates walk together. So, for a group of students of a computer science course an ad of a product that is related to the field of dentistry, probably will have no relevance. Hence, the audience is going to be low.

We believe that the technological resources that enable this kind of media (i.e., the indoor media) can be improved so that the advertised products and services can be customized to adapt to the context of people who are in the environment. For this, it is necessary to have information about the context of the users. It is necessary to know the location of users, because this can help to know “ who” is located next to such a display, and the profile of users. In addition, there must be a treatment of the context of the users, not only individually, but as a group, where the interests of individuals no longer prevail in advantage of the interests of the group.

4.1. Scenario

This case study was carried out in a university where a series of *displays* are positioned in strategic points like in the elevators, library, living room and coffee room. These displays are used to present cultural programming, to exhibit news about the university, and delivery ads.

In our scenario, the students were encouraged to fulfill a form with some personal information related to their profile and identifying their *bluetooth* devices. After this, once their presence is detected, the system processes what ads are more suitable to the group of students in the vicinity of the display, and the ads are exhibited, as illustrated in Figure 2.

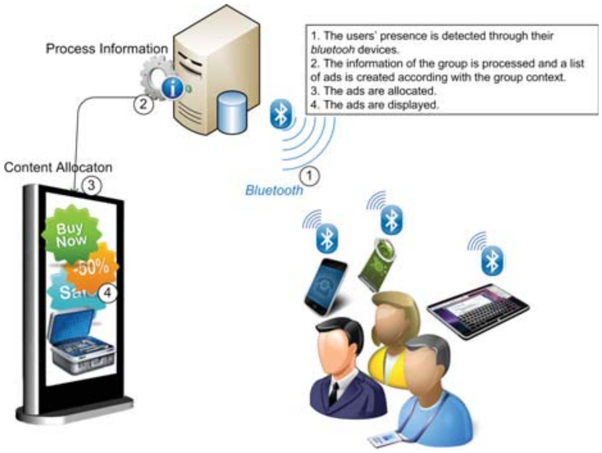

Process Information
1. The users' presence is detected through their *bluetooh* devices.
2. The information of the group is processed and a list of ads is created according with the group context.
3. The ads are allocated.
4. The ads are displayed.
2
Content Allocaton
3
1
Bluetooth
Buy Now
-50%
Sa
4

Figure 2. Case study scenario: the advertisements are exhibited according to people in the vicinity of the display.

4.2. Architecture

The designed architecture is showed in the Figure 3. As can be noticed in this figure, the architecture is divided into layers, as follows.

Persistence: in the persistence layer we have the **server**, which is used to store both advertisement and user information. Notice that the description of the stored information is given by the ontology created in this work.

Information Representation: all the dynamic information is processed in this layer. This includes the **device context**, the **user context**, and the **group context**.

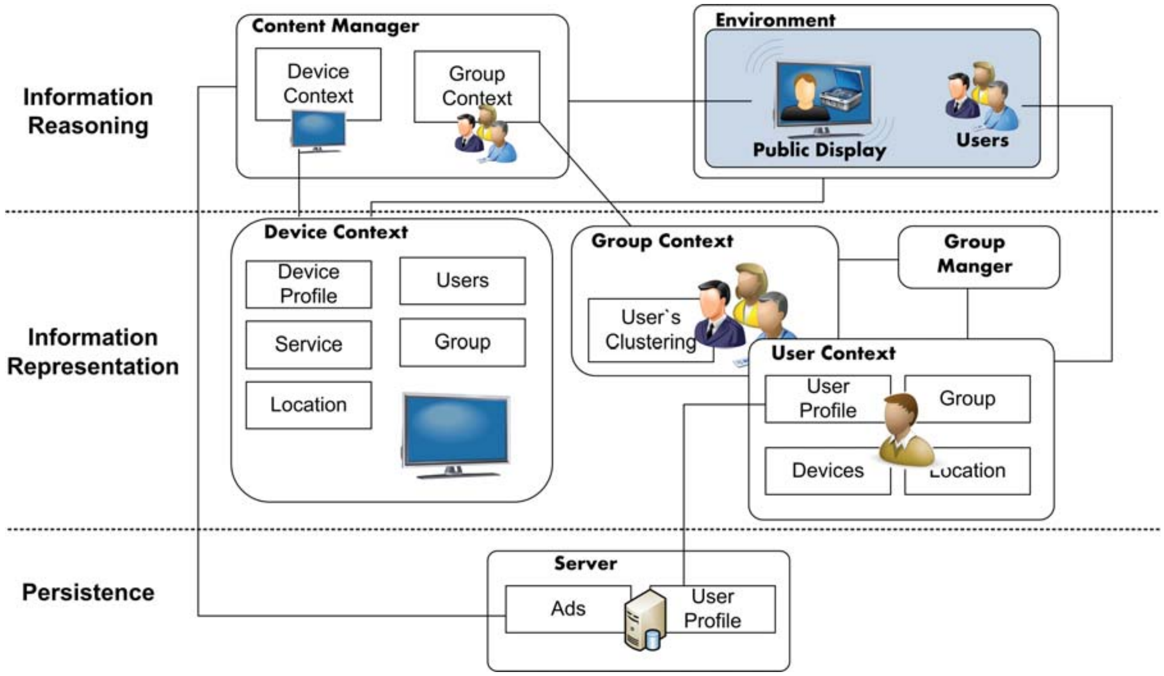


Figure 3. Architecture for delivering context-aware advertisements on a pervasive display scenario scenario.

Notice that the formation of the group is made by the **group manager** that is responsible by treat the context of a group of individuals.

Information Reasoning once we have the information about the users, the ads, and the devices (including the ones used for ad delivery, i.e., displays) it is possible now to allocate the content for the displays. This task is performed by the **content manager** module and this task is performed according with the location of the displays and the users, in what emerges the notion of **environment**. The environment represents the scenario, where the display and the group of users are located and the content is delivered.

4.3. Implementation

The implementation of this prototype was done by using an object oriented approach. The chosen language was *Java* and some design patterns like *Data Access Object* (DAO), *Model-View-Control* (MVC) and *Facade* were applied. To identify the users we adopt the *Bluetooth* technology, because it enables a good accuracy in determine the location of a device.

Through the *Bluetooth* technology it is possible to determine what users are in the vicinity of a display. This is done by getting the MAC (Media Access Control) address of a device which is unique and can be associated with the user. The searching for devices is done by using the *bluecove* API. In the Listing 1 a piece of code used to search *bluetooth* devices.

Listing 1. Search for *bluetooth* devices.

```
public void run(){ //Starting Search
  final Object inquiryCompletedEvent = new Object();
  DiscoveryListener listener = new DiscoveryListener(){
    public void deviceDiscovered(RemoteDevice btDevice,
                                 DeviceClass cod) {
      device = new Device();
      device.setMac(btDevice.getBluetoothAddress());
      try {
        device.setName(btDevice.getFriendlyName(true));
      } catch (IOException cantGetDeviceName)
      devices.add(device);
   }
```

During the development of the case study, we have some problems with *bluetooth*. The main problem is that the device discovery is quite slow, this means that state used to generate the group context can be not exactly the real state. To minimize this problem, we start a new search at the end of the last advertising, to this we adopt video (similar to the ones that plays in TVs) so that the gap between the discovery and the playing of the video be minimized.

4.4. Case Study Analysis

The developed case study was conceived to enable more relevant ads in the context of a university. In this sense, the profile of the users was defined in terms of *gender*, *age*, and *course*. We made an experimental in order to verify the hypotheses that the proposed architecture can bring ads more relevant than a random approach. The hypotheses are the following:

Null hypothesis - H_0: The relevance of the ad (represented by Φ) offered by this model ($this$) is, at most equivalent to the random delivery($random$).
$H_0 : \Phi_{this} \leq \Phi_{random}$

Alternative hypothesis - The relevance of the ad (represented by Φ) offered by this model ($this$) is superior to the random
$H_1 : \Phi_{this} > \Phi_{random}$

During a period of seven days, the ads ware exhibited in the coffee shop of the university. In this period the same amount of ads ware displayed using both approaches, the random and the one here described. The results show that we achieve a considerable improvement in the relevance of the ads, confirming the *alternative hypothesis*. More precisely, from the total ads sent, our model achieved an accuracy rate of *64,18%*, versus *36,05%* from random.

5. Conclusions

In this work we presented a novel model to represent context in Pervasive Advertising. The main advantage of this model is that it defines context in terms of user and device. The big benefit of this new way of organizing information is that it allows that new classes of information are added to the model without causing major changes, making the model easy to extend. In other words task ontologies can be easily added to the model. This is possible because the coupling occurs from the classes of information to the source of information, leaving the classes loosely coupled with each other.

To demonstrate the feasibility of the model, we developed a case study on a pervasive display scenario. Through this case study was possible to notice that the model is very comprehensive. This can be notice because the products advertised were from many different fields (such as sports, computing, and beauty) and the model was used without changes or extensions.

In this case study, was possible also to improve the effectiveness of the advertisements in more that 77%. We notice yet, that the results could be still more expressives if our base of advertisements was more comprehensive and if the *bluetooh* discovery was not so slow.

As current/future work, we are developing: (i) an application for a Digital Signage scenario, that is not intrusive; (ii) an application for an opportunistic Mobile Advertising scenario; and (iii) a multiagent approach for Pervasive Advertising, which contemplates the delivery of advertisement in both public displays and the mobile personal devices.

References

[1] M. Weiser, "The Computer for the 21st Century," *Scientific American*, vol. 265, pp. 66–75, September 1991.

[2] J. Krumm, "Ubiquitous Advertising: The Killer Application for the 21st Century," *Pervasive Computing, IEEE*, vol. PP, no. 99, pp. 1–16, 2010.

[3] A. K. Dey, "Understanding and Using Context," *Personal and Ubiquitous Computing*, vol. 5, no. 1, pp. 4–7, 2001.

[4] M. Strohbach, M. Bauer, M. Martin, and B. Hebgen, *Pervasive Advertising*, ch. Managing Advertising Context, pp. 185–205. Springer, 2011.

[5] F. Bublitz, E. Loureiro, H. Almeida, A. Perkusich, and E. de Barros Costa, "Context-Awareness in Pervasive Environments," in *Encyclopedia of Networked and Virtual Organizations* (M. M. Cunha, ed.), vol. 1, pp. 1958–1959, Hershey, PA, USA: Idea Group Publishing, 2008.

[6] H. Chen, F. Perich, T. Finin, and A. Joshi, "SOUPA: Standard Ontology for Ubiquitous and Pervasive Applications," in *International Conference on Mobile and Ubiquitous Systems: Networking and Services*, (Boston, USA), August 2004.

[7] D. J. Cho and M. W. Hong, "A Design of Ontology Context Model in Ubiquitous Learning Environments," in *ICCOMP'08: Proceedings of the 12th WSEAS international conference on Computers*, (Stevens Point, Wisconsin, USA), pp. 844–848, World Scientific and Engineering Academy and Society (WSEAS), 2008.

[8] A. Esposito, L. Tarricone, M. Zappatore, L. Catarinucci, and R. Colella, "A Framework for Context-Aware Home-Health Monitoring," *Int. J. Auton. Adapt. Commun. Syst.*, vol. 3, no. 1, pp. 75–91, 2010.

[9] L. Seremetia, C. Goumopoulosa, and A. Kameas, "Ontology-based Modeling of Dynamic Ubiquitous Computing Applications as Evolving Activity Spheres," *Pervasive and Mobile Computing*, vol. 5, pp. 574–591, 2010.

[10] F. Bublitz, H. Almeida, A. Perkusich, E. Loureiro, E. Barros, and L. Dias, "An Infrastructure for Developing Context Aware Applications in Pervasive Environments," in *SAC '08: Proceedings of the 2008 ACM symposium on Applied computing*, (New York, NY, USA), pp. 1958–1959, ACM, 2008.

[11] Y. Hu and X. Li, "An Ontology Based Context-Aware Model for Semantic Web Services," *Knowledge Acquisition and Modeling, International Symposium on*, vol. 1, pp. 426–429, 2009.

[12] C.-H. Liu, K.-L. Chang, J. J.-Y. Chen, and S.-C. Hung, "Ontology-Based Context Representation and Reasoning Using OWL and SWRL," *Communication Networks and Services Research, Annual Conference on*, vol. 1, pp. 215–220, 2010.

[13] P. Kotler and G. Armstrong, *Principles of Marketing*. Prentice Hall, 12th edition ed., Feb. 2007.

Ontology–based Representation of Simulation Models

Katarina Grolinger, Miriam A. M. Capretz
Department of Electrical and Computer Engineering,
Faculty of Engineering
The University of Western Ontario
London, ON, Canada N6A 5B9
{kgroling, mcapretz}@uwo.ca

José R. Marti, Krishan D. Srivastava
Department of Electrical and Computer Engineering,
Faculty of Applied Science
The University of British Columbia
Vancouver, BC, Canada V6T 1Z4
jrms@ece.ubc.ca, kd@interchange.ubc.ca

Abstract—**Ontologies have been used in a variety of domains for multiple purposes such as establishing common terminology, organizing domain knowledge and describing domain in a machine-readable form. Moreover, ontologies are the foundation of the Semantic Web and often semantic integration is achieved using ontology. Even though simulation demonstrates a number of similar characteristics to Semantic Web or semantic integration, including heterogeneity in the simulation domain, representation and semantics, the application of ontology in the simulation domain is still in its infancy. This paper proposes an ontology-based representation of simulation models. The goal of this research is to make use of ontologies to facilitate comparison among simulation models, querying, making inferences and reuse of existing simulation models. Specifically, such models represented in the domain simulation engine environment serve as an information source for their representation as instances of an ontology. Therefore, the ontology-based representation is created from existing simulation models in their proprietary file formats, consequently eliminating the need to perform the simulation modeling directly in the ontology. The proposed approach is evaluated on a case study involving the I2Sim interdependency simulator.**

Keywords-Ontology; Simulation Model; Ontology-based Model; Semantic Integration

I. INTRODUCTION

Ontologies are frequently associated with the Semantic Web where computers are capable of analyzing the content, meaning and sem antics of the data and performing the reasoning upon the content. Other ontology applications include data integration, application integration and interoperability, knowledge management, machine learning, information extraction, information browsing and navigation.

Simulation domain exhibits a n umber of similar characteristics to those fields including heterogeneity in the simulation domain, vocabulary, representation and semantics. However, the application of ontology to the field of simulation is still in its infancy and primarily contained within the research community.

The simulation heterogeneity is larg ely caused by its application in a variety of different domains including critical infrastructures, medicine, learning and chemical engineering. Consequently, a number of software simulation packages or simulation engines exist for the support of computer simulations in those domains [1]. Commonly, simulation packages are application-oriented, designed for the use in a specific domain, hence they apply diverse modeling approaches, different technologies, domain specific terminologies and store simulation models and results in a variety of formats. This diversity of application-oriented simulation engines presents a challenge for comparing simulation models and results, reusing and sharing existing models, as well as querying and making inferences.

The objective of this work is to address the following challenges of the application-oriented simulation approach:

- The extraction of specific information from model files or from simulation results is not straightforward. Simulation packages may provide basic information, nevertheless, the extraction of more detailed or specific summary information becomes demanding.
- The comparison between models of a single simulation engine or different engines is difficult. Typically the comparison relies on the simulation engine to provide the means for comparing specific pairs of model files.
- The comparison between results of different simulation runs of the same simulation engine or different engines is a challenging endeavor. Simulation packages focus on providing performance measures for a single simulation run while the comparison between simulation runs often requires external tools and a significant manual effort.

As a s olution, this paper proposes the representation of domain simulation models as instances of Simulators' Ontologies. By using the same formalism to represent various simulation models, we place them on the same platform, thus enabling a s implified comparison. Moreover, ontology-based representation allows for inquiries with ontology querying languages and inferences with ontology reasoners. The proposed approach uses existing models in the simulation engine proprietary file formats as the foundation for the creation of its ontology-based representation.

The remainder of the paper is organized as follows: Section II reviews related works, the proposed system is portrayed in Section III, while Section IV depicts a case study. Finally, the conclusions and future work are presented in Section V.

II. RELATED WORKS

Ontology can be described as an abstract, machine-readable model of a phenomenon that identifies the relevant concepts of that phenomenon as well as the relations among them. Furthermore, ontologies represent a way of establishing common terminology, organizing domain knowledge and representing this information in a machine-readable form. The potential use of ontologies in simulation and modeling is explored by Lacy and Gerber [2]. From the perspective of these authors, ontologies are beneficial in simulation and modeling through the formalization of semantics, the ability to query and inference, and the sharing and reuse of developed models.

Studies that are especially relevant to our research are related to the use of ontologies to represent real world scenarios for the simulation purposes such as Tofani et al. [3], Miller et al. [4] and Silver et al. [5].

Tofani et al. [3] use the ontology framework to model the interdependencies among critical infrastructures (CI). Their proposed framework consists of three ontologies: WONT (World ONTology) contains concepts and relations that are common across CI domains; IONT (Infrastructure ONTology) extends WONT to represent the knowledge of specific CIs and FONT (Federation ONTology) enables modeling relations among different infrastructures. The CI network is modeled twice: as instances of the ontology and in the simulation language of the domain. The mapping between ontology representations and simulation models is established manually.

Miller et al. [4] investigate the development requirements and benefits of ontologies in discrete event simulation (DES), and consequently, these authors present the Discrete-event Modeling Ontology (DeMO). The proposed DeMO consists of four main classes: DeModel, ModelComponent, ModelMechanism and ModelConcept. DeModel is composed of ModelComponents and activated by the ModelMechanism, while the ModelConcepts serve as a terminology upon which other classes are built. The main challenges in building DeMO, or a similar ontology for simulation and modeling, are twofold [6]; firstly, it needs to be domain-independent, as a DES can model any domain. Secondly, since simulation formalisms are founded in mathematics and statistics, the DES ontology should be based upon the ontologies of those domains.

Silver et al. [5] represent simulation models as instances of the extended DeMO PIModel (Process Interaction Model). In the proposed approach, reality is first represented as instances of the DeMo PIModel ontology. Subsequently, these DeMo PIModel instances are transformed to XPIM (Extensible Process Interaction Markup) instances, which are then translated to a JSIM (Java-based SIMulation) model.

Benjamin and Akella [7] use ontologies to facilitate semantic interoperability and information exchange between simulation applications. The ontology models for each simulation application domain are extracted from textual data sources, such as requirements and design documents. Subsequently, the established ontology mappings represent the translation rules for the ontology-driven translator, which facilitates information sharing between simulation applications.

III. Simulation Models as Instances of an Ontology

This section describes the fundamentals of the proposed system: system architecture, relations and model hierarchies definition, querying ontology-based models and the process of creating ontology-based simulation models.

A. System Architecture

The ontology-based representation of simulation models has a layered architecture, as described in Fig. 1.

The top layer consists of the upper ontology, which contains generic concepts that are common for all simulation engines.

The next architecture layer, the Simulators' Ontologies layer, extends the upper ontology in order to describe specifics of each simulator. Thus, in this layer, there in an ontology for each simulator involved. The terminology matches that of the simulators, facilitating domain experts' understanding of ontologies as well as enabling the creation of the ontological representations from the simulators' models.

The third layer, the ontology-based simulation model layer, contains ontology-based simulation models that are represented as instances of Simulators' Ontologies. More specifically, each simulation model, usually contained in a simulation engine proprietary file format, is represented as an ontology-based model consisting of interconnected instances of the Simulator's Ontology. Different simulation models contained in distinct proprietary files correspond to the various models in this layer.

The bottom architecture layer, or rule layer, is optional and contains a rule engine. In particular, this layer is intended for situations when ontology-based specifications are not sufficient and additional expressiveness is required. Furthermore, this layer expresses design rules and guidelines to which the simulation model should conform.

B. Defining Relations Between Simulation Model Entities

In the ontology-based simulation model, entities are represented as instances of the Simulator's Ontology, while the relations among them are established by means of object properties. The description of the object properties is found in the upper ontology and the Simulators' Ontologies.

Fig. 2 presents a fragment of the upper ontology in RDF/XML syntax. In this ontology the *cell* indicates an entity that performs a function transforming inputs into outputs, *channel* is a mean of transporting entities between *cells* and/or *controls*, while *controls* are entities responsible for distributing the flow among channels. The two object properties, *hasInput* and its inverse *hasEndNode*, are included in the ontology

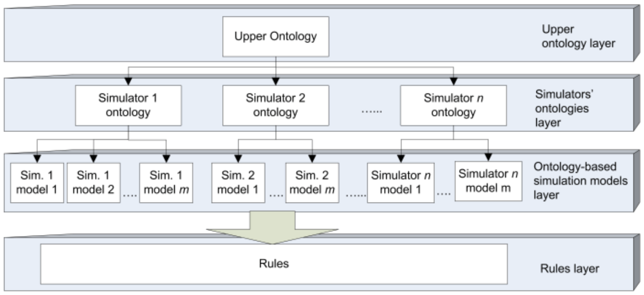
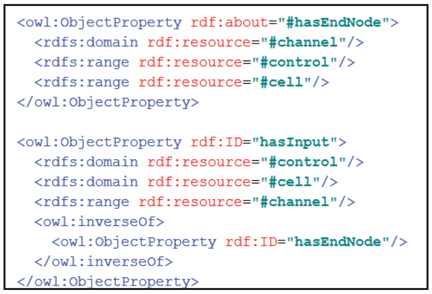


Figure 1. Architecture layers



```
<owl:ObjectProperty rdf:about="#hasEndNode">
  <rdfs:domain rdf:resource="#channel"/>
  <rdfs:range rdf:resource="#control"/>
  <rdfs:range rdf:resource="#cell"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="hasInput">
  <rdfs:domain rdf:resource="#control"/>
  <rdfs:domain rdf:resource="#cell"/>
  <rdfs:range rdf:resource="#channel"/>
  <owl:inverseOf>
    <owl:ObjectProperty rdf:ID="hasEndNode"/>
  </owl:inverseOf>
</owl:ObjectProperty>
```

Figure 2. Upper Ontology fragment

fragment shown in Fig. 2. The domain of the *hasEndNode* property is the *channel* while the range includes the *cell* and the *control*. Since the direction of object properties runs from the domain to the range, the inverse property enables the expression of relations in both directions. The direction that is used will be influenced by the manner in which the relation is expressed in the simulator's model file.

C. Defining Simulation Model Hierarchies

Frequently, to facilitate modeling of complex systems, simulation packages provide the ability to divide models into hierarchies of sub-models [8] as illustrated in Fig. 3. To represent the model hierarchies in ontology, the proposed approach uses the *parentSystem* object property. For each child model, the *parentSystem* property links the model to its direct parent. The set of assigned *parentSystem* properties establishes the model hierarchy. A fragment of a hierarchy depicted in Fig. 3 is represented as: *modelE.parentSystem(modelB)*, *modelB.parentSystem(modelA).* Since the sub-model entities do not belong to any of the Simulator's ontologies classes, we establish a new class *parentSystem* to contain entities that serve as containers for the other entities.

The elements from the parent and the child models are interconnected since they form a single simulation model. The relation between elements from different hierarchy levels is established in the same way as the relation between entities of a single non-hierarchical model as described in Section III.B.

We considered creating a separate ontology for each sub-model which would import ontologies of all its child models. This would establish the ontology hierarchy matching with its equivalent simulation model hierarchy. Simulation sub-models can be as simple as two linked entities that would not warrant the formation of a separate ontology. Nevertheless, simulation models could contain a large number of sub-models resulting in a large number of ontologies for a single simulation model and thus causing maintenance challenges. Therefore, at this stage of our research, we use one ontology to represent one simulation model with all its sub-models.

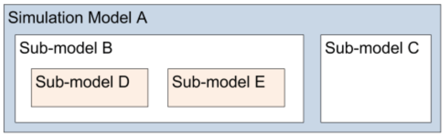


Figure 3. Simulation model hierarchy

D. Querying Ontology-based simulation models

Simulation models represented as instances of Simulator's ontologies can be queried using different querying languages. We explore two different querying language styles: the RDF querying language, SPARQL [9], and the ontology querying language, SQWRL [10].

Since SPARQL is the W3C recommendation for querying RDFs [9] and OWL can be serialized as an RDF, SPARQL can be used to query ontology-based simulation models. However, as SPARQL is not an ontology querying language, it ignores inferences imposing limitations on querying, as will be shown in the case study. This drawback can be overcome by using a genuine ontology querying language such as SQWRL (Semantic Query-Enhanced Web Rule Language), which is a SWRL-based (Semantic Web Rule Language) language for querying ontologies. Accordingly, in the presented scenario, we use both the SPARQL and SQWRL approaches, identifying their advantages and disadvantages in regards to querying ontology-based simulation models.

E. Creating Ontology-based Simulation Models

In the proposed approach, the simulation models represented in the domain simulation engine environment serve as an information source for the representation of models as instances of an ontology. While the approaches proposed by Tofani et al. [3] and Silver et al. [5] also describe simulation models as instances of ontologies, these approaches perform modeling directly in the ontology, which is then mapped or transformed to a different representation. In contrast, our approach uses existing domain simulation models as an origin for the creation of its own ontology-based representation. The advantages of this approach include:

- The use of existing, domain-specific models.
- The ability of domain experts to create new models in the simulation engine to which he/she is accustomed rather than creating models directly as an ontology.
- The use of proven domain simulators for simulation execution.
- There is no need for manual mapping between simulators and ontology models.

Fig. 4 portrays our approach for the creation of an ontology-based simulation model representation. Specifically, the Transformation Engine inputs consist of the Simulator's Ontology and the simulator's model in the domain simulation engine representation. The Simulator's Ontology is simulator-specific, while the simulator's models are model-specific, as each model is stored in a separate file.

The Simulator's Ontology is read by the Ontology Reader, which is independent of the simulation engine. While ontologies are stimulator-specific, they are always represented using the same ontology language, thus allowing for a simulator-independent reader. In particular, the Ontology Reader is responsible for acquiring information about simulator's classes and their properties. Classes are relevant concepts from a specific domain, such as *channel* and *cell*; they can be perceived as sets of individuals, which include actual objects from the domain, such as a set of all individual

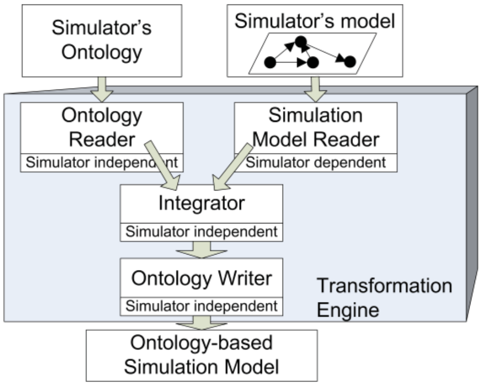


Figure 4. Ontology-based model creation from simulator's model

channels in a distribution network. Although the Simulators' Ontology does not contain individuals, they will be extracted by the Transformation Engine. Moreover, the Ontology Reader is responsible for reading properties, including data properties and object properties. Data properties connect individuals with literals; an example of a data property is th e capacity of a specific storage cell. Conversely, object properties connect pairs of individuals such as the *hasInput* property, which links the cell with the channel in the statement, 'Cell *x hasInput* Channel *y*'.

The second transformation source, the simulator's model, is read by the Simulation Model Reader. Since the format of the simulator's model depends on the specific simulator, a separate Simulation Model Reader has to be created for each simulator whose model requires transformation to an ontology-based representation. However, once a Simulation Model Reader is created for a specific simulator, the reader can be used to transform any model represented in that format. The architecture of the Simulation Model Reader depends on the model being read. For instance, the reader can utilize the simulator's API interface, directly read the model file or use external model readers.

The Integrator uses the data received from the Ontology Reader and the Simulation Model Reader for creating the ontology-based model representation. Specifically, the Integrator receives information about the simulator's classes from the Ontology Reader. For each class, the Integrator obtains knowledge about its individuals from the Simulation Model Reader. When an Integrator identifies individuals, it also obtains values for their data properties. After acquiring information about all in dividuals of all class es and their data properties, the Integrator proceeds to determine the object properties. Since object properties connect individuals of the same or different classes, all individuals must be determined before the object properties are defined.

Subsequently, the Integrator sends information about classes, individuals, data properties and object properties to the Ontology Writer, which writes an ontology-based simulation model representation. Rather than recreating classes, the output ontology imports the Simulators' Ontology to acquire domain-relevant concepts and properties. Then, individuals and property values are created from the information received via the Integrator, and the output is r ecorded in an ontology language such as OWL. Thus, the Ontology Writer is simulator-independent, as its purpose is to write ontologies from the Integrator's information.

Consequently, the Simulation Model reader is the only Transformation Engine component that is simulator-dependent. However, this reader can be replaced with a different simulator's reader in order to represent that specific simulator's model in an ontology-based representation.

IV. CASE STUDY

This work is part of the CANARIE-sponsored Disaster Response Network Enabled Platform (DR-NEP) project [11]. The project aims to improve the capability to prepare for and respond to large disasters. In particular, disaster modeling and simulation play a major role in the project, with a special focus on critical infrastructure (CI) interdependency simulation. Therefore, the proposed ontology-based representation of simulation models is evaluated using I2Sim [12] infrastructure interdependencies simulator.

The proposed approach is generic, as it is independent of any simulation engine; however, its implementation requires the creation of engine-specific Simulator Ontologies and the Simulation Model Reader. Therefore, the I2Sim ontology is created; the ontology design and the mapping to the upper ontology are presented in [13]. Since I2Sim is based on MATLAB's Simulink engine, the Transformation Engine inputs include the I2Sim ontology and the I2SIm model, which is stored in the Simulink style .mdl file. The Transformation Engine was implemented using the following technologies:

- The Ontology Reader and the Ontology Writer are implemented using Protégé OWL API [14] and Java 1.6.
- OWL is used for the representation of the ontology-based simulation models.
- The Integrator is implemented using Java 1.6.
- The I2Sim Simulation Model Reader uses the Simulink Java library from Technische Universität München [15].

A. *Ontology-based Representation of Simulation Models*

To explore ontology-based simulation models we used the I2Sim model developed as part of the DR-NEP project for the investigation of infrastructure interdependencies.

MATLAB's Simulink engine [8], upon which I2Sim is built, is an environment for multi-domain simulation and for dynamic and embedded systems. Simulink provides block libraries which can be customized to conform to a specific simulation domain. Complex models are managed by dividing models into hierarchies of sub-models. Accordingly, I2Sim builds upon Simulink by customizing Simulink blocks and providing entities specific for infrastructure interdependency simulation.

The I2Sim model that we used in this case study consists of several hierarchy levels. However, before transforming it to the ontology-based model we were not aware of the number of layers, the number of blocks or types of blocks used.

First, the I2Sim ontology was created [13] containing only I2Sim blocks as i llustrated in Fig. 5(a). Subsequently, the

I2Sim simulation model was transformed to an ontology-based representation, which is depicted from the perspective of the Protégé ontology editor in Fig. 5(b). Specifically, the left part of the screen shows I2Sim classes, such as *i2sim_source*, and *production_cell*. As the *channel* class is selected, the middle part of the screen displays all of the individual channels from the I2Sim model. On the right side are displayed the object and data properties for the selected channel, *from_feed_water_pump_1_to_steam_house_3*. As the channels are not named in I2Sim, we have chosen to use *from_sourceNode_port_to_targetNode_port* as a channel naming pattern. The object properties *hasStartNode* and *hasEndNode* indicate that the selected channel starts from the *feed_water_pump* and ends at the *steam_house*. In the I2Sim ontology, *hasStartNode* and *hasEndNode* are asserted properties, as the I2Sim model specifies the channel start and end. The inverse properties, *hasInput* and *hasOutput*, are inferred, allowing for ontology querying in both directions.

Another significant I2Sim modeling concept for establishing network topology is the port concept. I2Sim entities such as the production cells can have several input and output ports. Each channel connects to a specific input and output port as identified by the port number. Therefore, in the ontology-based representation each channel has *hasInPort* and *hasOutPort* data properties, as illustrated in Fig. 5(b). The channel selected in the figure connects to the first *feed_water_pump* port and the third *steam_house* port.

Observing the classes from I2Sim ontology, Fig. 5(a), and from the ontology-based simulation model, Fig. 5(b), it can be noticed that the ontology-based model contains additional entries such as *minmax*, *product* and *fcn*. Initially, we expected the I2Sim model to have only I2Sim blocks. However, when the model was transformed to its ontology-based representation, many entities belonged to the *other* class.

Subsequently, we analyzed those entities and identified that they are Simulink blocks. Since I2Sim is founded on Simulink by customizing and extending Simulink blocks, it allows for the use of Simulink blocks in conjunction with the I2Sim blocks. Accordingly, our case study of the I2Sim model actually contained I2Sim and Simulink blocks. Therefore, we created the *non_i2sim* class and in the transformation process we allowed for the creation of *non_i2sim* sub-classes representing Simulink blocks categories used in the observed I2Sim model.

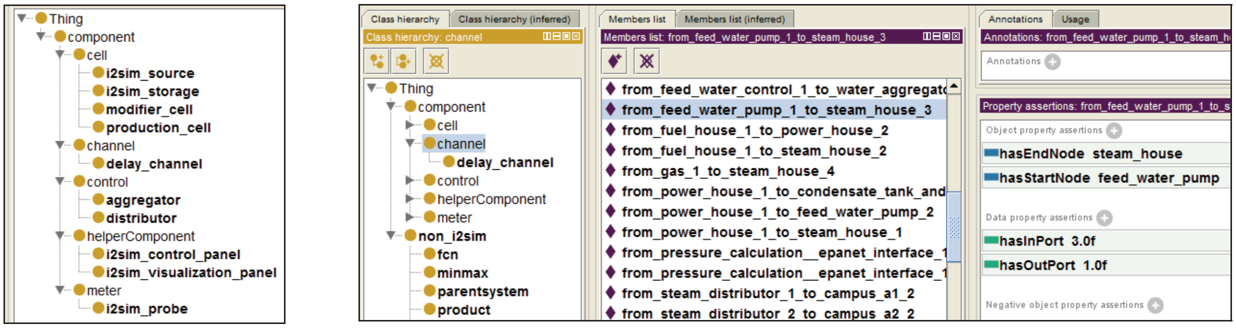
The ontology-based representation of the I2Sim model hierarchy is portrayed in Fig. 6. On the left part of the screen the *parentsystem* class is selected, while the segment to the right shows all entities that act as a container for the other elements. The selected entity *water_pump_1* is a child of the *water_pump_control* as indicated with the *parentSystem* object property. To simplify the illustration of hierarchies we have chosen to include the hierarchy chain in the entity name separating the hierarchy levels with a dash as in *steam-house-water_pump_control-water_pump_1*. The maximum number of hierarchy levels in the observed I2Sim model was three.

B. Querying ontology-based representation

A simulator's model that is represented as ontology instances can be queried using querying languages such as SPARQL or SQWRL. The use of these languages enables the extraction of specific information from the model. Since the number of entities in a system is one of the complexity indicators, after representing I2Sim models as instances of an ontology, we first wanted to obtain the number of instances in the ontology-based representation. However, the standard SPARQL included with Protégé does not support aggregate functions such as *count* and *avg*. Nevertheless, different implementations that support aggregates exist, such as ARQ [16]. Instead, we used SQWRL to identify the number of I2Sim and Simulink entities. I2Sim components are instances of the *component* class and its subclasses, while Simulink entities are instances of *non-i2sim* class and its subclasses:

```
simupper:component(?c)→sqwrl:count(?c)
non_i2sim(?c)→sqwrl:count(?c)
```

Those two queries identified that the observed I2Sim model consists of 449 I2Sim components and 230 Simulink

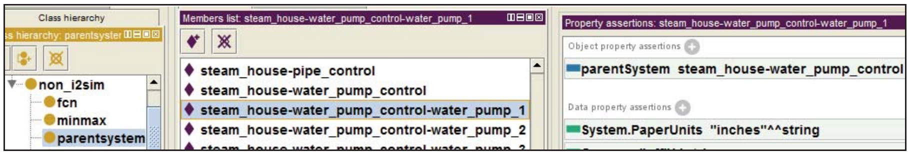

Thing
component
cell
i2sim_source
i2sim_storage
modifier_cell
production_cell
channel
delay_channel
control
aggregator
distributor
helperComponent
i2sim_control_panel
i2sim_visualization_panel
meter
i2sim_probe
Class hierarchy
Class hierarchy (inferred)
Class hierarchy: channel
Thing
component
cell
channel
delay_channel
control
helperComponent
meter
non_i2sim
fcn
minmax
parentsystem
product
Members list
Members list (inferred)
Members list: from_feed_water_pump_1_to_steam_house_3
from_feed_water_control_1_to_water_aggregat
from_feed_water_pump_1_to_steam_house_3
from_fuel_house_1_to_power_house_2
from_fuel_house_1_to_steam_house_2
from_gas_1_to_steam_house_4
from_power_house_1_to_condensate_tank_and
from_power_house_1_to_feed_water_pump_2
from_power_house_1_to_steam_house_1
from_pressure_calculation__epanet_interface_1
from_pressure_calculation__epanet_interface_1
from_steam_distributor_1_to_campus_a1_2
from_steam_distributor_2_to_campus_a2_2
Annotations
Usage
Annotations: from_feed_water_pump_1_to_steam_h
Annotations
Property assertions: from_feed_water_pump_1_to_s
Object property assertions
Data property assertions
Negative object property assertions

hasEndNode steam_house
hasStartNode feed_water_pump
hasInPort 3.0f
hasOutPort 1.0f

a) I2Sim Ontology

b) Ontology-based representation of the I2Sim model

Figure 5. Ontologies from the Protégé editor


Class hierarchy
s hierarchy: parentsyster
non_i2sim
fcn
minmax
parentsystem
Members list: steam_house-water_pump_control-water_pump_1
steam_house-pipe_control
steam_house-water_pump_control
steam_house-water_pump_control-water_pump_1
steam_house-water_pump_control-water_pump_2
Property assertions: steam_house-water_pump_control-water_pump_1
Object property assertions
Data property assertions

parentSystem steam_house-water_pump_control
System.PaperUnits "inches"^^string

Figure 6. The hierarchy levels of the I2Sim model

components. Subsequently, since I2Sim model extensively uses hierarchical modeling, we identified each sub-system together with the number of elements in each sub-system using the following query:

```
owl:Thing(?c) ∧
i2sim:parentSystem(?c,?subsystem) ∧
sqwrl:makeSet(?s, ?c) ∧
sqwrl:groupBy(?s, ?subsystem) ∧
sqwrl:size(?count, ?s) →
sqwrl:select(?subsystem, ?count) ∧
sqwrl:orderByDescending(?count)
```

The first few rows from this query results with our ontology-based representation of the I2Sim model are displayed in Fig. 7. Since the sorting is performed on the number of entities, the first few rows indicate sub-models with the highest number of components.

To illustrate the difference of querying ontology-based simulation models using SPARQL and SQWRL we used a simple example of finding channels in the *steam_house-boiler_1* sub-model. In SPARQL this query is written as:

```
SELECT *
WHERE { ?c rdf:type :i2sim.channel.
?c i2sim:parentSystem :steam_house-boiler_1}
```

While in SQWRL the same query is expressed as:

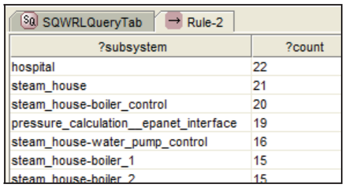
```
simupper:channel(?c) ∧
i2sim:parentSystem(?c, steam_house-boiler_1) →
sqwrl:select(?c)
```

The SPARQL query did not find any entities, while the SQWRL found the *steam_house-boiler_1-water_tank_tube*. This entity belongs to the *delay_channel* class. However, although the *delay_channel* is defined as *subClassOf channel*, SPARQL could not infer that *delay_channel* is also a *channel*, and thus, SPARQL did not identity this entity. Since SQWRL is an ontology querying language, it inferred that *delay_channel* is also a *channel* and therefore properly identified the entity.

V. Conclusions

Application-oriented simulation packages vary greatly in modeling approaches, technologies and vocabularies. However, this heterogeneity imposes several challenges, including the difficulty of comparison among simulation models of one or more simulation engines and the inability to query simulation models. This work proposes to solve those issues using ontology-based representations of simulation models where domain simulation models are represented as instances of Simulators' Ontologies. Existing domain simulation models in proprietary file formats are a foundation for this ontology-based representation.

The main benefits of using ontology-based representation for simulation models include: models from different simulation platforms are represented in a common manner, the models can be queried using ontology querying languages and inferences can be performed using ontology reasoners.



?subsystem	?count
hospital	22
steam_house	21
steam_house-boiler_control	20
pressure_calculation__epanet_interface	19
steam_house-water_pump_control	16
steam_house-boiler_1	15
steam_house-boiler_2	15

Figure 7. SQWRL query output from Protégé

While we have focused on representing the simulation model as contained in a model file, this model is only a static part of the simulation. Accordingly, we plan to explore the possibility of using ontologies for representing the dynamic part of the running simulation. Furthermore, we will investigate the use of proposed ontology-based simulation models to facilitate reuse, integration and information sharing among simulators of different domains.

Acknowledgment

Support for this work has been provided by Canada's Advanced Research and Innovation Network (CANARIE) and the Natural Sciences and Engineering Research Council (NSERC) of Canada.

References

[1] E. Abu-Taieh and A. El Sheikh. , "Commercial Simulation Packages: A Comparative Study", *International Journal of Simulation*, vol. 8, no. 2, pp. 66-76, 2007.

[2] L. Lacy and W. Gerber, "Potential Modeling and Simulation Applications of the Web Ontology Language - OWL", *Proc. Winter Simulation Conference*, vol. 1, pp. 265-270, 2004.

[3] A. Tofani, E. Castorinia, P. Palazzaria, A. Usovb, C. Beyelb, E. Romeb and P. Servilloc, "Using Ontologies for the Federated Simulation of Critical Infrastructures", *Proc. International Conference on Computational Science*, vol. 1, no. 1, pp. 2301-2309, 2010.

[4] J.A. Miller, G.T. Baramidze, A.P. Sheth and P.A. Fishwick, "Investigating Ontologies for Simulation Modeling", *Proc. 37th Annual Simulation Symposium*, pp. 55-63, 2004.

[5] G.A. Silver, L.W. Lacy and J.A. Miller, "Ontology Based Representations of Simulation Models Following the Process Interaction World View", *Proc. Winter Simulation Conference*, pp. 1168-1176, 2006.

[6] J.A. Miller and G. Baramidze, "Simulation and the Semantic Web", *Proc. Winter Simulation Conference*, pp. 2371-2377, 2005.

[7] P. Benjamin and K. Akella. , "Towards Ontology-Driven Interoperability for Simulation-Based Applications", *Proc. Winter Simulation Conference*, pp. 1375-1386, 2009.

[8] "Simulink - Simulation and Model-Based Design", http://www.mathworks.com/products/simulink/, 2011.

[9] E. Prud'hommeaux and A. Seaborne. , "SPARQL Query Language for RDF", http://www.w3.org/TR/rdf-sparql-query/, 2008.

[10] M.J. O'Connor and A. Das. , "SQWRL: A Query Language for OWL", *OWL Experiences and Directions, 6th International Workshop*, 2009.

[11] "DR-NEP (Disaster Response Network Enabled Platform) project", http://drnep.ece.ubc.ca/index.html, 2011.

[12] H.A. Rahman, M. Armstrong, D. Mao and J.R. Marti, "I2Sim: A Matrix-Partition Based Framework for Critical Infrastructure Interdependencies Simulation", *Proc. Electric Power Conference*, pp. 1-8, 2008.

[13] K. Grolinger, M.A.M. Capretz, A. Shypanski and G.S. Gill, "Federated Critical Infrastructure Simulators: Towards Ontologies for Support of Collaboration", *Proc. IEEE Canadian Conference on Electrical and Computer Engineering, Workshop on Connecting Engineering Applications and Disaster Management*, 2011.

[14] "Protégé OWL API", http://protege.stanford.edu/plugins/owl/api/, 2011.

[15] "Simulink Library, Technische Universität München", http://conqat.in.tum.de/index.php/Simulink_Library, 2011.

[16] "ARQ - A SPARQL Processor for Jena", http://incubator.apache.org/jena/documentation/query/.

An ontology-based approach for storing XML data into relational databases

Francisco Tiago Machado de Avelar, Deise de Brum Saccol, Eduardo Kessler Piveta
Programa de Pós-Graduação em Informática, Universidade Federal de Santa Maria - Santa Maria, RS, Brazil
ftiago.avelar@gmail.com, {deise, piveta}@inf.ufsm.br

Abstract

One of the efficient ways of managing information is by using a relational database (RDB). Several applications use XML (eXtensible Markup Language) as the standard format for data storing and processing. However, XML documents related to a certain domain may present different structures. Such structural heterogeneity makes harder to map documents to a specific database (DB) schema. To address this issue, our paper assumes an existing ontology that describes the XML documents. Then the ontology is mapped to a relational schema and the XML files are stored into the DB. Considering this scenario, this paper describes a mechanism that inserts XML data into the RDB. Specifically, this paper presents: (1) the definition of mapping rules to transform XML data to the relational format, (2) the definition of algorithms for solving semantic and structural conflicts, and (3) the mechanism for generating the SQL scripts to insert the original XML data into the DB.

1 Introduction

XML is a widely-used format for sharing information between programs, people, and computers. Many applications require efficient XML storage, which can be achieved by using a RDB. One possible way is to map the XML structure to a collection of relations, and then proceed with the insertion of the XML data as tuples into the DB.

For mapping the XML structure to a relational schema, the ideal scenario occurs when the XML files share the same schema; thus, such schema is translated to a set of tables using some of the existing mapping approaches [8]. However, XML files may have different structures, making the mapping process to a unique relational schema harder.

To deal with this heterogeneity issue, we group XML documents into similar knowledge domains. This grouping can be done by using ontologies, whose purpose is to establish a common and shared understanding about a domain between people and computer systems. In such scenario, searching and managing tasks become simpler.

There are some approaches for dealing with global schema representation for describing heterogeneous XML structures. One of these approaches is the *X2Rel (XML-to-**Rel**ational)* framework [6]. In such framework, the ontology is created from a schema integration process (*OntoGen* - ***Onto**logy **Gen**erator* component) [7]. Then the ontology is mapped to a relational schema by using a set of mapping rules (*OntoRel* - ***Onto**logy-to-**Rel**ational* component) [6]. After generating the relational schema, the original XML data can be inserted into the DB.

In this paper, we extend the *X2Rel* framework by adding the *XMap (XML **Map**per)* component, a module that provides mechanisms for inserting the original XML data into the RDB. We also propose a mechanism for solving structural and semantical conflicts found in those documents, such as nomenclature and data types conflicts. Therefore, the main contributions of this paper are: (1) the categorization of structural and semantic conflicts between XML documents; (2) the definition of algorithms that solve the conflicts found in the XML documents; and (3) the definition of mechanisms that map and insert the XML data into the existing relational schema.

This paper is organized as follows. Section 2 presents related work. Section 3 presents the background work, including the *X2Rel* components. Section 4 presents the XMap component, responsible for mapping and inserting the XML data into the RDB. Section 5 presents a brief categorization of conflicts and some of the proposed algorithms. Some experimental issues are described in Section 6. Finally, conclusions are presented in Section 7.

2 Related Work

There are some approaches [3, 11] that deal with XML mapping to OWL. The transformation from OWL to SQL is addressed in [1] and [14]. Direct translation from XML to SQL [12] uses mapping without the adoption of a OWL model. Consequently, conflict resolution only have solutions for a certain set of files.

The transformation from XML to SQL format requires the definition of a mapping schema and it may require a specialist intervention. The work at [10] classifies the conflicts during the integration of XML schemas and proposes a resolution mechanism using XQuery.

The approach in [9] performs a division of mappings. This technique decomposes the document into sub-trees. Joint operations are represented by an undirected graph with acyclic handling algorithms to solve conflicts. An XML framework proposed in [13] defines data integration based on the overall schema, a set of XML data and a set of mappings. They define an identification function that aims at globally identifying nodes coming from different sources.

To solve the problem of heterogeneity, this paper presents a ontology-based approach, which can be used to improve traditional techniques of XML mapping and storage into RDB. The purpose of using ontologies is to explicit the resource content regardless of how the information is structurally stored. This approach adds semantics to the XML files while reduces unneeded structural information to map the XML data to the relational format.

3 Background

To avoid submitting different XQuery queries, we propose to store XML data in a RDB. By mapping the XML structure to the relational model, only one SQL query is posed into the DB. To provide this solution, we need to define a set of transformation rules that map the XML structure to a collection of tables and columns.

However, XML documents related to the same application domain may present different structures, making the mapping process more difficult. To overcome this issue, the *X2Rel* framework provides the following functionalities, as described in Figure 1:

- Integrate the XML files into a global schema, described by a OWL ontology. Provided by the *OntoGen* component, this module receives a set of XML files and produces the ontology (integrated schema) [7];
- Translate the OWL ontology to a relational schema. Provided by the *OntoRel* component, this module receives the ontology and produces the relational schema (a SQL script with the *create table* statements) [6];
- Map and insert the original XML files into the RDB. Provided by the *XMap* component, this module receives the XML files, the ontology and the relational schema and produces the SQL script with a set of *insert* statements;
- And finally map the original XML queries into equivalent SQL statements. Provided by the *QMap* component, this module receives a XQuery statement and produces a corresponding SQL statement.

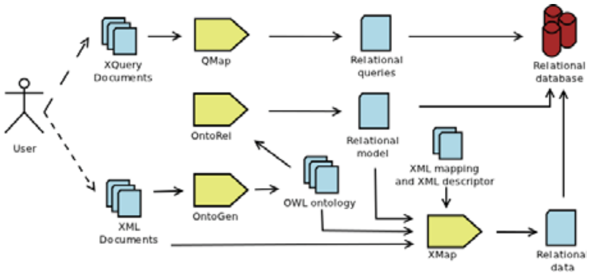


Figure 1. X2Rel framework architecture [6].

The *OntoGen* and *OntoRel* components are finished and published works. The *QMap* module is an ongoing project. The focus of this paper is to present the *XMap* component, as described in the next section.

4 The XMap Component

The *XMap* component uses as input the original XML files, the global ontology generated by *OntoGen* and the relational schema created by *OntoRel*. In order to specify the concept equivalences between the XML documents, the ontology and the relational schema, our approch is based on a mapping document, as later detailed in Section 4.2. The insertion of XML data into the RDB is structured in five ascending levels, as described in Figure 2.

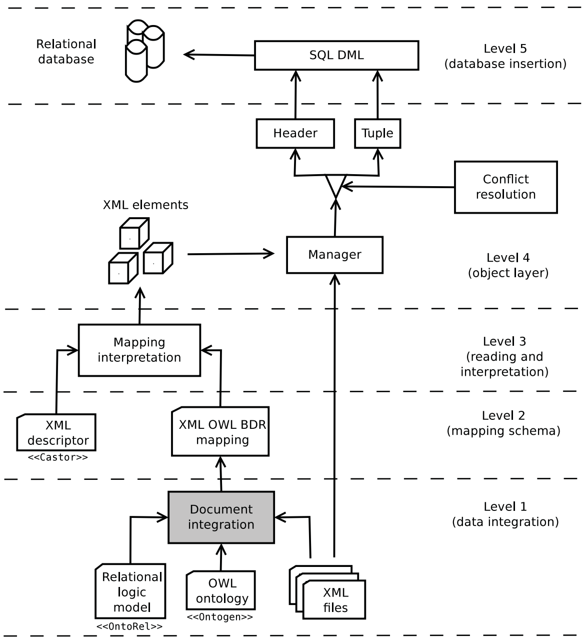


Figure 2. The XMap Architecture

The first level associates the input data. The integrated representation of such documents is responsible for specifying the equivalences between concepts, associating them in those three models. For example: a) the XML element `Writer` is equivalent to the OWL concept `Author` that is represented by the table `Author`; b) the XML element `name` is equivalent to the OWL concept `fullName` that is represented by the column `fullName`;

The output of the first level is the mapping file (XML OWL RDB (Relational Database) Mapping). The XML descriptor is an auxiliary file created by the programmer using the Castor library. Section 6 describes further details.

The third level is in charge of interpreting the mapping file with the XML descriptor. As a result, objects are created in the level 4. The manager is responsible for interpreting the XML files according to the mapping objects. If a conflict occurs, conflict resolution becomes necessary to resolve the data inconsistency. Section 5 addresses specifically about the conflicts that can occur between XML files.

With the mapping objects in memory, the manager is responsible for creating the header objects for the inclusion into the DB along with the respective tuples, as shown in Figure 3. Finally, level 5 generates the DML [1] file to insert the content of XML instances into tables and columns.

Header

```
INSERT INTO <table> (<column 1>, <column 2>, <column 3>, ..., <column n>) VALUES
```

Tuple(s)

```
(<value_1 column 1>, <value_1 column 2>, <value_1 column 3>, ..., <value_1 column n>),
(<value_2 column 1>, <value_2 column 2>, <value_2 column 3>, ..., <value_2 column n>),
(<value_3 column 1>, <value_3 column 2>, <value_3 column 3>, ..., <value_3 column n>),
...
(<value_m column 1>, <value_m column 2>, <value_m column 3>, ..., <value_m column n>);
```

Figure 3. Header layout

Next we present the input artifacts produced by *OntoGen* (ontology) and *OntoRel* (relational schema), as well as three XML documents. These artifacts are important for understanding the *XMap* approach and the remaining paper.

4.1 Input Artifacts

Each XML file can have a particular structure. Figure 4 illustrates three XML files that contains data about authors, publications and institutions. Although these files belong to the same application domain (published papers in scientific events), they have some structural differences, such as:

- On documents “A” e “C” (line 10), the author concept is `<author>`; on document “B” (line 10), the same concept is `<writer>`;
- On document “A” (line 10), the concept author is lexical [2]; on documents “B” e “C” (line 10), this concept is non-lexical [3];

Doc_A.xml

```
01 <?xml version="1.0" encoding="utf-8" ?>
02 <conferences>
03   <conference>
04     <title>XX SBBD</title>
05     <day>01/01/2005</day>
06     <papers>
07       <paper>
08         <title>Temporal Versioned Constraint Language</title>
09           <authors>
10             <author>Robson Mendes</author>   ...
11           </authors>
12           <institution>
13             <name>Instituto de Informática</name>
14             <city>Porto Alegre</city>
15             <state>RS</state>
16           </institution>
17       </paper> ...
18     </papers>
19   </conference> ...
20 </conferences>
```

Doc_B.xml

```
01 <?xml version="1.0" encoding="utf-8" ?>
02 <conferences>
03   <conference>
04     <title>XX Simposio Brasileiro de Banco de Dados</title>
05     <city>Belo Horizonte</city>
06     <state>MG</state>
07     <papers>
08       <paper>
09         <title>TVCL - Temporal Versioned Constraint Language</title>
10         <writer code="1">
11           <name>Edson Marques</name>
12           <e_mail>emarques@inf.ufsm.br</e_mail>
13         </writer>
14         <institution>
15           <name>UFSM</name>
16           <department>Instituto de Informática</department>
17         </institution>
18       </paper> ...
19   </conference> ...
20 </conferences>
```

Doc_C.xml

```
01 <?xml version="1.0" encoding="utf-8" ?>
02 <papers>
03   <paper>
04     <title>TVCL</title>
05     <conference>
06       <title>Simposio Brasileiro de BD</title>
07       <day>01/07/2011</day>
08     </conference>
09     <authors>
10       <author>
11         <code>2</code>
12         <name>Carlos Souza</name>
13         <e_mail>csouza@inf.ufsm.br</e_mail>
14         <age>28</age>
15       </author> ...
16     </authors>
17   </paper> ...
18 </papers>
```

Figure 4. Sample XML documents

Querying data in such heterogenous structure would require individual extraction processes that could represent a bottleneck to the system. To store the content of these files into a RDB, our work uses a ontology as a global schema to handle the structural distinctions between the documents. Figure 5 shows the resulting OWL ontology, considering the sample XML files as input to the *OntoGen* component. Details of the ontology generation are found in [7].

Then the ontology is mapped to a relational schema by the *OntoRel* component using some mapping rules:

[1] DML - Data Manipulation Language.

[2] Atomic elements directly represented in computer, such as a string.

[3] Complex element, such as an author composed by name and address.

Institution (cod_institution, city, state, name, department);

Paper (cod_paper, title, cod_institution), `cod_institution` references `Institution;`

Writer (cod_writer, name, code, e_mail, age);

Conference (cod_conference, day, title, city, state);

PaperWriter (cod_paper, cod_writer), `cod_paper` references `Paper` and `cod_writer` references `Writer;`

ConferencePaper (cod_conference, cod_paper), `cod_conference` references `Conference` and `cod_paper` references `Paper;`

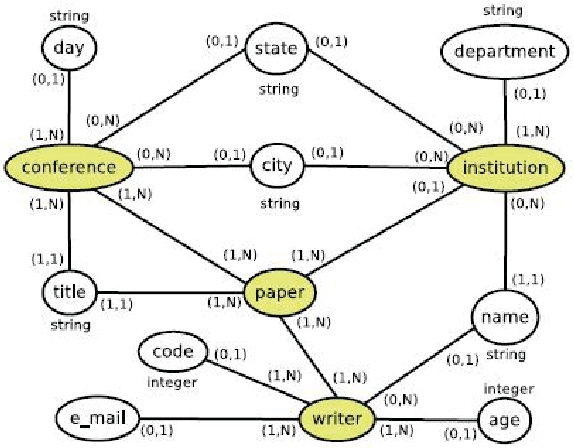


Figure 5. Global OWL Ontology

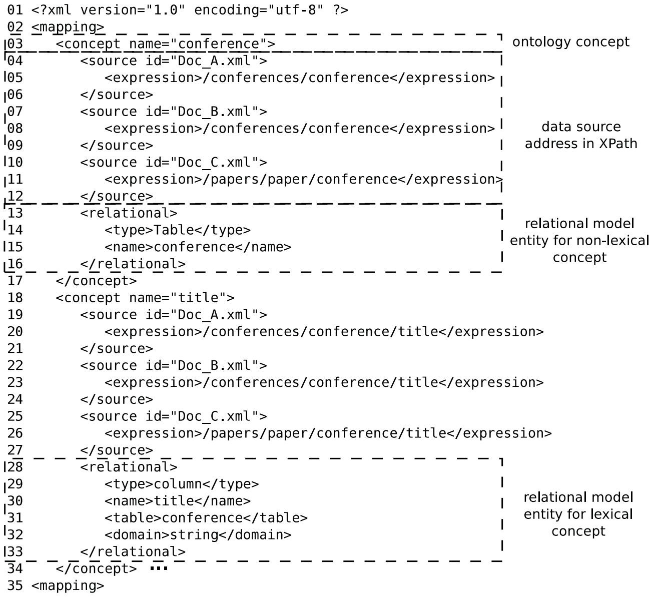
Basically the non-lexical concepts are mapped to tables, as well as the N:N relationships. The lexical concepts are mapped to columns in the corresponding tables. The columns whose cardinality is 1 become `not null` in the DB. Details of this mapping are described in [6].

4.2 Mapping Document

In order to describe and track the equivalences between concepts from the XML files, the ontology and the relational schema, we have defined a structured artifact named mapping document, as shown in Figure 6.

To explain the involved transformations for storing the input XML data into the relational schema, the mapping process represents the ontology concepts with the element `<concept>` (line 3). According to the existing concept in the global ontology, the XML data sources are identified by the `id` attribute of the element `<source>` (line 4). The XML content is pointed by the XPath expression in the element `<expression>` (line 5). If an XML source content has not refered to a certain ontology concept, the `<expression>` element is empty.

The element `<relational>` (line 13) indicates where the data will be stored. For non-lexical concepts, the storage is done into a table, as shown in `<type>` element (line 14); the table name is the `<name>` element (line 15).



```
01 <?xml version="1.0" encoding="utf-8" ?>
02 <mapping>
03    <concept name="conference">
04       <source id="Doc_A.xml">
05          <expression>/conferences/conference</expression>
06       </source>
07       <source id="Doc_B.xml">
08          <expression>/conferences/conference</expression>
09       </source>
10       <source id="Doc_C.xml">
11          <expression>/papers/paper/conference</expression>
12       </source>
13       <relational>
14          <type>Table</type>
15          <name>conference</name>
16       </relational>
17    </concept>
18    <concept name="title">
19       <source id="Doc_A.xml">
20          <expression>/conferences/conference/title</expression>
21       </source>
22       <source id="Doc_B.xml">
23          <expression>/conferences/conference/title</expression>
24       </source>
25       <source id="Doc_C.xml">
26          <expression>/papers/paper/conference/title</expression>
27       </source>
28       <relational>
29          <type>column</type>
30          <name>title</name>
31          <table>conference</table>
32          <domain>string</domain>
33       </relational>
34    </concept> ...
35 <mapping>
```

ontology concept
data source address in XPath
relational model entity for non-lexical concept
relational model entity for lexical concept

Figure 6. Mapping file

For lexical concepts, the element `<relational>` has two internal elements, indicating that the information is stored into a column. From line 28 on, it is described that the concept `title` specified on line 18 is stored as a column (line 29), whose name is `title` (line 30) in `conference` table (line 31). In this case, it is a lexical concept represented by a string (line 32).

We used the mapping file presented in Figure 6 for generating the insert statements. The mapping document is an essential mechanism that allows specifying the equivalences between the XML data, the ontology and the relational schema. For better understanding, only `conference` e `title` concepts were demonstrated. The mapping document generation is implemented by a semi-automatic tool named *CMap (**C**oncept **Map**per).*

5 Conflict Classification and Resolution

To store original XML data into the RDB, several conflicts have to be detected and solved. Because of space issues, we describe in details only the naming conflicts. Other conflicts are just mentioned in the end of this section. The original work (master thesis [5]) describes the full set of conflicts, as well as all the proposed algorithms, the implementation and the tests that deal with those conflicts.

The naming conflicts refers to naming inconsistencies in XML documents. There are two types of naming conflicts:

- Homonymy conflicts: occur when the same name is used for different concepts. For example, <name> of <writer> (line 11 of document “B”) and <name> of <institution> in document “A” (line 13).
- Synonymy conflicts: occur when the same concept is described by distinct names. For example, <writer> (line 10 of document “B”) and <author> in document “C” (line 10).

Each XML file is identified by the id attribute (element source), as specified in the mapping document (Figure 6). The element contains the XPath <expression> that points to the content located on the specific file. Exemplifying the homonym conflict, the element name has different parents in documents “B” (line 11) and “A” (line 13), meaning the author name and the institution name. Thus, a homonym conflict is detected. The question is about where to store such information: either writer or institution table.

The solution of this conflict is based on the mapping document and it considers the grouping algorithm of non-lexical concepts (algorithm 1), as described.

Algorithm 1 Grouping non-lexical concepts.

```
nc $\leftarrow$ number of $<$ concept $>$ in $<$ mapping $>$
for $i \leftarrow 1$ to nc do
    concept $\leftarrow$ i-th$<$ concept $>$ of $<$ mapping $>$
    if non-lexical concept then
        relational $\leftarrow$ $<$ relational $>$ of concept
        label $\leftarrow$ $<$ name $>$ of relational
        add(label, vector)
    end if
end for
```

Non-lexical concepts are placed in a vector according to Algorithm 1. As a result, this vector will contain the non-lexical concepts that correspond to tables in the RDB. Reading all the elements in this vector, the Algorithm 3 will create the insertion structure (header and container) for the DB. Both for-loops in Algorithm 3 mean that for each non-lexical concept (outer loop), the associated lexical concepts will be fetched (inner loop) in the mapping document. Thus, it is possible to access the content structure and the XML data (lines 1-3 of algorithm 2) through the mapping document in the form of objects.

The loop traverses the XML sources searching for the file identifier and the Xpath path (lines 5-7 of algorithm 2). If an expression is non-empty (line 8 of algorithm 2), the XML content pointed by the XPath expression is read from the XML file and added to the DynamicVector (lines 9 and 10 of algorithm 2). If the expression is empty, the content NULL is added to the vector. Since the insertion has the header, the table name and the columns are already grouped (as shown in algorithms 1 and 2). Thus, the tuple composition (line 15 of algorithm 2) produces the insert statement associating the name of the table, the columns and the content.

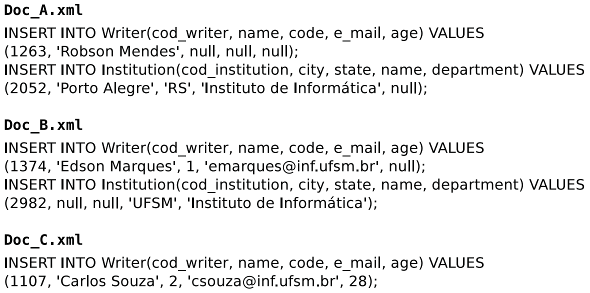
The inclusion into the DB occurs directly, as described in Figure 7.

Algorithm 2 Naming conflict resolution.

```
1: container $\leftarrow$ getContainer(insertion)
2: content $\leftarrow$ getContent(container)
3: sourceVector $\leftarrow$ getVector(content)
4: for $i \leftarrow 0$ to sourceVector.size $- 1$ do
5:     source $\leftarrow$ i-th element of sourceVector
6:     file $\leftarrow$ attribute id of source
7:     expression $\leftarrow$ $<$ expression $>$ of source {an XPath path}
8:     if expression not empty then
9:         value $\leftarrow$ read(expression, file)
10:        add(value, valueVector)
11:    else
12:        add(NULL, valueVector)
13:    end if
14: end for
15: composeTuples(valueVector)
```

Algorithm 3 Creating the structure for inserting XML content in a DB

```
for $i \leftarrow 0$ to vector.size $- 1$ do
    label $\leftarrow$ i-th element of vector
    header $\leftarrow$ create(label)
    container $\leftarrow$ create()
    for $j \leftarrow 1$ to nc do
        concept $\leftarrow$ i-th $<$ concept $>$ of $<$ mapping $>$
        if non-lexical concept then
            continue
        end if
        relational $\leftarrow$ $<$ relational $>$ of concept
        table $\leftarrow$ $<$ table $>$ of relational
        field $\leftarrow$ $<$ name $>$ of relational
        if table equals to label then
            add(field, header)
            sourceVector $\leftarrow$ array$<$ source $>$ of $<$ concept $>$
            content $\leftarrow$ create(sourceVector, relational)
            add(content, container)
        end if
    end for
    insertion $\leftarrow$ create(header, container)
    add(insertion, insertionVector)
end for
```



```
Doc_A.xml
INSERT INTO Writer(cod_writer, name, code, e_mail, age) VALUES
(1263, 'Robson Mendes', null, null, null);
INSERT INTO Institution(cod_institution, city, state, name, department) VALUES
(2052, 'Porto Alegre', 'RS', 'Instituto de Informática', null);

Doc_B.xml
INSERT INTO Writer(cod_writer, name, code, e_mail, age) VALUES
(1374, 'Edson Marques', 1, 'emarques@inf.ufsm.br', null);
INSERT INTO Institution(cod_institution, city, state, name, department) VALUES
(2982, null, null, 'UFSM', 'Instituto de Informática');

Doc_C.xml
INSERT INTO Writer(cod_writer, name, code, e_mail, age) VALUES
(1107, 'Carlos Souza', 2, 'csouza@inf.ufsm.br', 28);
```

Figure 7. Homonymy conflict insert

Because of space issues, only the naming conflicts were detailed. However, other types of conflicts that are also handled by *XMap* include:

- Structural conflicts: related to different choices of constructors. There are two types of structural conflicts, named type conflicts and generalization conflicts. Type conflict occur when one concept is represented by different constructors, such as representing the same information as an attribute and as an element. Generalization conflicts occur when an element in a document is a union of other elements, such as a *full name* element and *first name* + *surname* elements.

- Representation conflicts: occurs when an element has its content represented in a different format required for the storage into the DB. More specifically, the information available in XML is not compatible with the corresponding data type in the relational model. For example, a date format represented in the RDB as yyyy-mm-dd and represented in the XML document as dd/mm/yyyy.

6 Experiments

The *XMap* component was implemented as an extension of the *X2Re*l framework. We used Java as the programming language (version 1.6.0_25) and Castor library for mapping file interpretation. We have performed two group of experiments. The XML files of the experiments are found at [2]. This repository also stores the global ontologies created by *OntoGen*, the relational schemas created by *OntoRel* and the mapping documents (input artifacts for the *XMap* tool).

The second group of experiments used the files described in Figure 4, also available at [2]. By running the component, the tables presented in Section 4.1 were populated with the extracted data from the original XML files.

7 Final Remarks

This paper focuses on *XMap* development, a framework component responsible for mapping and storing XML data into a RDB. To perform this storage, a series of conflicts are solved. The solution is based on a mapping document, an useful artifact to express the equivalences between XML, ontology and the corresponding relational concepts (table and column). This document facilitates the mapping during data transformations and also the XML data retrieval through the use of XPath expressions.

So far we assume that the data synchronization process handles automatically once the XML data are mapped and stored into the RDB. This mechanism, known as view maintenance, is deeply explored in [4]. As a future work, we will address complementary issues, such as element order and duplicata elimination.

8 Acknowledgments

This work has been partly supported by SESU/MEC (PET-Programa de Educação Tutorial) and FAPERGS (Auxílio Recém Doutor - Process no. 11/0748-6).

References

[1] I. Astrova, N. Korda, and A. Kalja. Storing OWL Ontologies in SQL Relational Databases. volume 1. IJECES, 2007.

[2] F. Avelar. Repository of experimental data available at http://www.infovisao.com/xml_data/, 2012.

[3] H. Bohring and S. Auer. Mapping XML to OWL Ontologies. In *Leipziger Informatik-Tage, volume 72 of LNI*, pages 147–156. GI, 2005.

[4] V. P. Braganholo, S. B. Davidson, and C. A. Heuser. Pataxó: A framework to allow updates through xml views. *ACM Trans. DB Syst.*, 31(3):839–886, 2006.

[5] F. T. M. de Avelar. XMap: Mapeamento e Armazenamento de Dados XML em Bancos de Dados Relacionais. Master's thesis, Universidade Federal de Santa Maria, Março 2012.

[6] D. de Brum Saccol, T. de Campos Andrade, and E. K. Piveta. Mapping OWL Ontologies to Relational Schemas. *IEEE IRI'11*, 2011.

[7] D. de Brum Saccol, N. Edelweiss, R. Galante, and M. R. Mello. Managing application domains in p2p systems. *IEEE IRI'08*, 2008.

[8] D. Florescu and D. Kossmann. Storing and querying xml data using an rdbms. *IEEE Data Engineering Bulletin*, 22(3):27 – 34, 1999.

[9] Z. Kedad and X. Xue. Mapping Discovery for XML Data Integration. In *OTM Conferences (1)'05*, pages 166–182, 2005.

[10] K.-H. Leeo, M.-H. Kim, K.-C. Lee, B.-S. Kim, and M.-Y. Lee. Conflict Classification and Resolution in Heterogeneous Information Integration based on XML Schema. *IEEE TENCON*, 2002.

[11] P. Lehti and P. Fankhauser. XML data integration with OWL: experiences and challenges. In *IEEE/IPSJ'04*, pages 160 – 167, 2004.

[12] P. Martins and A. H. F. Laender. Mapeamento de Definições XML Schema para SQL: 1999. In *SBBD'05*, pages 100–114, 2005.

[13] A. Poggi and S. Abiteboul. XML Data Integration with Identification. In *DBPL*, pages 106–121, 2005.

[14] E. Vysniauskas and L. Nemuraite. Transforming ontology representation from owl to relational database. *Information Technology and Control*, 35(3A):333–343, 2006.

Automatic Generation of Architectural Models From Goals Models

Monique Soares, João Pimentel, Jaelson Castro, Carla Silva, Cleice Talitha, Gabriela Guedes, Diego Dermeval
Centro de Informática
Universidade Federal de Pernambuco/UFPE
Recife, Brazil
{mcs4, jhcp, jbc, ctlls, ctns, ggs, ddmcm}@cin.ufpe.br

Abstract—**The STREAM (Strategy for Transition Between Requirements and Architectural Models) process presents an approach that allows the generation of early architectural design described in Acme ADL from goal oriented requirements models expressed in *i**. The process includes activities that defines transformation rules to derive such architectural models. In order to minimize the effort to apply the process and decrease the possibility of making mistakes it is vital that some degree of automation is provided. This paper explains in detail the transformation rules proposed and their corresponding formalization in a model transformation language.**

Requirements Engineering, Software Architecture, Transformation Rules, Automation.

I. Introduction

The STREAM (Strategy for Transition between Requirements Models and Architectural Models) is a systematic approach to integrate requirements engineering and architectural design activities, based on model transformations to generate architectural models from requirements models [3]. The source language is *i** (iStar) [11] and the target language is Acme [2].

Our proposal is in line with the current MDD (Model-Driven Development) paradigm, as we support the transformation of models of higher levels of abstraction to more concrete models. The MDD advantages are: greater productivity and, therefore, a lower development time; increased portability; increased interoperability; and lower maintenance costs, due to the improved consistency and maintainability of the code [7].

Currently, the transformation rules defined in the STREAM approach are informally described and are manually applied. Hence, their use is time consuming and effortful, as well as error prone. In order to overcome these shortcomings, we propose to use an imperative transformation language (QVTO [8]) to properly describe them and to provide tool support.

The aim of this paper is to automate the vertical transformation rules proposed in the STREAM. For this, it was necessary to: Define these rules in a proper transformation language; Make these rules compatible with the IStarTool and AcmeStudio tools; and illustrate the use of them rules.

The remainder of this paper is organized as follows. Section 2 describes the background required for a better understanding of this work. Section 3 presents the description and automation of the vertical transformation rules in QVT. Section 4 presents an application example. Section 5 presents the related works and Section 6 discuss the results of this work and future research.

II. Background

This section presents an overview on *i**, Acme and the STREAM process, which are used in our approach.

A. iStar

The *i** language is a goal-oriented modeling language able to represent features of both the organization and of the system to be acquired/developed by/for the organization. Stakeholders and systems are represented as actors, which are active entities able to perform tasks, reach goals and provide resources. In order to achieve their own goals, actors have dependencies with each other [11].

*i** is comprised of two models: a SD (Strategic Dependency) model describes dependency relationships among actors in the organization; a SR (Strategic Rationale) model explains how actors achieve their goals and dependencies.

In a dependency, a *depender* actor relies on a *dependee* actor to achieve something (the *dependum*). A *dependum* can be a goal, which represents the intentional desire of an actor, to be fulfilled; a softgoal to be satisfied, which is a goal with the acceptance criterion not so clear; a resource to be provided; or a task to be performed. Figure 1 presents an excerpt of the *i** metamodel defined for the iStarTool tool [6].

B. Acme

Acme is an ADL (Architectural Description Language) designed to describe the components and connectors (C&C) view of the system architecture [2]. It relies on six main types of entities for architectural representation: *Components*, *Connectors*, *Systems*, *Ports*, *Roles*, and *Representations*. Figure 2 presents the Acme metamodel, based on the AcmeStudio tool [1].

Components represent the primary computational elements and data stores of a system. Connectors characterize interactions among components. Systems denote configurations of components and connectors. Each port identifies a point of interaction between the component and its environment. Roles define the connector's interfaces. Representation supports hierarchical descriptions of architectures. [2].

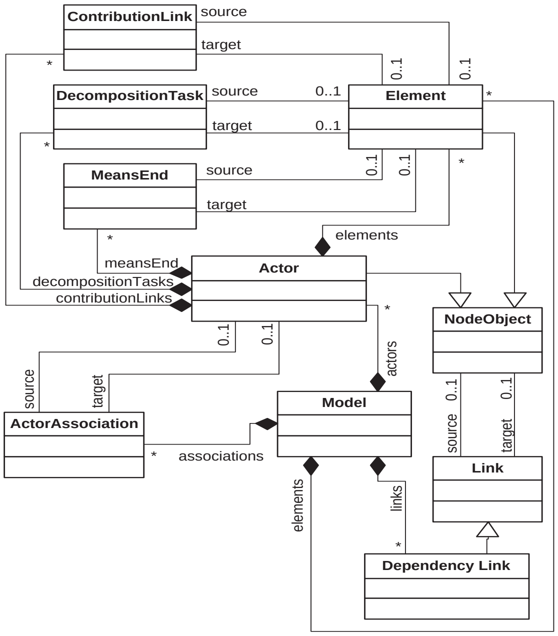


Figure 1. Excerpt of the *i** metamodel

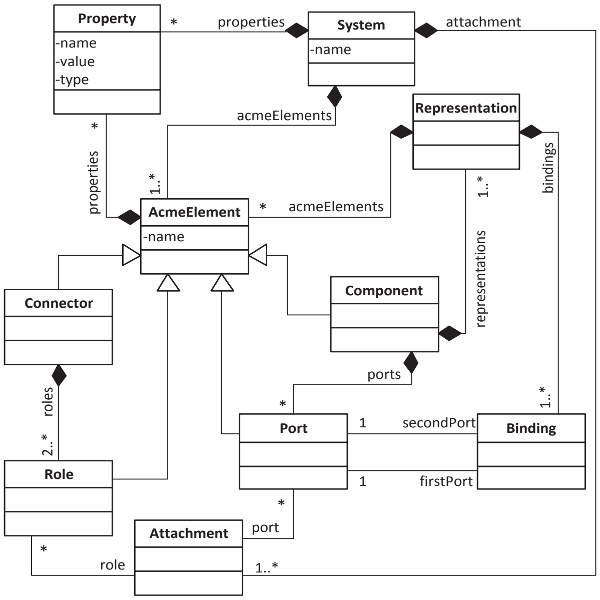


Figure 2. Acme metamodel

C. *STREAM*

The STREAM process includes the following activities: 1) Prepare requirements models, 2) Generate architectural solutions, 3) Select architectural solution, and 4) Refine architecture. In activities 1) and 2), horizontal and vertical rules are proposed, respectively. Horizontal rules are applied to the *i** requirements models to increase its modularity and prepare them for early architectural transformation. There are four horizontal transformation rules (HTRs) [9]. Vertical rules are used to derive architectural models in Acme from modularized *i** models. Non-Functional Requirements are used to select initial candidate architecture in the 3) activity. Certain architectural patterns can be applied to allow appropriate refinements of the chosen candidate architectural solution in 4) activity [5].

In the Vertical Transformation Rules (VTRs), the *i** actors and dependencies are mapped to Acme elements. Thus, an *i** actor is mapped to an Acme component. An *i** dependency becomes an Acme connector. The *depender* and *dependee* actors in a dependency can be mapped to the roles of a connector. In particular, we can distinguish between required ports (mapped from *depender* actors) and provided ports (mapped from *dependee* actors). Thus, a connector allows communication between these ports. A component provides services to another component using provided ports and it requires services from another component using required ports.

The four types of dependencies (goal, softgoal, task and resource) will define specific design decisions in connectors, ports and roles that are captured as Acme *Attachments*. An object dependency is mapped to a Boolean property. A task dependency is mapped directly to a port provided. The resource dependency is mapped to a return type of a property provided port. A softgoal dependency is mapped to a property with enumerated type

These transformation rules were defined in a semi formal way in [13] and now they need to be precisely specified using a suitable model transformation language, such as Query/View/Transformation Operational – QVTO [8]. In doing so, we can validate them, as well as provide support to (partially) automate the process, hence enabling the STREAM process to become a full-fledged MDD approach.

III. AUTOMATION OF THE VERTICAL TRANSFORMATION RULES

In the STREAM process, the user begins by using *i** to model the problem at hand. Some heuristics can guide the selection of sub-set(s) of candidate elements to be refactored. Once they are selected, the HTRs can be applied to improve the modularization of the *i** model [5].

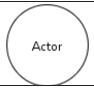
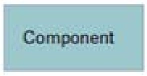
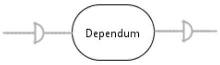
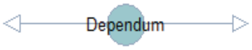
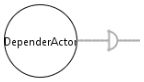
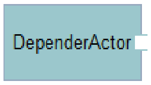
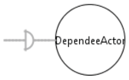
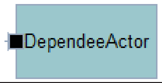
Since the vertical transformations do not consider the internal elements of the actors, we first create an intermediary SD model from the modular *i** SR model. We proceed to apply the VTRs (see Table I). The first rule (VTR1) maps *i** actors to Acme components, while VTR2 transforms *i** dependencies to Acme connectors. The VTR3 converts the *depender* actor onto a required port of the Acme connector. The VTR4 translates the *dependee* actor onto a provided port of the Acme connector.

We relied on the Eclipse based tool for *i** modeling, the iStarTool [6], to create the *i** model. This model is the input for the first STREAM activity. This tool generates a XMI of the modularized *i** SD model, which can be read by the QVTO Eclipse plugin [4], to serve as input for the VTRs execution.

The VTRs described in QVTO are based on the metamodels presented in Section 2, they are referenced during the execution of the transformation. The models created with the VTRs execution are represented as XMI files.

In our work, we were able to automate 3 horizontals (HTR2-HTR4) and 4 verticals transformation rules [14]. However, due to space limitation, in this paper, we only discuss how we dealt with the verticals rules. Table 1 illustrates the elements present in the source model and their corresponding elements present in the target model.

TABLE I. VERTICAL TRANSFORMATION RULES

	Source (*i**)	Target (Acme)
VTR1		
VTR2		
VTR3		
VTR4		

To map *i** actors to Acme components, we need to obtain the number of actors present in the modularized *i** SD model artifact. So, we create the same amount of Acme components, giving to this components, the same names of the *i** actors. The XMI file obtained as output of this transformation will contain the components represented by tags (Figure 6).

```
while(actorsAmount > 0) {
  result.acmeElements += object Component{
    name := self.actors.name->at(actorsAmount);
  }
  actorsAmount := actorsAmount - 1;
}
```

Figure 3. Excerpt of the QVTO code for VTR1

In the VTR2 each *i** dependency creates two XMI tags, one capturing the *depender* to the *dependum* connection and another one captures the *dependum* to the *dependee*.

In order to map these dependencies in Acme connectors it is necessary to recover the two dependencies tags, observing that have the same *dependum*. It is necessary not consider the actor which plays the role of *depender* in some dependency and *dependee* in another. Once this is performed, there are only *dependums* left. For each *dependum*, one Acme connector is created. The connector created receives the name of the *dependum* of the dependency link. Two roles are created within the connector, one named *dependerRole* and another named *dependeeRole*. The XMI output file will contain the connectors represented by tags (see Figure 4).

```
<acmeElements xsi:type="Acme:Connector" name="Conn">
    <roles name="dependerRole"/>
    <roles name="dependeeRole"/>
</acmeElements>
```

Figure 4. Connector in XMI

The VTR3 converts a *depender* actor into a required port present in the component obtained from that actor (see Figure 5). First, we create one Acme port for each actor *depender*. Each port created has a name and a property. The port name is given at random, just to control them. The property must have a name and a value, so to the property name is assigned "Required" as we are creating a required port and the value is *true*.

The XMI output file will contain "ports" tags within the *acmeElement* tag of component type. Moreover, since they are required ports, there will be one property with an attribute named "*Required*" whose value is set to "*true*".

Last but not least, the VTR4 maps all *dependee* actors to provided ports in the corresponding components obtained by those actors. For this, we list all *dependee* actors in the model. Every port generated has a name and a property. The port name is given at random. The port property name is "*Provided*", the port type is set to "*boolean*" and the port value is set to "*true*".

```
while(dependencyAmount > 0) {
  if(self.actors.name->includes(self.links.source-
>at(dependencyAmount).name) and
  self.actors.name-
>at(actorsAmount).=(self.links.source-
>at(dependencyAmount).name)) then {
    ports  += object Port{
        name := "port"+countPort.toString();
        properties := object Property {
          name := "Required";
          value := "true"
        };
    };
  } endif;
  dependencyAmount := dependencyAmount - 1;
  countPort := countPort + 1;
};
```

Figure 5. Excerpt of the QVTO code for VTR3

```
<acmeElements xsi:type="Acme:Component" name="Comp">
    <ports name="port17">
      <properties name="Provided" value="true"
       type="boolean"/>
    </ports>
</acmeElements>
```

Figure 6. Provided port, component and properties in XMI

After creation of the basics Acme elements, it is necessary to create the *Attachment* object, element responsible for associating the connectors to the required and provided ports present in the components. Therefore, an attachment is created for each port of a component. Each *Attachment* has a component as an attribute, a port, a connector and a role.

Next section presents an example to illustrate our approach.

IV. RUNNING EXAMPLE

MyCourses is a scheduling system that provides, as optimal as possible, a plan for scheduling courses. It allows universities to perform tasks, such as checking and listing available lecture rooms, teachers, students enrolled in any course. It was one of the project proposals for the ICSE 2011 Student Contest on Software Engineering [10].

The modularized i* SD model for the MyCourses system (Figure 7) was used as input model for the execution of the VTRs. These rules have the objective to transform a modularized i* SD model into an Acme initial architectural model.

After the automated application of the VTRs, a XMI model representing the output model and compatible with the Acme metamodel, will be generated. Figure 8 shows the graphical representation of that XMI model for the MyCourses system.

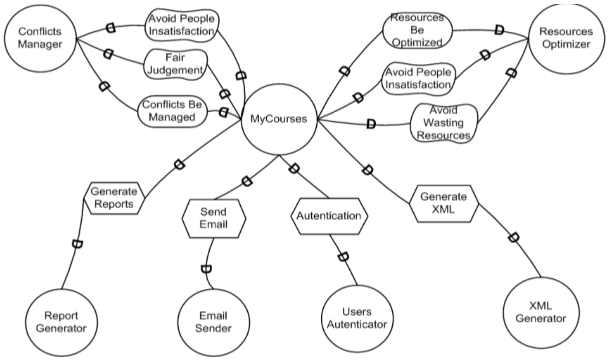


Figure 7. Modularized *i** SD model MyCourses

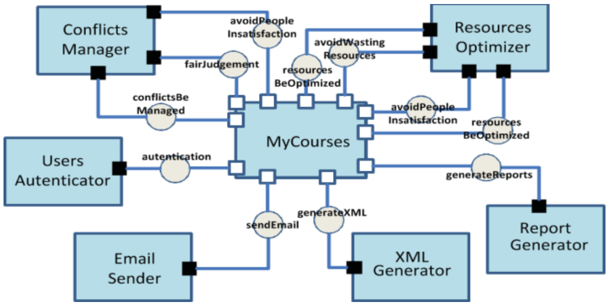


Figure 8. Acme Model from MyCourses

V. Related Work

Our work is unique in supporting the STREAM approach.

MaRiSA-MDD [15] presents a strategy based on models that integrate aspect-oriented requirements, architecture and detailed design, using AOV-graph, AspectualACME and aSideML languages, respectively. It defines representative models and a number of transformations between the models of each language. The transformation language used was ATL.

Silva et al [16] specify, through a model-driven approach, the transformations necessary for architectural models described in UML, from architectural organizational models described in i*. The transformation language used was ATL.

VI. Considerations and Future Works

It this paper, we advocated the use of model transformation to generate architectural models from requirements models. We reviewed the STREAM process, which defines and applies (manually) a set of model transformation rules to obtain Acme architectural models from i* requirements models.

In order to decrease time and effort required to perform the STREAM process and minimize the errors introduced by the manual execution of the transformation rules, we proposed to use the QVTO language to automatize the execution of these rules. Our focus was on the automation of the VTRs, responsible to generate an initial Acme architectural model. Metamodels of the i* and Acme languages were provided. The input models of the VTRs are compatible with the iStarTool and the output models are compatible with Acme metamodel, supported by the AcmeStudio tool.

Currently, the output of our process is an XMI file with the initial Acme architectural model. But the AcmeStudio tool reading files described in Acme textual language. As a consequence, the current architectural model cannot be graphically displayed. Hence, our plan is to provide new transformation rules to generate the textual representations. Case studies are still required to validate our approach.

References

[1] ACME. Acme. Acme - The AcmeStudio, 2009. Available in: <http://www.cs.cmu.edu/~acme/AcmeStudio/>. Accessed in: May 2012.

[2] GARLAN, D., MONROE, R., Wile, D. Acme: An Architecture Description Interchange Language. In: Proceedings of the 1997 conference of the Centre for Advanced Studies on Collaborative research (CASCON'97). Toronto, Canada.

[3] LUCENA, M., CASTRO, J., SILVA, C., ALENCAR, F., SANTOS, E., PIMENTEL, J. A Model Transformation Approach to Derive Architectural Models from Goal-Oriented Requirements Models. In: 8th IWSSA - OTM Workshops 2009. Lecture Notes in Computer Science, 2009, Volume 5872/2009, 370-380.

[4] ECLIPSE M2M. Model To Model (M2M). Available in: <http://eclipse.org/m2m/>. Accessed in: May 2012

[5] CASTRO, J.; LUCENA, M.; SILVA, C.; ALENCAR, F.; SANTOS, E.; PIMENTEL, J. C hanging Attitudes Towards the Generation of Architectural Models. Journal of Systems and Software, 2012.

[6] MALTA, A.; SOARES, M.; SANTOS, E.; PA ES, J.; ALENCAR, F.; CASTRO, J. iStarTool: Modeling requirements usingthe *i** framework. IStar 11, August 2011.

[7] OMG. Object Management Group. MDA Productivity Study, Juny 2003. Available in: <http://www.omg.org/mda/mda_files/MDA_Comparison-TMC_final.pdf>. Accessed in: May 2012

[8] OMG. QVT 1.1. Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification, January 2011. Available in: <http://www.omg.org/spec/QVT/1.1/>. Accessed in: May 2012

[9] LUCENA, M.; SILVA, C.; SANTOS, E.; ALENCAR, F.; CASTRO, J. Applying Transformation Rules to Improve *i** Models. SEKE 2009: 43-48.

[10] SCORE 2011. The Student Contest on Software Engineering - SCORE 2011, 2011. Available in: <http://score-contest.org/2011/>. Accessed in: May 2012.

[11] YU, E.; GIORGINI, P.; MAIDEN, N.; MYLOPOULOS, J. Social Modeling for Requirements Engineering. Cambridge, MA: MIT Press. 2011. ISBN: 978-0-262-24055-0.

[12] MENS, T.; CZARNECKI, K.; VAN GORP, P. A Taxonomy of Model Transformations. In: Proceedings of the Language Engineering for Model-Driven Software Development. Dagstuhl, Germany 2005.

[13] PIMENTEL, J.; LUCENA, M.; CASTRO, J.; SILVA, C.; SANTOS, E.; ALENCAR, F. Deriving software architectural models from requirements models for adaptative systems: the STREAM-A approach. Requirements Engineering Journal, 2011.

[14] SOARES, M. C. Automatization of the Transformation Rules on the STREAM process (In Portuguese: Automatização das Regras de Transformação do Processo STREAM). Dissertation (M.Sc.). Center of Informatic: UFPE, Brazil, 2012.

[15] MEDEIROS, A. MARISA-MDD: An Approach to Transformations between Oriented Aspects Models: from requirements to Detailed Project (In Portuguese: MARISA-MDD: Uma Abordagem para Transformações entre Modelos Orientados a Aspectos: dos Requisitos ao Projeto Detalhado). Dissertation (M.S.c). Center for Science and Earth: UFRN, Brazil, 2008.

[16] SILVA, C.; DIAS, P.; ARAÚJO, J.; MOREIRA, ANA. From Organizational Architectures in i* Agent-based: A model-driven approach (De Arquitecturas Organizacionais em i* a A rquitecturas Baseadas em Agentes: Uma abordagem orientada a modelos). WER'11.

Towards Architectural Evolution through Model Transformations

João Pimentel, Emanuel Santos, Diego Dermeval,
Jaelson Castro
Centro de Informática
Universidade Federal de Pernambuco
Recife, Brazil
{jhcp, ebs, ddmcm, jbc}@cin.ufpe.br

Anthony Finkelstein
Department of Computer Science
University College London
London, United Kingdom
a.finkelstein@ucl.ac.uk

Abstract—**The increasing need for dynamic systems, able to adapt to different situations, calls for underlying mechanisms to support software evolution. In this sense, model-based techniques can be used to automate one of the evolution aspects – the modification of models. However, the use of model-based techniques is tailored to the specific modeling languages being used. Thus, efforts to automate the modification of models can be undermined by the several different modeling languages used in different approaches for software evolution. Aiming to facilitate the use of model-driven development in the context of architectural evolution, we propose an approach to define basic transformation rules. This novel approach relies on a conceptual model and a set of basic operations for architectural evolution, which are used to define transformation rules for a specific architectural modeling language of interest. We illustrate the application of our approach by defining transformation rules for Acme using QVT.**

Keywords-software architecture; architectural evolution; autonomic systems

I. Introduction

Software evolution has become a key research area in software engineering [7]. Software artifacts and systems are subject to many kinds of changes at all levels, from requirements through architecture and design, as well as source code, documentation and test suites. Since the abstraction level of software architecture is adequate for identifying and analyzing the ramifications of changes [11], it could be one of the software evolution pillars [17]. As the architecture evolves, mechanisms are required for supporting these dynamic changes [1][14][19]. There are several approaches for tackling different aspects of architectural evolution, often relying in some kind of model transformation. However, these approaches do not use of model-driven engineering techniques, which provide underlying mechanisms for model transformation.

In this paper we present a novel approach for creating basic transformation rules with the focus of facilitating model-based architectural evolution. We defined a conceptual model and a set of basic operations that can be applied to the different languages used for architectural modeling. We illustrate our approach by defining transformation rules for architectural evolution on a specific ADL – Acme [9]. The contribution of this paper is twofold. On one hand, our basic operations can be used as a common vocabulary for the different architectural evolution approaches – facilitating their integration. On the other hand, it can guide the creation of transformation rules for a specific modeling language.

The remainder of this paper is structured as follows. In Section 2 we present the background for this work. Our conceptual framework is described in Section 3. Section 4 illustrates the use of our approach in Acme. Lastly, Section 5 concludes the paper.

II. Background

Architectural evolution has been acknowledged as a key element for achieving software evolution [17]. According to [6], there are 5 types of software evolution: Enhancive, Corrective, Reductive, Adaptive and Performance. In the case of autonomic, self-adaptive or self-managing systems this evolution is performed at runtime, with some degree of automation. Architectures that can evolve at runtime are classified as dynamic architectures. Our approach supports evolution both at design time and at runtime.

A survey on formal architectural specification approaches, regarding their ability to enact architectural changes at runtime is reported on [5]. That survey analyzes the approaches regarding the type of changes they support: Component Addition, Component Removal, Connector Addition and Connector Removal. As result, 9 out of 11 approaches were found to support all these basic operations: CommUnity, CHAM, Dynamic Wright, PiLar, Gerel, ZCL, Rapide, as well as the approaches by Le Métayer and by Aguirre-Maibaum. Moreover, all these 9 approaches have some kind of support for composing these operations. The need for structural change is clear in the architectural deployment view, for example, by replicating application servers in order to improve the system performance. Additionally, the use of subtyping mechanisms to enable architectural evolution has also been suggested [15]. From the 9 architectural description languages analyzed in [15], 4 show some kind of support for evolution through subtyping mechanisms: Aesop, C2, SADL and Wright.

As the elements and connectors of an architecture evolve, their properties may be modified as well. These properties may be related to the element itself (e.g. performance), or to the use of the element (e.g. workload). The modification of the properties may happen at design time or at runtime. For instance, at runtime, we may consider a connector (in a deployment view) that is reified in a physical network. Properties such as reliability and bandwidth of this connector may be subject to change over time. This kind of changes in properties of architectural elements is often ignored in the architectural models. We advocate that the evolution of these properties should be reflected in the models, as it allows us to (i) monitor the evolution of these properties, which in turn can be used to trigger adaptations, and to (ii) analyze the actual characteristics of a system using its architectural models. This alignment between what was designed (the initial model) and the actual implementation/deployment can help identify and reduce architectural erosion [15]. A particular research field that considers the evolution of the properties of architectural elements is that of service-oriented architectures. For instance [4][8] match services properties with non-functional requirements for selecting which services to use.

Some modeling languages that are not considered ADL can also be used for architectural modeling. Architectural behavior has been defined in languages such as Statecharts, ROOMcharts, SDL, Z, Use-Case, Use-Case Maps, Sequence diagrams, Collaboration diagrams and MSCs [2]. For instance, Statecharts can be used to describe the different states of a component, as well the transitions between them; Use-Case diagrams can express the different activities performed by an element of a system, from the user' point-of-view. Thus, when dealing with architectural evolution, we cannot neglect these languages. Similarly, the use of goal-based notations such as Kaos and *i** for architectural modeling has been promoted in some research endeavors [12][18].

III. Conceptual Framework

Based on a review of the architectural modeling approaches mentioned in the previous section, we devised a framework for empowering architectural evolution through model transformations, which can be applied to different modeling languages. This framework comprises a conceptual model, used to classify the different constructs of a modeling language, as well as a set of basic architectural evolution operations defined upon the conceptual model.

Architectural models are composed of architectural elements – e.g., components, services, classes and states – and links that define connections between these elements – e.g., connectors, requests, association links and events. Both elements and links may have properties, which can be used to provide a detailed description of elements and links. Moreover, elements may have sub-elements, i.e., elements that are part of them. Fig. 1 shows a model that represents these concepts. Using the graph terminology, elements can be considered typed nodes, links can be considered directed edges and properties can be considered labels of a node/edge.

This conceptual model can be applied to different architectural description languages, as well as to other modeling languages that are used for architectural modeling (e.g., Statecharts, Sequence diagrams, Use-Case and *i**). The (meta-)metamodel of the VPM language [3] resembles our conceptual model, being essentially composed by elements (entities) and links (relations) but neglecting their properties. That work provides a metamodeling language, whereas our conceptual model is used to classify the constructs of existing metamodels in order to guide the creation of transformation rules. Thus, our approach is not tied to any particular metamodeling nor transformation specification language.

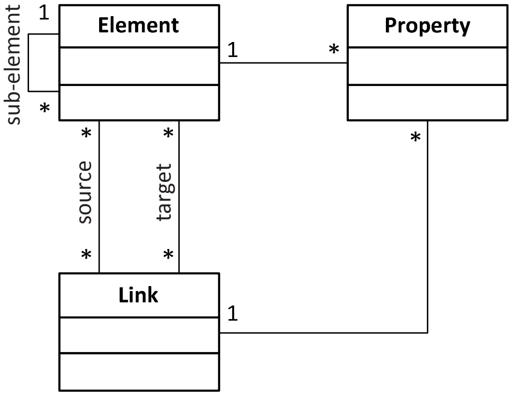


Figure 1. Conceptual Model

A. Basic Architectural Evolution Operations

This conceptual model enabled us to define 7 basic operations required to support architectural evolution with model transformations: Add Element, Remove Element, Add Link, Remove Link, Add Property, Remove Property and Change Property. These operations support the 2 different types of architectural model changes described in Section 2 – structural/topology changes and property changes. These operations are described next.

1) Add Element

Inserts a new element in a model. A particular case is when the new element is a sub-element of another element. Usually, an element has a name or an identifier.

2) Remove Element

Deletes an element from a model. Caution should be taken when removing elements from a model, as it is important to maintain the integrity of the model. For instance, if an element is removed, it is most likely that it will also be necessary to remove the links associated with that element.

3) Add Link

Inserts a new link connecting elements of the model.

4) Remove Link

Deletes a link from a model. Here again caution should be taken, as elements without links may result in invalid models.

5) Add Property

Inserts a property in an element or in a link. Usually, the property has a name or an identifier, as well as other attributes such as value, default value and type.

6) Remove Property

Deletes a property from an element or from a link.

7) Change Property

Modifies the content of an attribute of a property, e.g., the actual value, or its type. A similar effect could be achieved by combining the *Remove Property* and the *Add Property* operations. However, as modifying a property is conceptually different from replacing it, we decided to define this specific operation.

We decided against the definition of *Change Element* and *Change Link* operations as they essentially consist of changing their properties or adding/removing sub-elements.

These operations are similar to the five basic operations for graph-based model transformations [13]: create node, connect nodes, delete node, delete edge and set label. Our notion of property can be seen as an elaborated kind of label, leading us to defining three different property-related operations. Moreover, our conceptual model also supports the definition of sub-elements, which is an important feature in some architectural modeling languages.

B. Model Transformation Rules

In order to use the power of model-driven development for architectural evolution, it is necessary to define transformation rules for a particular modeling language. In order to describe these rules, the first step is to classify the constructs of the language based on the conceptual model of Fig. 1 - i.e., to identify which are the elements, sub-elements, links and properties of that particular language. Then one can proceed to instantiate the basic architectural evolution operations for each element, link and property of the language.

Once all basic operations are defined, the transformation rules can be developed using a model transformation framework (such as QVT) or using a general-purpose programming language. The development of these rules can be quite straightforward, as presented in the next section. However, the complexity of the language in focus may pose some additional challenges.

IV. Model Transformations for Architectural Evolution on Acme

In this section we illustrate the use of our conceptual framework to enable architectural evolution on a specific ADL: Acme [9]. Acme components characterize computational units of a s ystem. Connectors represent and mediate interactions between components. Ports correspond to external interfaces of components. Roles represent external interfaces of connectors. Ports and roles are points of interaction, respectively, between components and connectors – they are bound together through attachments. Systems are collections of components, connectors and a description of the topology of the components and connectors. Systems are captured via graphs whose nodes represent components and connector and whose edges represent their interconnectivity. Properties are annotations that define additional information about elements (components, connectors, ports, roles, representations or systems). Representations allow a component or a connector to describe its design in detail by specifying a sub-architecture that refines the parent element. The elements within a representation are linked to (external) ports through bindings.

A. Applying the conceptual framework

Considering our conceptual framework, we identified the basic operations required to evolve an architectural model in Acme. The Acme elements are *Component*, *Connector*, *Role, Port* and *Representation*. Particularly, the last three elements are sub-elements – *Role* is a sub-element of *Connector*, *Port* is a sub-element of component and *Representation* is a sub-element of both *Component* and *Connector*. Please note that *Connector* is not an actual link – instead, it is an element that is (indirectly) linked to *Component* through *Attachment*. The only links in Acme are *Attachment* and *Binding*. An attachment links an internal port of a component to a role of a connector, while a binding links an internal port to an external port of a representation. Lastly, *Property* expresses the properties of each element or of a *System*.

Thus, the basic operations for architectural evolution in Acme were defined: Add Component, Add Port, Add Connector, Add Role, Add Representation; Remove Component, Remove Port, Remove Connector, Remove Role, Remove Representation; Add Attachment, Add Binding; Remove Attachment, Remove Binding; Add Property; Remove Property; Change Property. Since all properties in Acme share the same structure, we did not need to define different property operations – e.g., there is no benefit in defining different *Add Component Property* and *Add Connector Property* operations.

B. Acme evolution with QVT

In this section we present how the basic operations can be implemented using a model transformation framework. Here we are using QVT, which comprises a language for specifying model transformations based on the Meta Object Facility – MOF and on the Object Constraint Language – OCL.

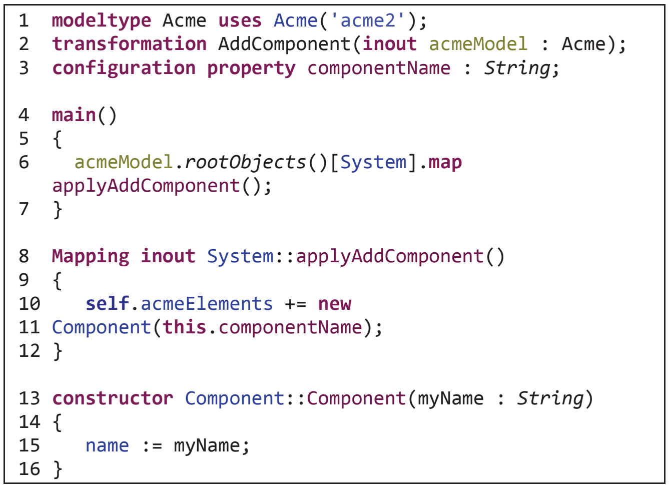
Fig. 2 s hows the QVT Operational code for the *Add Component* operation. The first line states the metamodel that will be used in the transformation, which is the Acme metamodel. The second line declares the *Add Component* transformation, informing that the same model (aliased *acmeModel*) will be used both as input and as output. Other than the input models, QVT transformations accept input through the mechanism of configuration properties. Line 3 presents the input variable of this transformation, which is the component name. In QVT, the entry point of the transformation is the signature-less *main* operation. Our *main* operation (lines 4-7) calls the mapping of the root object of the model, which is S*ystem*. The transformation itself is performed by the *Apply Add Component* mapping (lines 8-12), which inserts a new component. The component constructor is defined in lines 13-16, it simply assigns the given name to the component.

Fig. 3 presents the mapping of the *Remove Port* transformation (the other elements of the transformation are very similar to that of Fig. 2). In this mapping we traverse all ports of the model (Line 3), so that when port with the given name is found (Line 5) it is deleted from the model (Line 7).

Here we have described a straightforward implementation of the basic architectural evolution operations for Acme. However, there is room for improvement. For example, *where* clauses can be defined for preventing the addition of a component with an empty name, or the removal of a port that is still attached to some role.

V. Conclusion and Future Work

The conceptual framework defined in this paper is generic enough to allow its application on different architectural



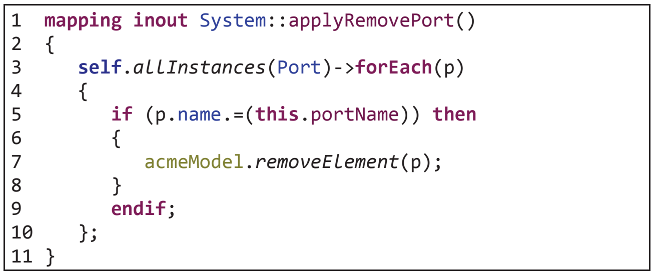
```
modeltype Acme uses Acme('acme2');
transformation AddComponent(inout acmeModel : Acme);
configuration property componentName : String;

main()
{
  acmeModel.rootObjects()[System].map
applyAddComponent();
}

Mapping inout System::applyAddComponent()
{
   self.acmeElements += new
Component(this.componentName);
}

constructor Component::Component(myName : String)
{
   name := myName;
}
```

Figure 2. QVT code for the Add Component operation



```
mapping inout System::applyRemovePort()
{
   self.allInstances(Port)->forEach(p)
   {
      if (p.name.=(this.portName)) then
      {
         acmeModel.removeElement(p);
      }
      endif;
   };
}
```

Figure 3. QVT code excerpt for the Remove Port operation

modeling languages of different architectural views (e.g., module, components and connectors, and allocation views [10]). By classifying the constructs of the modeling language according to our conceptual model, and then defining its basic operations, one can systematically develop transformation rules for that particular language. These rules can then be used to automate model transformation, in the context of model-driven engineering. Moreover, by defining transformation rules based on a common set of basic operations, the integration of different architecture evolution approaches can be facilitated. It is worth noting that our approach is not intended to replace current approaches for architectural evolution, but instead to empower them by facilitating the use of model transformations on the diverse set of modeling languages in use.

One of the limitations of our approach is that it does not consider architectural evolution as a whole, focusing solely on the modification of models. In future works we intend to explore the use of triggers to initiate the modification of the architectural model, as well as the selection of which modification to perform [20]. Additionally, we intend to automate the creation of the basic transformation rules for a given language, based on its metamodel. Moreover, we will investigate how this approach can be used in connection with modeling languages other than architectural ones.

References

[1] Allen, R., Douence, R., Garlan, D. Specifying and Analyzing Dynamic Software Architectures. Proc. of 1998 Conference on Fundamental Approaches to Software Engineering, Lisbon, Portugal, March, 1998.

[2] Bachmann, F., Bass, L., Clements, P., Garlan, D., Ivers, J., Little, R., Nord, R., Stafford, J. Documenting Software Architecture: Documenting Behavior. CMU/SEI-2002-TN-001, January 2002.

[3] Balogh, A., Varró, D. Advanced Model Transformation Language Constructs in the VIATRA2 Framework. ACM SAC, pp. 1280-1287, France, 2006.

[4] Baresi, L., Heckel, R., Thöne, S., Varró, D. Modeling and validation of service-oriented architectures: application vs. style. SIGSOFT Softw. Eng. Notes 28, 5 , pp. 68-77, September 2003.

[5] Bradbury, Jeremy S.; Cordy, James R.; Dingel, Juergen and Wermelinger, Michel (2004). A survey of self-management in dynamic software architecture specifications. In: Proceedings of the 1st ACM SIGSOFT Workshop on Self-managed Systems, November 2004.

[6] Chapin, N., Hale, J., Khan, K., Ramil, J., Than, W.. Types of software evolution and software maintenance. Journal of software maintenance and evolution, pp. 3–30, 2001.

[7] Fernandez-Ramil, J., Perry, D., Madhavji, N.H. (eds.) Software Evolution and Feedback: Theory and Practice, Wiley, Chichester (2006).

[8] Franch, X., Grünbacher, P., Oriol, M., Burgstaller, B., Dhungana, D., López, L., Marco, J., Pimentel, J. Goal-driven Adaptation of Service-Based Systems from Runtime Monitoring Data. In: Proceedings of the 5th International IEEE Workshop on Requirements Engineering for Services (REFS), Munich, Germany, July 2011.

[9] Garlan D, Monroe R, Wile D (1997) Acme: An Architecture Description Interchange Language. In: Proceedings of the 1997 conference of the Centre for Advanced Studies on Collaborative research (CASCON'97). Toronto, Canada.

[10] Garlan, D., Bachmann, F., Ivers, J., Stafford, J., Bass, L., Clements, P., Merson, P. Documenting software architectures: views and beyond, 2nd ed. Addison-Wesley Professional, 2010.

[11] Garlan, D., Perry, D. Introduction to the Special Issue on Software Architecture. In: Journal IEEE Trans. on Soft. Eng. Vol. 21, Issue 4 (1995)

[12] Grau, G., Franch, X. On the Adequacy of *i** Models for Representing and Analyzing Software Architectures. Proceedings of the ER Workshops 2007, LNCS 4802, pp. 296-305 (2007).

[13] Levendovszky, T., Lengyel, L., Mezei, G., Charaf, H. A Systematic Approach to Metamodeling Environments and Model Transformation Systems in VMTS. GraBaTs'04.

[14] Magee, J., Kramer, J. Dynamic Structure in Software Architectures. Proceeding SIGSOFT '96 Proceedings of the 4th ACM SIGSOFT symposium on Foundations of software engineering 1996.

[15] Medvidovic, N. A Classification and Comparison Framework for Software Architecture Description Languages. Technical Report UCI-ICS-97-02, University of California, February 1996.

[16] O'Reilly, C., Morrow, P., Bustard, D. Lightweight prevention of architectural erosion. Proceedings of the Sixth International Workshop on Principles of Software Evolution (IWPSW), pp. 59-64, September 2003.

[17] Oreizy, P., Medvidovic, N., Taylor, R. Architecture-Based Runtime Software Evolution. Proceedings of the International Conference on Software Engineering 1998 (ICSE'98). Kyoto, Japan, April 1998.

[18] Pimentel, J., Franch, X., & Castro, J. (2011). Measuring architectural adaptability in i* models. In Proceedings of the XIV Ibero-American Conference on Software Engineering, pp. 115-128, 2011.

[19] Pimentel, J., Lucena, M., Castro, J., Silva, C., Santos, E., Alencar, F. Deriving software architectural models from requirements models for adaptive systems: the STREAM-A approach. In: Requirements Engineering Journal, published online, 2011.

[20] Pimentel, J.; Santos, E.; Castro, J. Conditions for ignoring failures based on a requirements model. In: Proceedings of the 22nd International Conference on Software Engineering and Knowledge Engineering (SEKE). p. 48-53, 2010.

Using FCA-based Change Impact Analysis for Regression Testing*

Xiaobing Sun, Bixin Li, Chuanqi Tao, Qiandong Zhang
School of Computer Science and Engineering, Southeast University, Nanjing, China
State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences
{sundomore, bx.li, taocq, zhangqd}@seu.edu.cn

Abstract

Software regression testing is one of the most practical means to ensure software quality during software evolution. Test case selection and prioritization are two effective techniques to maximize the value of the increasing test cases, and usually studied independently. In this paper, we integrate these two techniques together, and finish them based on the impact results predicted by the change impact analysis (CIA) technique. First, we use formal concept analysis (FCA) to predict a ranked list of impacted methods from a set of changed classes. Then, test cases which cover these impacted methods are included in the new test suite. As each method predicted by the CIA is assigned with an impact factor (IF) value corresponding to the probability of this method to be impacted, test cases are ordered according to the IF values of the individual methods. Initial empirical study demonstrates the effectiveness of our regression testing approach.

1 Introduction

Software change is a fundamental ingredient of software maintenance. Changes made to software will inevitably have some unpredicted and undesirable effects on other parts of the software, thus may induce some faults to the modified software. And new faults induced during software maintenance can be identified and isolated by software change impact analysis and regression testing [2, 12, 15]. Change impact analysis (*CIA*) is an approach to identify potential ripple effects caused by changes made to software [2, 10]. And whether these potential effects inject faults into the software is often checked by software regression testing. Regression testing is one of the most practical means of ensuring software quality during software evolution. It is used to provide confidence that: 1) the modified parts of the program can run consistently with the new system, and 2) the unmodified parts are not affected by the modifications and can behave correctly as before. To date, a number of different approaches have been studied to regression testing [15], such as test case selection and test case prioritization. On the one hand, test case selection is used to reduce testing cost by selecting a subset of test cases from original test suite that are necessary to test the modified software [8, 12]. On the other hand, test case prioritization is used to identify an '*ideal*' ordering of test cases to yield benefits such as earlier feedback to testers and earlier fault detection [11].

While the research community has made considerable progress in regression testing areas, one important problem was overlooked, that is, most current regression testing techniques are proposed standalone, for example, some focused on test case selection [13, 8] while some on test case prioritization [4, 5], and they are usually studied independently. But in practice, test case selection and prioritization are performed together to generate a ranked list of test cases to be directly used. This paper is proposed to address this lack by combining test case selection and prioritization together. In our study, we use CIA to support regression testing activity. The CIA computes a ranked list of impacted methods from a set of changed classes based on the formal concept analysis (FCA) technique. And each method in the impact results is assigned with an *impact factor* (IF) value, which corresponds to the probability of this method to be impacted. On the one hand, test cases are selected based on the coverage information of these impacted methods; on the other hand, test cases are ordered based on the probability of the method to be impacted. And these two procedures are integrated together and performed through only a round.

This paper is organized as follows: in the next section, we introduce the FCA-based CIA technique. Section 3 presents our regression testing approach. In Section 4, em-

*This work is supported partially by National Natural Science Foundation of China under Grant No. 60973149, partially by the Open Funds of State Key Laboratory of Computer Science of Chinese Academy of Sciences under Grant No. SYSKF1110, partially by Doctoral Fund of Ministry of Education of China under Grant No. 20100092110022, partially by the Scientific Research Foundation of Graduate School of Southeast University under Grant No. YBJJ1102.

pirical evaluation is performed to show the effectiveness of our regression testing approach. In Section 5, some related work of regression testing techniques is introduced. Finally, we conclude and show some future work in Section 6.

2 FCA-based Change Impact Analysis

In this paper, our regression testing is performed based on the CIA technique. Here we use FCA-based CIA approach to predict the ripple effects induced by the proposed changed classes, which we give a simple introduction here. More details can be referred to our previous work [14].

2.1 Basics for Formal Concept Analysis

Formal Concept Analysis (FCA) is a field of applying mathematics based on the schematization of *concept* and *conceptual hierarchy* [7]. The input of the FCA process is the *formal context*, defined as:

Definition 1 (Formal Context) *A* ***formal context*** *is defined as a triple $\mathbb{K} = (\mathcal{O}, \mathcal{A}, \mathcal{R})$, where $\mathcal{R}$ is a binary relation between a set of formal objects $\mathcal{O}$ and a set of formal attributes $\mathcal{A}$. Thus $\mathcal{R} \subseteq \mathcal{O} \times \mathcal{A}$.*

With the formal context, concept lattice is generated based on the lattice constructing algorithm [7]. The concept lattice is composed of a set of *formal concepts*, defined as follows:

Definition 2 (Formal Concept) *A* ***formal concept*** *is a maximal collection of formal objects sharing common formal attributes. It is defined as a pair (O, A) with $O \subseteq \mathcal{O}$, $A \subseteq \mathcal{A}$, $O = \tau(A)$ and $A = \sigma(O)$, where $\tau(A) = \{o \in \mathcal{O} | \forall a \in A : (o, a) \in \mathcal{R}\} \wedge \sigma(O) = \{a \in \mathcal{A} | \forall o \in O : (o, a) \in \mathcal{R}\}$.*

$\tau(A)$ is said to be the *extent* of the concept and $\sigma(O)$ is said to be its *intent*. On the generated concept lattice, there are relations between these formal concepts, which forms a partial order on the set of all concepts. We use the following definition *subconcept* to denote the relations between different formal concepts [7]:

Definition 3 *Given two concepts $Co_1(O_1, A_1)$ and Co_2 (O_2, A_2) of a formal context, Co_1 is called the subconcept of Co_2, provided that $O_1 \subseteq O_2$ (or $A_1 \supseteq A_2$). we usually mark such relation as: $Co_1 \leqslant Co_2 \iff O_1 \subseteq O_2 \iff A_1 \supseteq A_2$*

The set of all concepts of a formal context forms a partial order, and composes a concept lattice, defined as follows.

Table 1. Formal context

	$M1$	$M2$	$M3$	$M4$	$M5$	$M6$	$M7$	$M8$	$M9$	$M10$
$C1$	×	×	×		×	×				
$C2$	×	×	×	×	×			×		
$C3$	×					×	×		×	
$C4$				×				×		
$C5$						×	×		×	×
$C6$				×						×

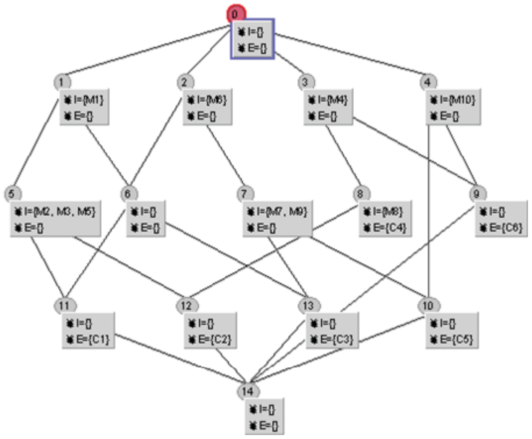


Figure 1. Graphical representation of the concept lattice

Definition 4 (Concept Lattice) *The concept lattice $\mathcal{L}(Co)$ is a complete lattice. $\mathcal{L}(Co) = \{(O, A) \in 2^{\mathcal{O}} \times 2^{\mathcal{A}} | O = \tau(A) \wedge A = \sigma(O)\}$, where infimum and supremum of two concepts (O_1, A_1) and (O_2, A_2) are defined as: $(O_1, A_1) \wedge (O_2, A_2) = (O_1 \cap O_2, \sigma(O_1 \cap O_2))$, and $(O_1, A_1) \vee (O_2, A_2) = (\tau(A_1 \cap A_2), A_1 \cap A_2)$, respectively.*

A formal context can be easily represented by a relation table, as shown in Table 1. In this table, rows represent formal objects and columns represent formal attributes. A *cross* (×) in row C and column M means that the formal object C has relationship with formal attribute M. By applying *Galicia tool*[1] to the formal context in Table 1, the concept lattice composed of formal concepts are generated. Figure 1 shows the corresponding graphical representation of the concept lattice. Each lattice node on the concept lattice indicates a formal concept, and is marked with its intent (*I Set*) and extent (*E Set*). The edges between them represent the containment relationship between concept intents, which forms a partial (hierarchical) order on the sets of all concepts.

2.2 Change Impact Analysis

CIA is an important predictive measurement of the ripple effects induced by the proposed changes. Here, the input of

[1] http://www.iro.umontreal.ca/ galicia/

Table 2. Impact set of $C1, C2, C5$ changes

Node	IS	*IF*
$co5$	$M2, M3, M5$	2.3
$co1$	$M1$	2.2
$co2$	$M6$	2.2
$co7$	$M7, M9$	1.5
$co8$	$M8$	1.5
$co4$	$M10$	1.5
$co3$	$M4$	1.3

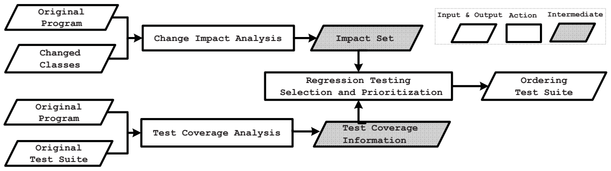


Figure 2. Overview of the regression testing approach

the CIA is composed of a set of changed classes, and its output is a ranked list of potentially impacted methods.

The way to use FCA to support CIA is as follows. First, we provide a formal context: formal objects are classes, and formal attributes are methods; the relation between them is defined as: class c is related to m, if and only if, at least one of the following conditions is satisfied: 1) m belongs to c, 2) m belongs to any superclass of c, 3) c depends on another method k calling m, and 4) c depends on another method k called by m.

With the formal context, concept lattice is generated [7]. As the containment relationship between concept intents forms a partial order on the sets of all formal concepts, the generated lattice structures methods into a hierarchical order. With this observation, impact results are computed based on two assumptions: (1) upward reachable methods are shared by an increasing number of unchanged classes, they are expected to be less and less affected by these changes; and (2) if upward methods are reachable from the nodes labeled by an increasing number of changed classes, these methods are more probably affected by these joint classes changes. According to the two assumptions, we propose an *Impact Factor* (IF_j) metric of the lattice node j, which is defined as:

$$IF_j = n + \frac{1}{\sum_{i=1}^{n} min(dist(j,i)) + 1}$$

In this formula, n is the number of changed classes upward reachable to lattice node j; $min(dist(j,i))$ is the least number of edges which need to be traversed straightforward upward from lattice node i to node j. Although the IF metric is defined on the lattice node, the lattice node is labeled by method(s). Thus the methods are also labeled by these IF values.

Through the CIA process, a ranked list of potentially impacted methods is produced, and these methods are ordered by the IF values showing their probability to be affected. We give a simple example to illustrate the CIA. Assuming there is a simple program and the dependencies between classes and methods have been analyzed. With the classes and methods, and their dependencies extracted from the program, a formal context is obtained as shown in Table 1. We apply FCA technique to this formal context and generate the concept lattice as shown in Figure 1. In addition, we assume $\{C1, C2, C5\}$ are the changed classes. CIA is then performed on this concept lattice to estimate their impact results. Thus, the IF values of upward reachable concepts labeling the potentially impacted methods are computed based on the above IF formula. Column 1 of Table 2 lists some lattice nodes reachable from the concepts labeled by the changed classes, Column 2 shows the impact set (IS), and Column 3 their corresponding IF values.

3 Regression Testing

In this paper, we propose an approach to perform two important activities in regression testing: test case selection and prioritization. Moreover, these two activities are integrated together and performed through only a round. The overview of our approach is shown in Figure 2.

From Figure 2, we see that regression testing is composed of two parts: change impact analysis (*CIA*) and test coverage analysis (*TCA*). From the CIA part, we obtain the potential ripple effects induced by the changed classes. We have presented our FCA-based change impact analysis technique in the above section. From the *TCA* part, we obtain the test coverage information of the test cases. Then through the results from these two activities, we compute a new ordering test suite for the modified program.

Different from previous regression testing techniques which are studied independently, we combine them together. For test case selection, our approach is performed by selecting the set of test cases which cover the methods potentially impacted by these changed classes; for test suite prioritization, test cases are ordered according to the probability of the method to be impacted, i.e., the methods with higher IF values may be more probably to be impacted by these changes, thus they are more preferential to be tested to check their correctness.

Algorithm 1 shows the algorithm of our regression testing approach. For the change set C, the algorithm first calculates a ranked list of impacted methods IS (Line 1). Then for each method m in the impact set (Loop 2-14), the al-

Algorithm 1 RegressionTesting

```
Input:
    $P$: original program
    $T$: test suite for $P$
    $E$: a set of executions
    $C$: a set of changed classes
Declare:
    $m$: a method
    $t$: a test case
    $IS$: a ranked list of potentially impacted methods
    $EXE$: a set of methods executed by a test case
Use:
    $CIA(C)$: returns a ranked list of methods potentially impacted by the set of
    changed classes $C$.
    $TCA(t)$: returns a set of methods covered by test case $t$.
Output:
    A set of ordering test cases $T^{'}$
 1: $IS = CIA(C)$
 2: for each $m$ in $IS$ do
 3:     for each test case $t$ in $T$ do
 4:         if $TCA(t) \cap m == \emptyset$ then
 5:             continue
 6:         end if
 7:         $T^{'} = T^{'} \cup t$
 8:         $T = T - t$
 9:         $EXE = TCA(t) \cap IS$
10:         if $EXE \neq \emptyset$ then
11:             $IS = IS - EXE$
12:         end if
13:     end for
14: end for
15: return $T^{'}$
```

gorithm checks whether the test case t traverses m, if not, continues to the next test case (Lines 4-6). If t traverses m, this test case is added to the new test suite $T^{'}$ (Line 7). Then, the methods in IS traversed by this test case are removed (Lines 9-12) because they have been covered by this test case. When all the methods in IS are processed, the whole regression testing process is finished. And the new test suite $T^{'}$ will traverse all the methods in the impact set, and the test cases in $T^{'}$ are ordered according to the possibility of the methods to be impacted. Thus, our approach directly generates a new ordering test suite. In Algorithm 1, there are two loops to scan the set of test cases and the impact set. So the time to our approach is $\mathcal{O}(|T| \times |M|)$, where $|T|$ is the size of the test suite, and $|M|$ is the size of the impact set. However, in most cases, the time of our approach is less because the two loops may end at an earlier time when some test cases can cover all the methods in the impact set.

We also give an example to exemplify our regression testing approach. The test coverage information of the above example program is assumed as Table 3. There are six test cases to coverage these 10 methods. The $\surd$ in row T and column M means that the test case can cover the method. Then, we use the results in Table 2 and Table 3 to generate an ordered test suite. According to Algorithm 1, we should first select the test case which covers the method with highest IF value in the new test suite. As $M2, M3, M5$ have the highest IF value and the test case $T1$ covers these methods, we should first select $T1$ into the new test suite. Then we see which methods in the impact set can be covered by this test case. As a result, these methods $\{M2, M3, M5, M6, M10\}$ are covered by $T1$, and should be removed from the impact set. At this time, we check the method with the second highest IF value, we identify the $M1$ method, and find $T2$ can cover this method. Thus $T2$ is added as the second test case in the new test suite. Similarly, we again remove all the methods in the impact set covered by this test case. After this, we select $T3$ into the new test suite, and find that there is no method in the impact set. The whole regression testing process is finished. Ultimately, we finish the process in three loops and three test cases are generated in the new test suite, in which $T1$ has the highest prioritization to be run, then $T2$, and followed by $T3$.

Table 3. Test coverage information

	$M1$	$M2$	$M3$	$M4$	$M5$	$M6$	$M7$	$M8$	$M9$	$M10$
$T1$		√	√		√	√				√
$T2$	√	√	√	√	√			√		
$T3$						√	√		√	
$T4$				√				√		
$T5$						√	√		√	√
$T6$		√								√

4 Initial Empirical Study

4.1 Setup

We use the Java Hierarchical Slicing and Applications ($JHSA$) program as our research subject. JHSA is a tool developed in our group, and used to construct hierarchical dependence graphs from package level to statement level [9]. We extracted six versions (V_0 to V_5) of $JHSA$ from its CVS repository, along with a test suite used to test the software. There are about 13 classes, 121 methods for $JHSA$. The test suite contains 72 test cases, which provide 80% method coverage of the program. In addition, to perform our study, we require fault data, and we utilized mutation faults [3] to study the test case prioritization. The average number of mutation faults of the program is 30.

In our study we use three measures to evaluate the proposed regression testing approach. They are the percentage of faults that the new test suite can identify (PF), the percentage of test cases selected from original test suite (PTS) and the weighted average of the percentage of faults detected ($APFD$) during the execution of the test suite, which is defined as follows:

$$APFD = 1 - \frac{TF_1 + TF_2 + ... + TF_m}{nm} + \frac{1}{2n}$$

In this Formula, n is the number of test cases, m is the number of faults revealed by T, and TF_i is the first test case which reveals fault F_i. The $APFD$ is a commonly used metric to evaluate the regression test suite prioritization [4]. And higher $APFD$ values imply better fault detection rates.

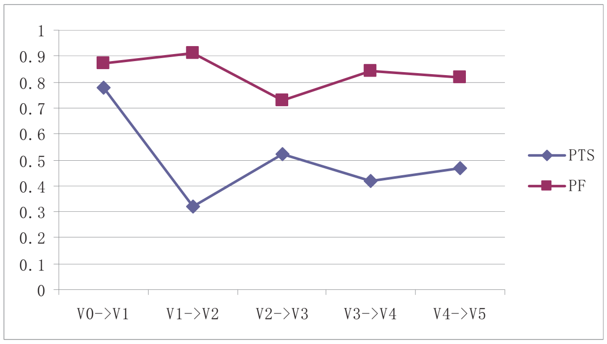


Figure 3. The PF and PTS results for the subject program

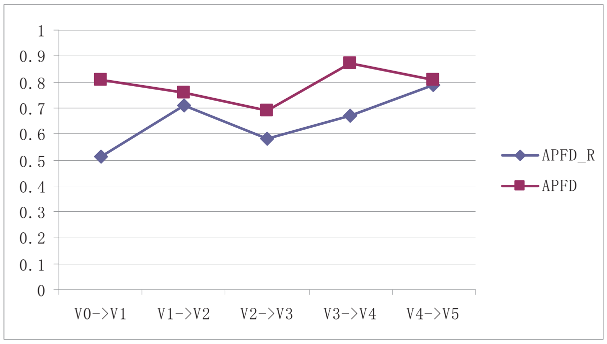


Figure 4. The $APFD$ and $APFD\_R$ results for the subject program

4.2 Results

First, we see the effectiveness of test case selection. Figure 3 shows the results of the PF and PTS values for the subject program during its evolution. From the results of PF, we see that our approach could select the test cases covering most of the faults in the program. The percentage of the identified faults can reach 80% above in most cases. In addition, Figure 3 also shows the percentage of test cases that were selected for each version of the subject. It illustrates that the percentage of selected test cases varies widely from about 30% to 80%, which is similar to the results from other studies evaluating test case selection on procedural programs [13]. The results do not show any obvious difference that is peculiar to object-oriented paradigm. Hence, from the results, we see that our approach can select a small set of test cases which can identify most of the faults.

Second, we see the effectiveness of the test case prioritization. To evaluate this, we manually interrupted the newly test suite $T^{'}$, and randomly gave another ordering test suite $T^{''}$ based on $T^{'}$. Then we computed the $APFD$ value for $T^{''}$ ($APFD\_R$). Figure 4 shows the $APFD$ and $APFD\_R$ values using our prioritization approach and the random prioritization approach, respectively. The data illustrates that our regression testing prioritization approach has higher $APFD$ values than that of random prioritization approach in all cases. Moreover, in most cases, the $APFD$ values of our approach are 10% (or above) higher than that of random prioritization approach. This shows that our prioritization approach is more effective compared to the random prioritization for its better ability of early fault detection.

4.3 Threats to Validity

In this section, we discuss some threats to the validity of our empirical study. The main external threat is the representativeness of our subjects and mutation faults. The subject program is of small size. Thus we cannot guarantee that the results can be generalized to other more complex or arbitrary programs. However, it is a real-world program widely used in our lab [14]. A second concern is the threat to construct validity. The goal of test case prioritization is to maximize some predefined criteria by executing the test cases in a certain order. Here, our focus is to maximize the rate of fault detection and we used $APFD$ to measure it. However, $APFD$ is not the only possible measure for fault detection rate. Some other measures may obtain different results. Finally, we consider the threat to internal validity. In our experiment, we utilized mutation faults to study the test case prioritization. Some other methods to generate the faults may be better to evaluate our regression testing approach. However, it is widely used by the academic community in evaluating test case prioritization [4, 5].

5 Related Work

A number of approaches have been studied to facilitate the regression testing process. Test case selection and prioritization are two effective techniques to conduct regression testing.

Test case selection attempts to reduce the size of original test suite, and focuses on identification of the modified parts of the program. To date, many regression test selection techniques have been proposed. Rothermel et al. proposed a safe and efficient regression test selection technique based on comparison of differences on the control flow graph [13]. In addition, regression testing based on slicing techniques has attracted much attention for a long time. And Binkley summarized the application of slicing techniques to support regression testing [1]. Orso et al. also proposed a similar regression testing approach to ours in this paper, which used the impact results from the CIA process to support regression testing [12]. They leveraged field data to support various testing activities, i.e., test case selection, augmentation, and prioritization.

Test case prioritization seeks to find the optimal permutation of the sequence of test cases, and hopes for early maximization of some desirable properties, such as the rate of fault detection [15]. Some work has been proposed to address this issue [4, 5]. Elbaum et al. attempted to improve the rate of fault detection for their prioritization technique based on statement-level and function-level coverage criteria [4]. In addition, Elbaum et al. proposed a metric, called average of the percentage of faults detected, to measure the effectiveness of these prioritization techniques. This metric is also used in this paper to evaluate the regression testing approach in the empirical study. However, this metric generated better results on two assumptions: all faults are of equal severity and all test cases have equal costs. To deal with the problem, they proposed a cost-cognizant metric, which took into account both the percentage of total test case cost incurred and percentage of total fault severity detected [5]. Elbaum et al. also conducted some empirical studies to compare some regression testing prioritization techniques [6].

Our regression testing approach in this paper is different from these introduced above. We integrate regression testing selection and prioritization together based on a ranked list of impact results from the FCA-based CIA process. Our goal is to first find those methods which are more probably impacted, and these methods may need suitable operation.

6 Conclusion and Future Work

This paper proposed a novel approach to regression testing, which integrated test case selection and prioritization together. Test case selection and prioritization were performed based on the impact results from the FCA-based CIA technique. Initial empirical study on a real-world program showed that: 1) the number of selected test cases ranges from about 30% to 80% of the original test suite, which can identify more than 80% faults, and 2) the ability of early fault detection of our approach is better than that of the random prioritization approach.

In the future, we will focus on a more complete empirical evaluation of our regression testing approach on open and large-scale programs. In addition, we will attempt to use different prioritization strategies based on the impact results to order the test cases, i.e., to prioritize the test cases according to the sum of *impact factor* values of the impacted methods covered by the test cases. Finally, we hope to compare our approach with current test case prioritization techniques under the same experimental benchmark.

References

[1] D. Binkley. The application of program slicing to regression testing. *Information and Software Technology*, 40(11-12):583–594, 1998.

[2] S. Bohner and R. Arnold. *Software Change Impact Analysis*. IEEE Computer Society Press, Los Alamitos, CA,USA, 1996.

[3] H. Do and G. Rothermel. On the use of mutation faults in empirical assessments of test case prioritization techniques. *IEEE Transactions on Software Engineering*, 32(9):733–752, 2006.

[4] S. G. Elbaum, A. G. Malishevsky, and G. Rothermel. Prioritizing test cases for regression testing. In *Proceedings of the International Symposium on Software Testing and Analysis*, pages 102–112, 2000.

[5] S. G. Elbaum, A. G. Malishevsky, and G. Rothermel. Incorporating varying test costs and fault severities into test case prioritization. In *Proceedings of the International Conference on Software Engineering*, pages 329–338, 2001.

[6] S. G. Elbaum, A. G. Malishevsky, and G. Rothermel. Test case prioritization: A family of empirical studies. *IEEE Transactions on Software Engineering*, 28(2):159C182, 2002.

[7] B. Ganter and R. Wille. *Formal Concept Analysis: Mathematical Foundations*. Springer-Verlag, Berlin, 1986.

[8] M. J. Harrold, J. A. Jones, T. Li, D. Liang, and A. Orso. Regression test selection for java software. In *Proceedings of the ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications*, pages 312–326, 2001.

[9] B. Li, X. Fan, J. Pang, and J. Zhao. A model for slicing java programs hierarchically. *Journal of Computer Science & Technology*, 19(6):848 –858, 2004.

[10] B. Li, X. Sun, H. Leung, and S. Zhang. A survey of code-based change impact analysis techniques. *Journal of Software Testing, Verification and Reliability, DOI: 10.1002/stvr.1475*, 2012.

[11] Z. Li, M. Harman, and R. M. Hierons. Search algorithms for regression test case prioritization. *IEEE Transactions on Software Engineering*, 33(4):225–237, 2007.

[12] A. Orso and M. J. Harrold. Leveraging field data for impact analysis and regression testing. In *Proceedings of the ACM SIGSOFT Symposium on Foundations of Software Engineering*, pages 128–137, 2003.

[13] G. Rothermel and M. J. Harrold. Empirical studies of a safe regression test selection technique. *IEEE Transactions on Software Engineering*, 24(6):401–419, 1998.

[14] X. Sun, B. Li, S. Zhang, C. Tao, X. Chen, and W. Wen. Using lattice of class and method dependence for change impact analysis of object oriented programs. In *Proceedings of the Symposium on Applied Computing*, pages 1444–1449, 2011.

[15] S. Yoo and M. Harman. Regression testing minimization, selection and prioritization: a survey. *Software Testing, Verification and Reliability*.

Forecasting Fault Events in Power Distribution Grids Using Machine Learning

Aldo Dagnino and Karen Smiley
ABB Corporate Research
Industrial Software Systems
Raleigh, NC, USA
aldo.dagnino@us.abb.com
karen.smiley@us.abb.com

Lakshmi Ramachandran
North Carolina State University
Department of Computer Science
Raleigh, NC. USA
lramach@ncsu.edu

Abstract— **Fault events in distribution grids and substations can cause costly power outages. Forecasting these fault events can reduce response time and enhance preparedness to repair the outage, which result in significant cost savings. Identification of fault events in distribution grids has been mostly a reactive and manual process with a relatively low level of automation. For this reason, any tools that can automate the diagnostics or prediction of fault events in the grid are welcome in the industry. The objective of the investigation presented in this paper is to develop machine-learning models capable of predicting fault events and their location in power distribution grids. Data related to historical fault events, grid electrical values, types of infrastructure, and historical weather were combined to create the forecasting models. A variety of machine learning algorithms such as Neural Networks, Support Vector Machines, Recursive Partitioning, and Naïve Bayes were utilized to create the machine learning models. Neural Network models performed best at forecasting fault events given certain weather conditions, and identifying the specific grid zone where a fault occurred. The Recursive Partitioning models were better at forecasting the substation and feeder where a fault occurred. An implementation at a US utility was prototyped to demonstrate the forecasting capabilities of these models.**

Keywords - machine learning, data mining, fault events, forecasting, power distribution grids, substations, weather.

I. INTRODUCTION

The electrical power utilized in cities, factories, office buildings, industries, and housing is p roduced in power generating stations or power plants. Such generating stations are conversion facilities where heat energy from fuel (coal, oil, gas, or uranium) and sun, as well as mechanical energy from wind, or falling water is converted into electricity. The transmission system transports electricity in large quantities from generating stations to the consumption areas. Electric power delivered by transmission circuits and power lines must be "stepped-down" in facilities called substations, to voltages more suitable for use in industries, buildings, and residential areas. The segment of the electric power system that takes power from a bulk-power substation to consumers, commonly about 35% of the total plant investment, is called the distribution system, and inc ludes the power distribution grid, electrical equipment, and the substations. Based on ABB's experience, it is estimated that over half of the power transmission and distribution infrastructure in the US and other parts of the western world is over 50 years old. A key issue currently facing Utilities is to efficiently distribute their limited maintenance and r epair funding. Studies in th e UK show that more than 70% of unplanned customer minutes lost of electrical power are due to problems in the distribution grid caused by deterioration or weather [8]. According to a survey conducted by the Lawrence Berkeley National Laboratory, power outages or interruptions cost the US around $80 billion annually [7]. Prediction of fault events in distribution grids is an important capability that helps utilities to reduce outage costs. For this reason, developing models that can forecast these fault events and their locations is h ighly desirable. In many utilities, distribution grid operators rely only on manual methods and reactive approaches to diagnose outages, and very limited fault forecasting methods. This makes the dispatching of repair crews a slow process that can be highly improved by utilizing more automated diagnostic and fault forecasting capabilities.

II. SOURCES OF FAULTS IN POWER DISTRIBUTION GRIDS

There are many factors identified in the literature that can cause fault events in a distribution grid [2] [3] [4] [6] [9] [11] [12] [14]. Fig 1 presents a summary of these factors.

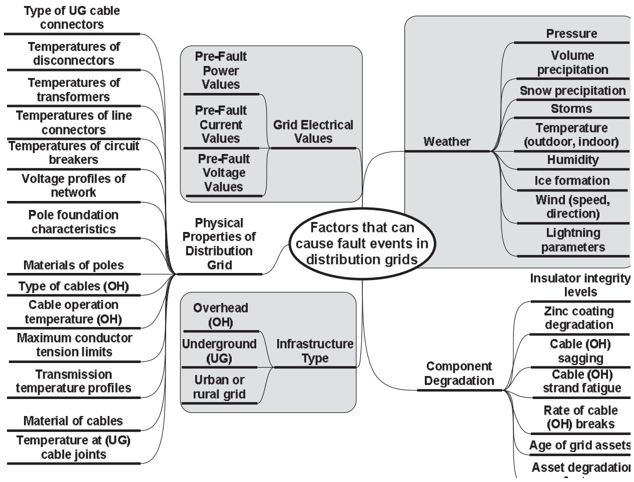

Factors that can cause fault events in distribution grids
Physical Properties of Distribution Grid
Type of UG cable connectors
Temperatures of disconnectors
Temperatures of transformers
Temperatures of line connectors
Temperatures of circuit breakers
Voltage profiles of network
Pole foundation characteristics
Materials of poles
Type of cables (OH)
Cable operation temperature (OH)
Maximum conductor tension limits
Transmission temperature profiles
Material of cables
Temperature at (UG) cable joints
Grid Electrical Values
Pre-Fault Power Values
Pre-Fault Current Values
Pre-Fault Voltage Values
Infrastructure Type
Overhead (OH)
Underground (UG)
Urban or rural grid
Weather
Pressure
Volume precipitation
Snow precipitation
Storms
Temperature (outdoor, indoor)
Humidity
Ice formation
Wind (speed, direction)
Lightning parameters
Component Degradation
Insulator integrity levels
Zinc coating degradation
Cable (OH) sagging
Cable (OH) strand fatigue
Rate of cable (OH) breaks
Age of grid assets
Asset degradatio

Figure 1. Factors that can contribute to faults in power distribution grids

The investigation described in this paper focuses on forecasting fault events in a distribution grid utilizing historical data on fault events that occurred in the grid and their associated electrical values, data on weather conditions at the time of the fault, and the type of grid infrastructure, as shown in the shaded areas in Fig. 1. Although large storms can cause considerable damage to distribution grids, “relatively normal” weather conditions can also have a significant impact on faults in the distribution grid, as shown by the results of this project. Lu et al. [11] discuss the influence that changing climatic conditions and weather have on the wear of electric equipment. Their analyses indicate that during the hot summer months, when the load on a feeders increase to 60-70%, there is an over-charge in the lines and assets, which can result in increased number of faults and reduced voltage of the distributed power.

III. Data Collection and Pre-processing

The results described in this investigation are associated with the study of a utility in the US whose identity and name cannot be revealed due to confidentiality agreements. Hence, the utility in this study will be referred to as Investor Owned Utility (IOU).

Several types of historical datasets associated with the IOU were collected and utilized during this investigation. The historical dataset types utilized in this investigation include: (a) fault data and electrical values from the IOU; (b) weather data; and (c) infrastructure type of the IOU. Fig. 2 shows the dataset types and their associated data attributes. The fault data from the IOU was collected utilizing an automated system developed at ABB. This system consists of intelligent electronic devices (IED’s) with sensing and analytic capabilities located at the end of the feeders of distribution lines. These IED’s monitor electrical values from the distribution lines, and are able to detect a fault event in the grid soon after it occurs. The historical fault data utilized includes these electrical values, which were corroborated with data entries documented by IOU engineers after restoring service. The weather data was collected from the US National Weather Service (NWS) and from the WeatherBug (WBUG) weather services. The NWS data was collected by their weather station every five minutes in METAR format. The WeatherBug data were collected from small weather stations located in various locations close to the different substations of the IOU. Finally, the lightning data were also obtained from the WeatherBug weather services organization. Fig. 1 has three gray rectangles that show the factors utilized to develop the machine learning algorithms discussed in this paper. It is expected that future studies will include data associated with the remaining factors in Fig. 1, to increase the precision and accuracy of the forecasted results.

An essential aspect of any data mining activity is preparing the data to be utilized for the analyses, or "data pre-processing". A software utility was developed in this project to automate the pre-processing of the raw datasets, to generate the data warehouse used to extract data for analyses, and to populate the forecasted results. The data pre-processing utility contains rules that address: (a) weather-fault time zone alignments; (b) weather-fault distances; (c) weather direction; and (d) information in free-format comments in the fault events, among others. Fig. 2 shows a detailed list of the data attributes utilized to run the prediction models presented in Section V.

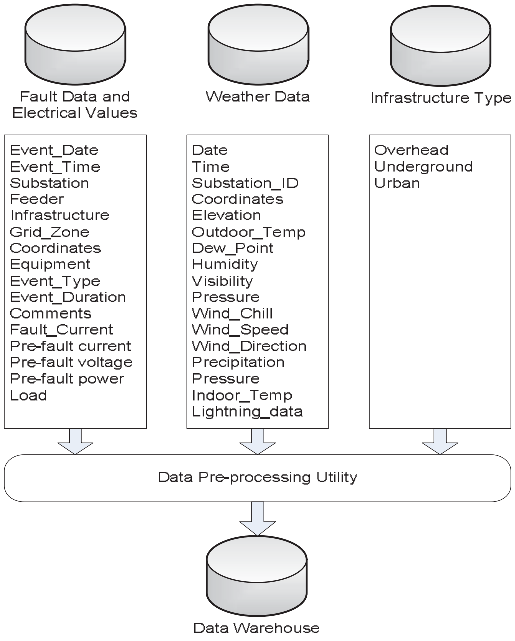


Figure 2. Primary data attributes utilized by machine learning algorithms

Data pre-processing involved cleaning of the fault, weather, and lightning data; aligning all of the datasets based on both time and geographic location; and fusing the various datasets together. A total of 1725 fault events were obtained over a two-year period, across eight feeders in four substations (two feeders per substation). These feeders range from a few miles apart to 10 miles apart. The main tasks in cleaning and pre-processing fault data included generation of “mineable” parameters from the fault comment texts, which described the equipment involved in the fault and the type of problem (e.g. animal contact). The fault infrastructure coding in the fault events was supplemented with information on the power grid topology, e.g. which feeders were almost entirely underground (UG) or overhead (OH). Five-minute weather observation data were obtained from NWS for one airport within 9-25 miles of the four substations, with 93% completeness (about 100,000 observation records per year). Hourly weather observation data were obtained from WeatherBug for four local “Earth Networks” weather stations within varying distances from the four substations at IOU (about 8000 observation records per year for a single weather station).

Preprocessing the five-minute airport weather data required several transformation steps: first, to translate the coded “METAR” strings to their equivalent text and numeric parameters; then, to parse and group the weather conditions into mineable nominal parameters. The hourly WeatherBug data contained somewhat different parameters than the five-minute METAR data (e.g., it included sunlight level readings which the airport data did not contain, and it did not have weather condition comments), but the pre-processing was otherwise similar. Lightning stroke data was also obtained for 2009 and 2010 from WeatherBug Total Lightning Network (WTLN) within a five-mile radius of each substation. Since this lightning stroke data were time-stamped to the micro-second, preprocessing it for data mining purposes required various aggregate counts and sums of strokes and amplitudes for both intra-cloud and cloud-to-ground lightning. These aggregates were calculated for both five-minute periods and one-hour periods so the lightning counts could be joined to the weather data.

To enable selection of the "closest" weather data for each fault, preparing all of this data for mining required aligning both the timestamps and the geographic locations. Calculating geographic location parameters was also different for each dataset. For the airport weather data, the precise (lat, lon) of the airport weather station was known. For the hourly weather data, the (lat, lon) values were not available for the weather stations, so the geographic center of each weather station's zip code was used. Since each microsecond-time-stamped lightning stroke had its own unique (lat, lon), a reference (lat, lon) was determined algorithmically to tag the five-minute and one-hour aggregate records. The precise (lat, lon) had not been recorded for many faults, so the (lat, lon) of the associated substation was used for all fault records to ensure consistency.

Each fault and weather dataset had a different reference time zone: some used local time and some used UTC, and some that used local time reflected Daylight Savings while others did not. Therefore, our preprocessing included calculation of a new timestamp parameter for each dataset, adjusted to the same reference time zone (local standard time was chosen). The lightning aggregates were then joined to the five-minute weather data and the hourly weather data. Approximate 'Great Circle' distances were calculated between each weather station and each substation, and for each lightning aggregate, the distances to the four substations were calculated. These distances were used to choose for each fault the "closest" five-minute and "closest" one-hour weather record, using both timestamp and distance.

IV. FORECASTING FAULT EVENTS

Several approaches to predicting fault events in a distribution grid have been proposed. Butler [1] discusses a failure detection system, which makes use of electrical property parameters (such as feeder voltages, phase currents, transformer windings' temperatures, and noise from the transformer during its operation) to identify failures. Gulachenski and Bsuner [6] use a Bayesian technique to predict failures in transformers. Some of the features considered in these studies include ambient temperature, varying loads on the transformer, and age-to-failure data of transformers. Quiroga et al. [12] search for fault patterns assuming the existence of relationships between events. Their approach considers factors such as values of over-currents of past fault data and the sequence, magnitude, and duration of the voltage drops. Although the above mentioned approaches are predictive in nature, they do not consider weather properties to predict faults. The hypothesis associated with the work presented in this paper is that a fault event occurrence is likely to follow a pattern with respect to weather conditions, infrastructure type, and electrical values in the distribution grid at the time of the fault (as seen in Fig. 1).

The objective of the investigation discussed in this paper is to develop the basis of a Decision Support System (DSS) using machine learning that forecasts fault events in the power distribution grid based on expected weather conditions. The strategy has two primary phases as shown in Figure 4. In the first phase, the historical data collected (as explained in Section 3) is utilized to develop and train machine learning models that can foretell fault events using weather forecasts. During the first phase, four algorithms are utilized to perform five generic analyses (fault prediction, zone prediction, substation prediction, infrastructure prediction, and feeder prediction) and train the models. Four measurements are used to compare machine learning algorithms: precision, recall, accuracy, and *f*-measure. Precision determines the fraction of records that actually turns out to be positive in the group the classifier has declared as a positive class. Recall is computed as the fraction of correct instances among all instances that actually belong to the relevant subset. Accuracy is the degree of closeness of the predicted outcomes to the actual ones. The *f*-measure is the harmonic mean of precision and recall.

During the second phase, the best performing algorithm is selected for each of the five analyses, to be utilized as part of the DSS for future predictions at the IOU.

V. CREATION OF MACHINE LEARNING MODELS

Supervised classification techniques are utilized to forecast the occurrence of faults in the distribution power grid of the IOU. Four supervised classification machine learning algorithms are utilized to conduct the analyses: Neural Networks (NN), kernel support vector machines (KSVM), decision-tree based classification (recursive partitioning; RPART), and Naïve Bayes (NB). A NN is an interconnected group of artificial neurons that use a computational model that allows them to adapt and change their structure based on external or internal information that flows through the network. A SVM is a linear binary classification algorithm [1]. Since the datasets consist of more than two classes, we choose to use kernel support vector machine (KSVM), which has been found to work in the case of non-linear classifications. RPART is a type of decision tree algorithm that helps identify interesting patterns in data and represents them as a tree. RPART is chosen because it provides a suitable tree-based representation of any interesting patterns that are observed in the data sets, and also because it works well with both nominal and continuous data. NB is a probabilistic classification technique that works well with relatively small datasets. These four algorithms are selected because of their distinct properties and their ability to work with different types and sizes of data, and the objective is to select the algorithm that performs the best for the type of prediction being conducted. Five analyses are conducted utilizing these four algorithms: (a) fault event prediction; (b) grid zone prediction; (c) substation prediction; (d) type of grid infrastructure; (e) feeder number prediction. As mentioned earlier, the primary data attributes shown in the shaded areas of

Fig. 1 and the data elements in Fig. 2 are considered when creating and training the predictive models.

A. Fault Prediction Models

Four models are created to identify weather patterns that are most likely to result in a fault event using the NN, KSVM, RPART, and NB algorithms. The models were constructed by taking weather data points joined to fault events, as well as random weather data samples when no fault events were recorded in the selected IOU substations. Since there were a large number of weather points for the times at which a fault did not occur, the random sample of records is taken making sure that all months and days and hours in the day are covered in the sample. The dataset contained a total of 3471 records (1725 with faults and 1746 without faults) of which 2430 were used for training each of the four models and 1041 for testing the models (see Fig. 3). The selection of records without faults was done in a random fashion. The best-performing model is the one created with the feed-forward trained by a multi-layer perceptron back-propagation NN algorithm (see shaded area in Fig. 3). This trained model produced an accuracy of 75%, an average precision of 77%, an average recall of 73%, and an *f*-measure of 75%. Table I shows a sample of the results generated in Rapid Miner for the NN model.

B. Zone Prediction Models

The four zone prediction models were trained by considering historical fault data from the IOU grid and weather data. Of the 1725 records with faults and weather data, 70% were used for training and 30% for testing the trained models. The output of these models predict in what zone (AMZ, UMZ, PMZ) on the IOU grid the fault occurred. The best-performing model was the one created training a Neural Network algorithm, as shown in the shaded area in Fig. 3. The model contains one hidden layer with 20 nodes. The model produced an accuracy of 66%, an average precision of 69%, an average recall of 68%, and an *f*-measure of 68%.

C. Substation Prediction Models

The four substation prediction models were trained by considering historical fault data from the IOU grid and weather data. Of the 1725 records with faults and weather data, 70% were used for training and 30% for testing the trained models. The output of these models predicts the IOU substation ID where the fault occurred. The best performing model was the one created with the RPART algorithm, as shown in the shaded area in Fig. 3. This trained model produced an accuracy of 59%, an average precision of 66%, an average recall of 54%, and an *f*-measure of 59%.

D. Infrastructure Prediction Models

The four infrastructure prediction models were trained by considering historical fault data from the IOU grid and weather data. Of the 1725 records with faults and weather data, 70% were used for training and 30% for testing the trained models. The output of these models predicts the type of infrastructure, OH (overhead) or UG (underground) on the section of the IOU grid where the fault occurred. The best-performing model was the one created training a Neural Network algorithm, as shown in the shaded area in Fig. 3. The model produced an accuracy of 77%, an average precision of 62%, an average recall of 52%, and an *f*-measure of 57%.

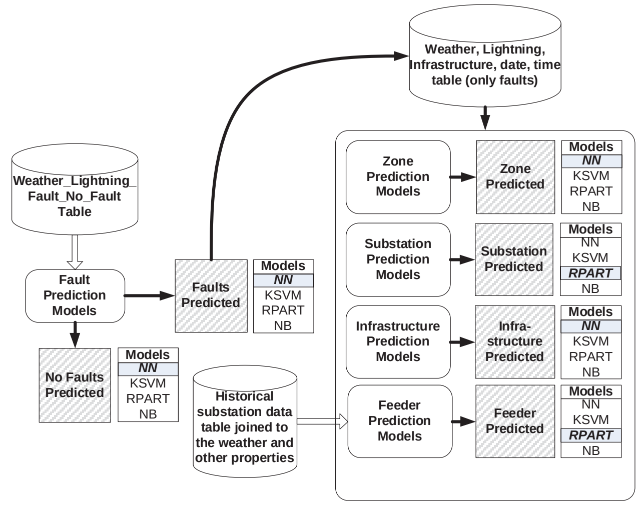
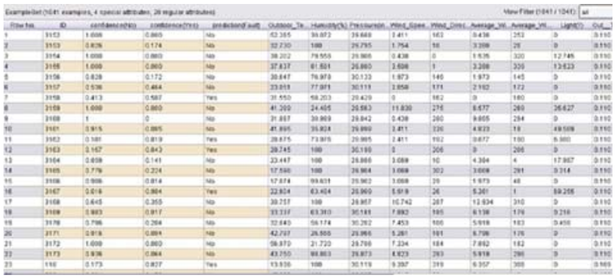


Figure 3. Machine Learning Models

TABLE I. NEURAL NETWORK OUTPUT OF FAULT PREDICTION MODEL



ExampleSet (1041 examples, 4 special attributes, 26 regular attributes)   View Filter (1041 / 1041): all

Row No.	ID	confidence(No)	confidence(Yes)	prediction(Fault)	Outdoor_Te...	Humidity(%)	Pressure(in...	Wind_Spee...	Wind_Direc...	Average_Wi...	Average_Wi...	Light(?)	Out_...
1	3152	1.000	0.000	No	52.385	36.872	29.668	2.411	163	0.438	252	0	0.110
2	3153	0.826	0.174	No	32.230	100	29.795	1.754	56	3.288	28	0	0.110
3	3154	1.000	0.000	No	38.202	79.558	29.908	0.438	0	1.535	320	12.749	0.110
4	3155	1.000	0.000	No	37.837	61.601	29.860	2.630	1	3.288	320	13.623	0.110
5	3156	0.828	0.172	No	30.647	76.978	30.133	1.973	140	1.973	145	0	0.110
6	3157	0.536	0.464	No	23.018	77.971	30.119	2.850	171	2.192	172	0	0.110
7	3158	0.413	0.587	Yes	31.550	58.203	29.429	0	162	0	160	0	0.110
8	3159	1.000	0.000	No	41.300	24.405	29.683	11.830	276	8.677	260	26.627	0.110
9	3160	1	0	No	31.887	39.969	29.842	0.438	260	9.605	284	0	0.110
10	3161	0.915	0.085	No	41.895	35.824	29.890	2.411	126	4.823	18	49.508	0.110
11	3162	0.181	0.819	Yes	28.875	73.975	29.985	2.411	192	0.877	180	8.300	0.110
12	3163	0.157	0.843	Yes	28.745	100	30.150	0	206	0	206	0	0.110
13	3164	0.859	0.141	No	23.447	100	29.866	3.068	10	4.384	4	17.987	0.110
14	3165	0.776	0.224	No	17.540	100	29.964	3.068	302	3.068	291	0.314	0.110
15	3166	0.986	0.014	No	17.874	99.631	29.962	3.068	20	1.973	48	0	0.110
16	3167	0.016	0.984	Yes	22.824	63.404	29.969	5.918	26	5.261	1	69.258	0.110
17	3168	0.645	0.355	No	30.757	100	29.957	10.742	287	12.934	310	0	0.110
18	3169	0.983	0.017	No	33.337	63.310	30.181	7.892	195	6.138	176	0.216	0.110
19	3170	0.796	0.204	No	32.640	56.174	30.262	7.453	100	5.918	183	0.458	0.110
20	3171	0.916	0.084	No	42.707	26.566	29.966	5.261	161	6.798	176	0	0.110
21	3172	1.000	0.000	No	56.870	21.720	29.788	7.234	184	7.892	182	0	0.110
22	3173	0.936	0.064	No	43.750	66.863	29.873	6.823	263	5.918	286	0	0.110
23	116	0.173	0.827	Yes	13.538	100	30.119	9.297	319	6.357	300	0	0.569

E. Feeder Prediction Models

The four feeder prediction models were trained by considering historical fault data from the IOU grid and weather data. Of the 1725 records with faults and weather data, 70% were used for training and 30% for testing the trained models. The output of these models predicts the IOU Feeder where the fault occurred. The best-performing model was the one created with the recursive partitioning algorithm, as shown in the shaded area in Fig. 3. This trained model produced an accuracy of 74%, an average precision of 79%, an average recall of 70%, and an *f*-measure of 74%.

F. Comparative f-measures of Predictive Models

Fig. 4 displays a graph with the average *f*-measure values from the four different models created for each analysis. The *f*-measure is calculated as the harmonic mean of precision and recall, using the formula given in Eq. 1. Precision pertains to the fraction of classified set of data points that have been correctly classified. Precision can be viewed also as the probability that a (randomly selected, using the uniform distribution) retrieved data point is relevant. Recall is the fraction of the actual set of data points that have been correctly classified. Recall is the probability that a (randomly selected) relevant data point is retrieved in a search. Precision and recall are calculated using the formulas in Eq. 2 and Eq. 3 respectively.

$$f - \text{measure} = 2 * \frac{precision * recall}{(precision + recall)} \quad (1)$$

$$precision = \frac{\#\ of\ records\ classified\ correctly\ into\ a\ certain\ class}{total\ \#\ of\ records\ classified\ into\ that\ class\ by\ the\ model} \quad (2)$$

$$recall = \frac{\#\ of\ records\ classified\ correctly\ in\ a\ certain\ class}{total\ \#\ of\ records\ that\ actually\ belong\ to\ that\ \ class} \quad (3)$$

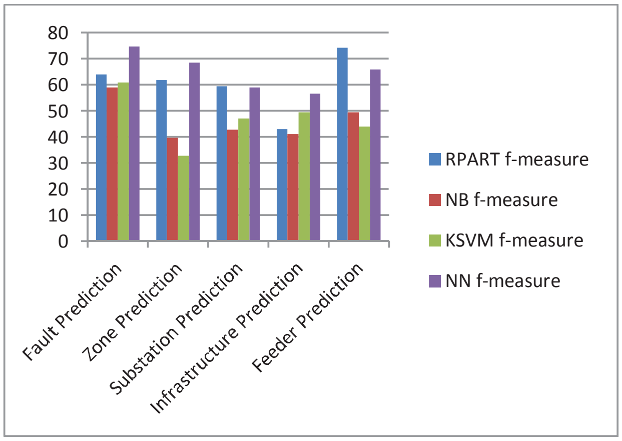


Figure 4. Performance Comparison of Machine Learning Models

Based on the highest *f*-measure of the trained models for each analysis, the following selections were made to become the basis for the IOU Decision Support System:

a) NN model was selected to predict faults based on weather;

b) NN model was selected for prediction of the zone in the grid where a fault may occur;

c) RPART model was selected to predict the substation where a fault may occur;

d) NN model was selected to predict if the fault occurs in the overhead or underground lines;

e) RPART model was selected to predict the feeder where a fault may occur in the grid.

VI. Forecasting Fault Events at IOU

The analytic models developed in thi s project and summarized in Section V formed the basis for the development of a decision support system (DSS) at IOU to forecast fault events in their power distribution grid. The analytic models have been trained using historical data which is pertinent to the IOU infrastructure and weather patterns in the r egions where their distribution grid is located. Fig. 5 depicts the modular view of the architecture of the fault event forecasting DSS, and the output types that are expected from the trained machine learning models. Given a certain weather forecast or forecasts, the Forecast Prediction NN model identifies if o ne or more fault events are expected. If a fault event is expected, then the remaining trained algorithms will determine in what zone, substation, and feeder the fault is expected to occur, and also if the fault will occur in an overhead or underground line. The modules presented in Fig. 5 form the basis for the DSS for forecasting fault events in the power distribution grid.

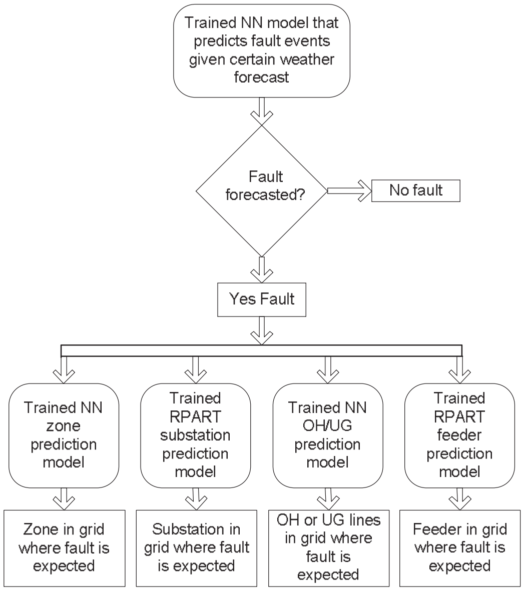

Trained NN model that predicts fault events given certain weather forecast
Fault forecasted?
No fault
Yes Fault
Trained NN zone prediction model
Trained RPART substation prediction model
Trained NN OH/UG prediction model
Trained RPART feeder prediction model
Zone in grid where fault is expected
Substation in grid where fault is expected
OH or UG lines in grid where fault is expected
Feeder in grid where fault is expected

Figure 5. Machine Learning Models as Basis for DSS

VII. Conclusions and Future Work

This paper presented a machine learning approach to forecast fault events in a p ower distribution grid. The investigation developed and summarized in this paper was conducted as an applied research project utilizing real-life historical data from a power distribution utility (referred as IOU) in the US. The data from IOU dates as far back as 2008, and includes historical data on fault event occurrences, the electrical values from the grid at the time that the fault event occurred, as well as data associated with the topology of the grid (overhead and underground lines). Historical weather records collected from the US National Weather Service (NWS) and from the WeatherBug (WBUG) weather service during the time period when the fault events occurred were also included in the datasets employed to train the machine learning models. An additional historical dataset on lightning, also provided by WBUG, was included as part of the weather data. The fields used to fuse fault event data and weather data were the location and time stamp associated with the fault event. In situations when readings of weather are taken over certain intervals of time, a decision has to be made about what weather reading will be associated with a specific time stamp. In our study, the weather reading utilized was the one closest after the fault time stamp reading, since that weather reading covered the observation interval just ended which encompassed the fault time. However, an alternate approach would be to use a weather reading preceding the time stamp reading.

Once all datasets were collected, all these data points were aligned using the closest reading to the location and time stamp at which the fault event occurred in a power line, to create an analysis dataset. Moreover, a random selection of weather data points with no faults was also included in the datasets utilized to train and test the machine learning models. A total of 1725 faults occurred during the study period. The dataset then consisted of fault-related data and weather corresponding to the 1725 fault events as well as 1746 weather entries when no faults occurred.

The machine learning algorithms evaluated for creating the prediction models include neural networks, kernel support vector machines, recursive partitioning, and Naïve Bayes. The *f*-measure was utilized to evaluate the overall performance of the machine learning algorithms; the average *f*-measure value for the selected prediction models was 67%. The models that performed the best included neural networks and recursive partitioning, which were selected to become the basis for the fault prediction decision support system (DSS). If we were to pick only one machine learning algorithm that performed the best across all five prediction areas, it is the neural network algorithm. The authors believe this is because the inputs to the model are well understood (we understood which data attributes were likely to be important for prediction, but did not know how to combine them). Moreover, the predicted data attributes were clearly defined. Another reason was that we had enough examples to be able to train the neural network algorithms.

It is the authors' expectation that including data such as physical properties of the power grid (e.g. pole foundation characteristics, materials of poles and cables, types of connectors) to train the machine learning algorithms in the DSS can result in better predictive models. Similarly, including historical data on maintenance (such as rate of cable breaks, equipment replacement history, and specific maintenance history) can also enhance the predictive capability of the models. Also, including component degradation data such as insulator integrity levels, cable zinc coating degradation, overhead cable sagging, cable strand fatigue, age of assets, etc., can also positively contribute to better-performing predictive algorithms.

Other areas of future work include extending this study in multiple dimensions. First, adding more years of both fault and weather data for these substations can improve the training of the algorithms. Second, having more precise (lat, lon) locations will enable a closer alignment of faults with weather data. Third, including data from more substations and from utilities which face substantially different climatic conditions can help enhance the capabilities of the DSS algorithms.

It is the belief of the authors, based on the investigation conducted and presented in this paper, that the use of machine learning algorithms to help forecast fault events in the power distribution grid has the potential of reducing the time and effort in restoring electrical power in the grid after a fault event has occurred.

ACKNOWLEDGMENTS

The authors wish to recognize and thank the WeatherBug organization for sharing a portion of the historical weather data that was utilized for the analyses conducted in this work. The positive results of this project were enhanced by the data and help provided by the WeatherBug weather services.

REFERENCES

[1] I. Berg, H. Lobl, S. Grossman, and F. Golletz, "Thermal behaviour of network components depending on outdoor weather conditions", CIRED, 20th International Conference on Electricity Distribution, Prague, June 8-11, 2009, paper no. 0568.

[2] J. S. Bowers, A. Sundaram, C. L. Benner, and B. D. Russell, "Outage avoidance through intelligent detection of incipient equipment failures on distribution feeders", in IEEE Power and Energy Society General Meeting - Conversion and Delivery of Electrical Energy in the 21st Century, 2008, pp. 1-7.

[3] K. Butler, "An expert system based framework for an incipient failure detection and predictive maintenance system", Int. Conf. ISAP 1996, 1996, pp. 321 – 326.

[4] M. Chow, L. S. Taylor, "Analysis and prevention of animal-caused faults in power distribution systems", IEEE Transactions on Power Delivery, vol. 10, no. 2, 1995, pp. 995-1001.

[5] C. Cortes and V. Vapnik, "Support-vector networks", Machine Learning, vol. 20, no. 3, pp. 273-297, 1995.

[6] E.M. Gulachenski and P.M. Bsuner, "Transformer failure prediction using bayesian analysis", IEEE Trans. on Power Systems, vol. 5 no. 4, 1990, pp. 1355 - 1363.

[7] K. Hamachi LaCommare, and J. Eto, "Understanding the cost of power interruptions to U.S. electricity consumers", September 2004 report.

[8] J. Hamson, "Urban network development", Power Engineering Journal, pp. 224-232, 2011.

[9] P. Heine,J. Turunen, M. Lehtonen, A. Oikarinen, "Measured Faults during Lightning Storms", Proc. IEEE Power Tech 2005, Russia, 2005, pp.1- 5.

[10] G. F. Linoff and M. J. A. Berry, Data Mining Techniques, third edition, Wiley, 2011.

[11] N. Lu, T. Taylor, W. Jiang, C. Jin, J. Correia, L. Leung and P.C. Wong, "Climate change impacts on residential and commercial loads in the western U.S. Grid", IEEE Transactions on Power Systems, vol. 25, No. 1, pp. 480 – 488, 2010.

[12] O. Quiroga, J. Meléndez, and S. Herraiz, "Fault-pattern discovery in sequences of voltage sag events", 14th International Conference on Harmonics and Quality of Power (ICHQP), 2010, pp. 1 – 6.

[13] D. L. Rudolph, "A systematic approach to the replacement of an aging distribution system", IEEE Industry Applications Magazine, May-June 1988, pp. 32-36.

[14] R. H. Stillman, "Power line maintenance with minimal repair and replacement", Proceedings of the Annual IEEE Reliability and Maintainability Symposium, 2033, pp. 541-545.

[15] S. Yokoyama and A. Askawa ''Experimental Study of Response of Power Distribution Lines to Direct Lightning Hits'', IEEE Transactions on Power Delivery, 1989, Vol. 4, No. 4.

Testing Interoperability Security Policies [*]

Mazen EL Maarabani [1] César Andrés [2] Ana Cavalli [1]

[1] TELECOM & Management SudParis
CNRS UMR Samovar, Evry, France
e-mail: {mazen.el_maarabani, Ana.Cavalli}@it-sudparis.eu
[2] Dpto de Sistemas Informáticos y Computación
Universidad Complutense de Madrid, Spain
e-mail: c.andres@fdi.ucm.es

Abstract

Testing is one of the most widely used techniques to increase the quality and reliability of complex software systems. In this paper we present the notion of testing interoperability security rules in virtual organizations. In particular, we incorporate mechanisms to test those interactions among the organizations of the business communities when the resources are shared. In order to apply our technique to increase the confidence on the correctness of these systems, we need to obtain a set of tests compiling the relevant properties of the interoperability security policies. We present a model based testing approach for checking the correctness of these policies in this environment.

In addition to provide the theoretical framework, we show how this formalism, based on extended finite automata, has been used to test a hospital scenario. This exercise convinced us that a formal approach to test systems can facilitate some of the development phases. In particular, how to choose which tests to apply, is simplified since tests are automatically extracted from the specification.

Keywords: Integrity, Testing, Quality, Interoperability Security Policy.

1. Introduction

Among those areas where the development of Computer Science has changed our society during the last years, the relevance of the collaboration among different information systems is remarkable [2]. In particular, there is a strong demand for access control of distributed shared resources in *Virtual Organizations*, in short VO [8] where the classical notion of client server architecture is obsolete and useless. In particular, a VO is composed of several organizations and their employees, where they share some services or resources among them. Cross-organizational interoperability is a major challenge to VO applications [6]. To be able to specify not only the functional aspect of a VO but also those aspects that guarantee the interoperability security policies is an industrial necessity [12]. Currently, we cannot dissociate the functional aspect of a system from its security consideration. Let us remark that security policies restraint the behavior of a system in order to guarantee a certain level of security. Moreover, it is possible that a security policy adds new behaviors to the system such as obligation actions in the case where these actions are not supported by the system. Therefore, checking only the functional part of a system is not sufficient to guarantee that a system behaves as required and provides the intended services, this process requires the application of *sound techniques*.

Formal methods refer to techniques based on mathematics for the specification, development, and verification of software and hardware systems. The use of formal methods is especially important in reliable systems where, due to safety and security reasons, it is important to ensure that errors are not included during the development process. Formal methods are particularly effective when used early in the development process, at the requirements and specification levels, but can be used for a completely formal development of a system. One of the advantages of using a formal representation of systems is that it allows to rigorously analyze their properties. In particular, it helps to establish the *correctness* of the system with respect to the specification or the fulfillment of a specific set of requirements, to check the semantic *equivalence* of two systems, to analyze the *preference* of a system to another one with respect to a given criterion, to predict the possibility of *incorrect behaviors*, to establish the *performance* level of a system, etc. In this line, formal testing techniques [10] can be used to test the correctness of a system with respect to a specification.

[*] Research partially supported by the ISER project and the TESIS project (TIN2009-14312-C02-01). The work was carried out while the second author was visiting TELECOM SudParis.

It is worth to point out that in recent years, there has been a new emphasis on applying formal testing to check local security policies [3, 11].

It has been argued, usually with very good arguments, both that formal methods are very appropriate to guide the development of systems and that, in practice, they are useless since software development teams are usually not very knowledgeable of formal methods in general, and have no knowledge at all of what academia is currently developing (see, for example, [9, 1] among many others). To solve this, in this paper we present our formal testing methodology [11] in an *informal* way. We present the modeling of the specification, the test architecture, and the test case generation using our experiences obtained from the application of our methodology in real systems. Moreover, we present a complete case study where the net of two hospitals organizations is tested.

The rest of the paper is structured as follows. In Section 2 we present the environment where we apply the testing task. In Section 3 our methodology to test virtual organizations is introduced. Next, in Section 4 a complete case study where our methodology is applied is presented. Finally, in Section 5 we present the conclusions and some lines of future work.

2. Testing Hypothesis in a Organization to Organization Environment

In this Section we present the *Organization to Organization*, in short O2O, environment that we are going to test in order to list a set of assumptions that we can assume during the testing task. We introduce the basic principles of O2O presenting our running example. This example is depicted in Figure 1. There are represented two organizations A and B that might share resources. Each organization has its *local set of security policies* that defines the roles and the access for its employees. This fact is graphically represented by "orgA" and "orgB" respectively. When there is an exchange of information between different organizations, the notion of local policies must be extended.

According to O2O, each organization defines its own Virtual Private Organization, in short VPO. A VPO is associated with an interoperability security policy that administrate the access of external users to the organization resources. In Figure 1 we represent these VPOs as "$\texttt{VPO}_{A2B}$" and "$\texttt{VPO}_{B2A}$" respectively. In particular, $\texttt{VPO}_{A2B}$ is associated with a security policy that manages how employees from organization A, called O-grantee in O2O, may have an access to the resources of organization B, called O-grantor in O2O.

An important notion in O2O is the concept of *authority sphere*. This sphere restricts the scope of every security rule to the organization in which the rule is applied. Each organization has its own sphere, as we can observe in Figure 1.

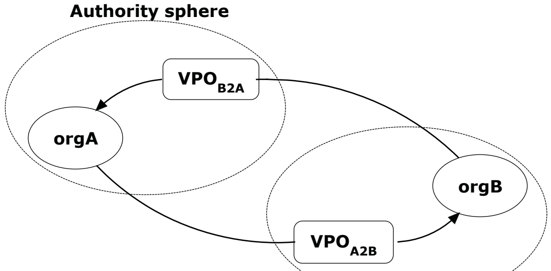


Figure 1. Interaction in O2O.

Thus, in this paper we focus in testing each authority sphere. In each authority sphere O2O uses context [7] to express different types of extra conditions or constraints that control the activation of rules expressed in the access control policy. There are several types of context defined for O2O. For instance the temporal context that depends on the time at which the subject is requesting for an access to the system or the special context that depends on the subject location.

Taking into account the information provided by the contexts of the security rules, we can assume the following testing hypothesis. We are provided with a global clock to check the temporal contexts. This fact allow us to describe the scenarios for the tests using time restrictions in an easy way. We can monitor into logs at most this information: the software and the hardware architecture, the subject purpose, the system database, and the historical interactions.

Each organization that belongs to a net O2O has associated a set of security policies for local employees and several VPOs, one for each organization that belongs to the net. Finally, the interoperability security policies of the VPOs are considered provided by experts in a formal way. Moreover, security policy properties such as *completeness* and *consistency* are considered verified. In particular, we say that a security policy is complete if it computes at least one decision for every incoming request, and it is consistent if for any request the policy computes at most one decision.

3. Testing Approach

In this Section we present our testing approach. Our main goal is to generate test cases to stimulate the O-grantee to send specific requests to the O-grantor. These requests target the security rules to be tested. We assume that we have only access to the O-grantee system. For each request sent from the O-grantor one and only one security rule can be involved[1]. After applying the test case, we check that the set of received outputs conforms with the test case

[1] Let us remark that the interoperability security policy is consistent and decision complete.

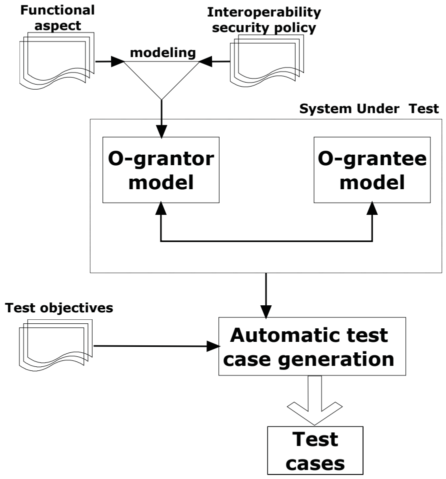


Figure 2. Overview of the Test Generation Framework.

only if the conditions of context are satisfied. Following we present an overview of our tests suite generation framework for O2O. Graphically, it is represented in Figure 2. The process of "Automatic test case generation" receives two inputs parameters. On the one hand the specification of the system by using a formal model based on Extended Finite Automata. This model represent the joint behavior of the two systems (the O-grantor and the O-grantee). In particular, the O-grantor model describes the functional aspect of the system and the interoperability security policy [11]. On the other hand the test objectives specifying the security properties to be tested. Regarding the hole approach, these are the main concepts that allow us to describe it in detail:

1. *The test architecture*,
2. *the systems behaviors using a formal specification language*,
3. *the test objectives and the test case generation*, and
4. *the test execution and test verdict*.

Let us focus in the first topic 1, by means the test architecture. Our system consists of two systems: a system that represents the O-grantee and the other one that represents the O-grantor. On the one hand, the access is done by the users of the O-grantee so the unique control point should be placed at the user side of the O-grantee that initializes the interaction between the two systems. On the other hand we have a set of compulsory minimum set of points of observation to attach to the architecture, but we are allowed to define higher number of points of observations where the exchanged data can be collected at runtime. Note that in our approach it is necessary to attach the following points of observations: those points that allow us to collect the interactions of the O-grantor with its information system which in charge of managing the security policies contexts, and those interactions between the O-grantor and the O-grantee when the information of a context are managed by the O-grantee.

Now, let us focus on 2, that is the systems behaviors using a formal specification language process. The description shows the behavior of the communicating system by taking in consideration the test architecture, the interoperability security policy, and the different actions that intervene in the correct behaviors of the service. In our approach each system behavior is modeled with an Extended Finite Automaton in IF language. The extended automaton that covers the joint-behavior of the two systems is built from the automata of these systems.

Next we present the following concept, by means 3: the test objectives and test generation. This task is based on selecting a set of tests according to a given criteria. These criteria are defined with respect to the security requirements to be tested. We formally define a test objective as a list of ordered conditions. This means that the conditions have to be satisfied in the order that is given by the test objective. The test case generation is guided by the set of these test objectives. The inputs of the generated test case stimulate the system O-grantee that sends specific requests to the system O-grantor. Whereas, the outputs of the test cases represent the expected behavior of the system when the security rules hold. In this step we have also to check the *activation* of the security rules. We define for each security rule of the interoperability security policy a test objective. Thus, we generate at least for each security rule a test case to check if the O-grantor conforms this rule. The test cases are generated from the extended automaton that covers the joint-behavior of the two systems. When we generate test cases from the extended automata (in 2) it is necessary to use some methodologies based on partial generation of the reachability graph to obtain the tests [4]. In these methods the test case generation process is guided by the test architecture and a predefined set of test objectives.

Finally, to automatize the process of tests generation presented in Figure 2 we use the TestGen-IF tool [5]. This tool is open source and it accepts systems modeled with the IF language. In Figure 3 the architecture of this tool is presented. The tool allows to build tests sequences with high fault coverage and to avoid the state explosion and deadlock problems encountered respectively in exhaustive or exclusively random searches. Let us remind that in the architecture of this tool the test coverage considers a complete test generation if all set of chosen properties are specified in the test objectives.

Following we present the last notion, by means 4: test execution and test verdict. In order to execute the test cases generated with TestGen-IF in an O2O scenario, it

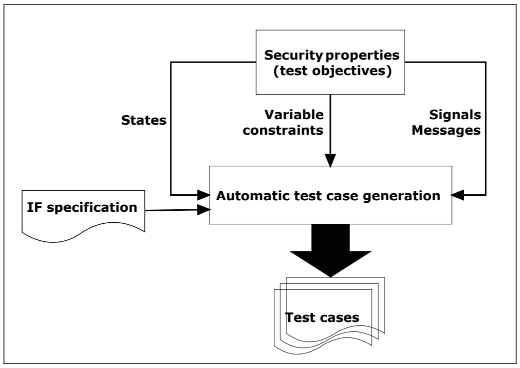


Figure 3. Architecture of TestGen-IF.

is mandatory to *instantiate* them. The instantiation of an abstract test case is composed of two sub-processes: the concretization (addition of details) and the translation to executable scripts. The final test case is a script that contains details of the implementation such as the real values of variables and the signals of the test case. This script will be executed in the O-grantee system to stimulate this system generating specific request to the O-grantor.

Regarding the verdict, when we apply the inputs of the test case on the O-grantor system, the generated set of outputs has to be conform with respect to the test. However, a rule can be only checked when the rule holds. Thus, in the case where the security rule is constrained by a context, an additional information that gives the state of this context is needed in the tests.

The test verdict is built based on the information on the security rule context and the behavior related to the execution of the test case. Following we present the test verdict after executing a test case.

$$\texttt{Verdict} = \neg(\texttt{Test case} \oplus \texttt{context}) \tag{1}$$

Remark that the global verdict, that is the verdict after performing all the test cases, has the form of a conjunction of the verdicts produced for each test case. Therefore if the verdict of one test case is false the global verdict is also false.

4. Case Study

In this Section we present our testing methodology approach through a *hospital network case study*. A hospital network is a network or group of hospitals that work together to coordinate and deliver a broad spectrum of services to their community in an O2O scenario. We considered the case where the hospital network consists of two hospitals, hospital A and hospital B. It was assumed that each hospital had its local security policy to manage the privileges of its local users. We also assumed that the following *roles* existed in the two hospitals: doctor, nurser, aduser (an user in the administrative staff), and ITuser (a user in the information technology staff), and that in both hospitals a patient had a medical report, data related for payment and sensitive data which was related to personal information such as previous medical report, insurance company, etc.

In this case study we only focus on one interaction way: from hospital A to hospital B. Therefore, the resources to be accessed are located in hospital B and the VPO will be $\texttt{VPO}_{A2B}$.

4.1. Modeling the System Under Test

Next we introduce the interoperability security policy objectives that describe the privileges of employees of the hospital A in B and belong to $\texttt{VPO}_{A2B}$. These interoperability security policies represent a set of objectives that hospital B must respect when it is accessed by the employees of hospital A. The complete interoperability security policy is composed of 11 objectives. Due to space limitations following we only describe 3 of them.

The first one focus on the following situation. A doctor from Hospital A has accessed to restricted and sensitive information of a patient of Hospital B after filling a non release form. The second one represents that the system must notify to any doctor when one of his/her patient's medical reports is edited by another doctor. In particular if it is edited by a doctor of a different hospital. Finally, the last one focus on the nurse role. A nurse from hospital A is not allowed to create, drop or edit any data of a medical report.

After defining the objectives, the interoperability security policy objectives are modeled according to the O2O model. As a result we got in this case study 53 different security rules.

Next, both hospitals were modeled within extended automata using the IF language. Note that these models, do not specify the complete behavior of the hospitals but a partial one which is related to the interoperability security policy that we have to test.

We integrated the O2O rules in the automata applying the methodology described in [11]. This integration process increase the number of states and transitions of the original automata. Following we present some metrics of the integration process.

	# States	# Transitions	# Signals
Before int.	3	10	15
After int.	4	12	17

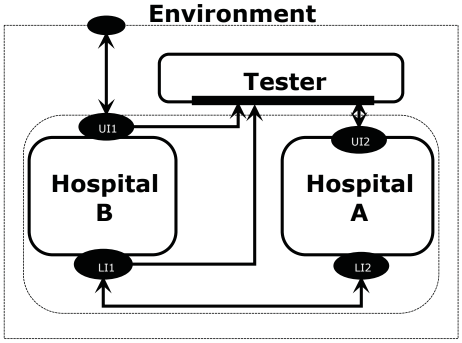


Figure 4. Testing Architecture.

4.2. Test Generation and Results

The system under test consists of the information systems of hospitals A and B. The Figure 4 illustrates the test architecture. As we mentioned before, there is only one control point in the interface between the tester and the information system. It is denoted by "UI2".

Next we generated the test cases using the TestGen-IF tool. The tool performed the following sequence of actions to generate the test cases. It computed a partial accessibility graph of the specification. The partial graph was constructed using a breath first search, where the depth of the partial graph was specified in the tool input parameters. For this running example we applied depth = 100. After this, the tool performed a local search from the current state in a neighborhood of the partial graph. If a state was reached and one or more test objectives were satisfied, the set of test cases was updated and a new partial search was conducted from this state.

Next we defined the test objectives for this net of hospitals. A test objective is described as a list of ordered conditions. This means that the conditions have to be satisfied in a given order. A condition is a conjunction of: a process instance constraint, a state constraint, an action[2] constraint and a variable constraint. In particular, a process instance constraint indicates the identifier of a process instance, and and action constraint describes how to send or receive the messages.

In our case study the test case generation algorithm implemented in the tool checked whether the properties in the reachability graph that described the joint-behavior of the two systems held. When the property was satisfied, the algorithm generated the path for this property. This path was transformed to an executable test case to be applied later to the system implementation.

Let us remark that the use of this tool is friendly. For instance, Figure 5 presents a test objective and its test case, generated by the tool. The test objective describes that "a doctor (with an $\text{id} = 6$) of the hospital A is allowed to access sensitive information of a patient (with an $\text{id} = 65$) of hospital B". The generated test case shows that the tester has to send two inputs. A first input, namely signe_NDF_REQ$\{6,55\}$, is to sign a non divulgation form. The second input, namely access_file_REQ$\{6,65\}$, is to check that if this doctor request to access the sensitive information the hospital B will grant the access. Following we report some relevant data of the use of this tool after performing this task.

Objective

$$\begin{array}{lcc}
tp & \leftarrow & \left\{ \bigwedge_{1\leq i\leq 5} c_i \right\} \\
c_1 & \leftarrow & (process : instance == \{hospitalB\}) \\
c_2 & \leftarrow & (state : source == \text{"}sign\text{"}) \\
c_3 & \leftarrow & (state : target == \text{"}idle\text{"}) \\
c_4 & \leftarrow & (action : input == \text{"}access\_file\{6,65\}\text{"}) \\
c_5 & \leftarrow & (action : output == \text{"}access\_grant\text{"})
\end{array}$$

Test Case

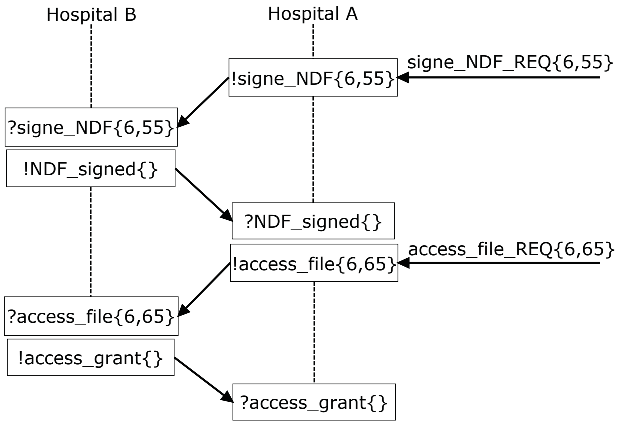


Figure 5. Example of objective and test case in TestGen-IF.

[2] In this context an action represents any input or any output.

Objectives	Partial graph generation	Depth limit	Visited states	Time to generate the test cases
53	BFS	100	12855	28 min and 2 s

The next step in the testing process was the test case execution. It was performed on a web based prototype implementation and we used three different tools to get the results:

1. The execution of the test cases was performed by using the ACS-Automated-testing tool (`http://openacs.org/xowiki/acs-automated-testing`).

2. The tclwebtest (`http://tclwebtest.sourceforge.net/`) framework to describe the executable test cases for web applications.

3. The Wireshark (`http://www.wireshark.org/`) tool was also used to capture the interaction of the hospitals within their information systems.

After the execution of each test case we extracted from the collected traces the contexts of the hospitals. Within this set of contexts information we verified if the security rule presented in the test case held. According to the equation 1 we said that the verdict after executing this test case was positive: a) if rule held in this context and the verdict provided by ACS-Automated-testing was positive, or b) if the rule did not hold in this context and the verdict provided by the ACS-Automated-testing tool was negative. As a result of this experiment we got with a positive verdict 48 test cases meanwhile we got with a false verdict for 5 test cases. For instance, one of the detected problems was that the system of an hospital did not notify to the doctors of the other hospital when a shared report was modified.

5. Conclusion and Future Work

In this paper we have presented our formal methodology to check interoperability security policies in an informal way. We have tried to describe our experience while testing complex environments by using our formal approach. In this paper we present our specification formalism. In particular, we focus on representing the interoperability security policies of these systems. Next, we show how we can automatically extract a finite set of tests from the specification and how these tests are run again the system under test in order to get a verdict.

In addition with our testing framework, in this paper we have shown a case study in a *controlled environment* where the interoperability security rules were checked. We consider that the use of this testing methodology is very positive and encouraging as to support the use of formal methods.

As future work we would like to add some extra (probabilistic) information in our tests in order to increase their coverage in a virtual organization environment.

References

[1] J. Bowen and M. G. Hinchey. Ten commandments of formal methods ... Ten years later. *Computer*, 39(1):40–48, 2006.

[2] J. Cao, J. Chen, H. Zhao, and M. Li. A policy-based authorization model for workflow-enabled dynamic process management. *Journal of Network and Computer Applications*, 32(2):412–422, 2009.

[3] A. Cavalli, A. Benameur, W. Mallouli, and K. Li. A passive testing approach for security checking and its practical usage for web services monitoring. In *9th Conf. Int. sur Les NOuvelles TEchnologies de la REpartition, NOTERE 2009*. ACM, 2009.

[4] A. Cavalli, D. Lee, C. Rinderknecht, and F. Zaïdi. Hit-or-Jump: An algorithm for embedded testing with applications to IN services. In *Formal Description Techniques for Distributed Systems and Communication Protocols (XII), and Protocol Specification, Testing, and Verification (XIX)*, pages 41–56. Kluwer Academic Publishers, 1999.

[5] A. Cavalli, E. M. D. Oca, W. Mallouli, and M. Lallali. Two complementary tools for the formal testing of distributed systems with time constraints. In *12th IEEE/ACM Int. Symposium on Distributed Simulation and Real-Time Applications, DS-RT'08*, pages 315–318. IEEE Computer Society, 2008.

[6] C. Coma, N. Cuppens-Boulahia, F. Cuppens, and A. Cavalli. Interoperability of context based system policies using O2O contract. In *4th Int. Conf. on Signal-Image Technology & Internet-based Systems, SITIS'08*, pages 137–144. IEEE Computer Society, 2008.

[7] F. Cuppens and N. Cuppens-Boulahia. Modeling contextual security policies. *International Journal of Information Security*, 7(4):285–305, 2008.

[8] U. Franke. *Managing virtual web organizations in the 21st century*. IGI Publishing, 2002.

[9] M. Gogolla. Benefits and problems of formal methods. In *9th Ada-Europe Int. Conf. on Reliable Software Technologies, Ada-Europe'04, LNCS 3063*, pages 1–15. Springer, 2004.

[10] R. M. Hierons, K. Bogdanov, J. Bowen, R. Cleaveland, J. Derrick, J. Dick, M. Gheorghe, M. Harman, K. Kapoor, P. Krause, G. Luettgen, A. Simons, S. Vilkomir, M. Woodward, and H. Zedan. Using formal methods to support testing. *ACM Computing Surveys*, 41(2), 2009.

[11] M. E. Maarabani, I. Hwang, and A. Cavalli. A formal approach for interoperability testing of security rules. In *6th Int. Conf. on Signal-Image Technology & Internet-based Systems, SITIS'10*, pages 277–284. IEEE Computer Society, 2010.

[12] R. Salay and J. Mylopoulos. The model role level: a vision. In *29th international conference on Conceptual modeling, ER'10*, pages 76–89. Springer, 2010.

A New Approach to Evaluate Path Feasibility and Coverage Ratio of EFSM Based on Multi-objective Optimization*

Rui Yang[1,2], Zhenyu Chen[1], Baowen Xu[1,2+] , Zhiyi Zhang[1] and Wujie Zhou[3]

[1] State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China
[2] Computer Science and Technology, Nanjing University, Nanjing, China
[3] School of Computer Science and Engineering, Southeast University, Nanjing, China
[+] Corresponding author: bwxu@nju.edu.cn

Abstract—**Extended Finite State Machine (EFSM) is a popular model for software testing. Automated test generation on EFSM models is still far from mature. One of the challenges is test case generation for a given path and the path may be infeasible. This paper proposes a novel approach to solve the path ordering problem by the Multi-objective Pareto optimization technique. Two fitness functions are designed to obtain the Pareto-optimal solutions of path sequence, such that test cases could be generated more effectively. The optimization process is to find the path set, with trade-off path length, coverage criterion and path feasibility. An experiment was designed with four popular EFSM models. The experimental results show that our approach is more effective by comparing it to previous techniques.**

***Keywords-** path feasibility evaluation, Multi-objective optimization, test case generation, EFSM model-based testing*

I. INTRODUCTION

Automated testing usually derives the test case via creating a model of the software which represents the real system and employs the model to generate test case that can be applied to the system implementation. The generated test cases are utilized to verify whether the implementation meets the specification.

In model-driven testing, Finite State Machine (FSM) and Extended Finite State Machine (EFSM) are widely used for the purpose of generating test case. There are several issues that limit the application of FSM-based testing. The main issue is that FSM can only represent the control part of a system. However, Many complex systems include not only control parts but also data parts. Extended Finite State Machine (EFSM),which consists of states, variables and transitions among states can express both control flow and data flow of system, can be considered as an enhanced model of FSM [1]. T here have been lots of research works on FSM-based test sequence generation, whereas test case generation for the EFSM model is still a challenge .

In general, the steps of automated test case generation from an EFSM model including: (1)Generate state identification sequence. (2)Generate test paths for specified coverage criterion. (3)Generate test data to trigger the test paths. (4)Create test oracle.

The difficulty of automated testing on EFSM models since the fact that the EFSM models may contain infeasible paths due to the existence of conflict between predicates and assignment statements in transitions. Moreover, the detection of an infeasible path is generally undecidable [2].

In terms of state identification sequence generation, many methods have already been proposed in the literature [3][4][5]. There also exist a few studies address the problem of infeasible path even test data generation of EFSM models[6][7][8][9][10]. However, automated EFSM-based testing is still far from mature. In our previous work[11], an approach that combines static analysis and dynamic analysis techniques is presented to address the problem of path infeasibility in the test case generation process on EFSM model. A metric is presented in order to find a path set that has fewer paths, longer path length and goodness feasibility to meet specified coverage criterion since previous study shown that the test suite with small number of longer test cases improves fault-detection efficiency compared to shorter ones[12]. This metric is designed carefully to trade-off path length and feasible evaluation value, since a longer path has higher probability of infeasible due to the fact that a longer path may contains more conflict conditions among transitions. However, the formula is still not precise enough.

Therefore, this paper presents a new approach that overcomes aforementioned problems by Multi-objective Pareto optimization technique to explore the relationship between path length and path feasibility. Multi-objective optimization is appropriate for solving problems when the solutions need to meet several conflicting objectives. By using this technique, we transform the aforementioned problem into a bi-objective optimization problem. The optimization process is given to find a path set to achieve the trade-off between path length and path feasibility to meet the specified test criterion(longer paths may contain more transitions to achieve specified coverage criterion). In addition, we present detailed experimental study to evaluate the efficiency by applying our approach to four popular EFSM models.

The main contributions of this article are embodied in the following three aspects:

1. A new Multi-objective approach is presented to find a optimized path set with small number of longer feasible paths to generate test data and meet the test criterion.
2. We present two fitness functions of an individual to solve the path ordering problem of EFSM models. The idea also can be extended to other path ordering problems in path-oriented testing.
3. An experiment is designed and we apply the proposed approach to four popular EFSM models. The experimental results confirm that the approach is more effective by comparing it to our previous metric.

The rest of this paper is organized as follows: Section II introduces the basic concept of an EFSM model; Section III

*The work described in this article was partially supported by the National Natural Science Foundation of China (90818027, 61003024, 61170067).

introduces our previous research of test case generation process; The experiment is described in Section IV; Section V reviews related work; Section VI concludes this paper and provides some possible opportunities for future research.

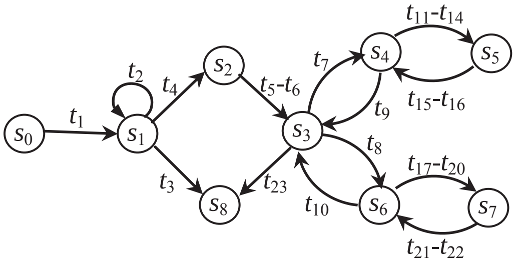


Figure 1. EFSM model of Automated Teller Machine [15]

II. Preliminaries

An EFSM can be formalized as a 6-tuple M=(S, s_0, V, I, O, T), where S represents a finite set of state; $s_0 \in S$ represents the initial state of the EFSM; V represents a finite set of context variables; T represents finite set of transition; I represents a set of inputs of transition and O represents a set of outputs. Each transition $t \in T$ is also a 6-tuple T= (s_i, s_j, P_t, A_t, i_t, o_t), where s_i represents the start state of transition t; s_j represents the end state of transition t; P_t represents the predicate operation on context variables(known as Guard) and A_t represents the operation on current variables(known as Action); $i_t \in I$ represents an input parameter and $o_t \in O$ represents output results.

At first, the EFSM model is at an initial state s_0 with vector x_0 of initial variable values. After some transitions occurred, the EFSM move to the state s_i with the current vector x_i of variable values. If EFSM model receiving input event or input parameter i_t, and x_i is valid for predicate operation P_t, then the transition t will be triggered and state of EFSM moves to s_j. A self-loop transition is a transition that has the same start state and the end state (see **Figure 1**). When a transition is triggered, action A_t changes the current variable values, and output events or output results may also be produced. Some transitions are difficult to be satisfied since it may have complex conditions, while some transitions may have no predicate conditions. A transition path is infeasible if associated predicate conditions of the transition never be satisfied. The existence of infeasible paths creates difficulties for automated test case generation for EFSM. In addition, some models are free of exit state, while others contain both initial state and exit state.

If any group of transition has the same input that transforms a state, and the guards of more than one transition cannot be satisfied at the same time in this transition group, then the EFSM is deterministic[13]. Otherwise the EFSM model is non-deterministic. In this paper, we assume that the initial state of EFSM is always reachable from any state through Reset operation, and only deterministic EFSM model is considered.

III. Implementation

3.1 Test Case Generation Process of Previous Research

For the sake of clarity, we first describe our previous work[11][29], the process of our approach to generate test cases(including feasible paths, test data and oracle information) from an EFSM can be split into following steps:

Step1: Candidate Path Set Generation Algorithm

Firstly, a candidate path set is generated from the initial state to other states on an EFSM model in order to satisfy the test adequacy criterion. Then, some paths which have highest likelihood of feasibility are selected to generate test data. The transition paths in candidate path set contain loops (including self-loops). The generated paths would infinite if the infinite loops are contained in transition paths. Therefore, a constraint condition that just contains loops or self-loops only one time is imported to generate paths by means of candidate path set generation algorithm named *Path-Gen*(detail described in [11]). The paths that contain a certain number of loops (self-loops) can be also generated conveniently by expanding aforementioned algorithm.

Step2: Feasibility Evaluation Strategy

The existence of infeasible paths on EFSM model, which is due to the variable interdependencies among the actions and conditions, has become a challenging problem of path-oriented test case generation. In order to evaluate the feasibility of paths that generated in the *Step1*, a metric is presented through static analysis to evaluate the path feasibility of the EFSM and achieve all-transitions coverage criterion. Afterwards, paths in the candidate set are sorted by the evaluation value to assist test data generation in *Step3*. However, in some cases, static analysis cannot detect the infeasible path thoroughly. Therefore, we present a dynamic analysis method in order to detect infeasible paths in the test data generation process. In order to find a path subset with the property that has fewer paths, longer path length and goodness feasibility to achieve all-transitions criterion, the metric is presented as follows:

$$f = \begin{cases} \dfrac{\sum_{i=0}^{k} v(df_i)}{|TP|^d} & If \quad \sum_{i=0}^{k} v(df_i) \neq 0 \\ 0 - |TP| & If \quad \sum_{i=0}^{k} v(df_i) = 0 \end{cases} \qquad (1)$$

where k represents the number of definition-p-use transition pairs in a path; df_i represents the definition-p-use transition pair in a path and $v(df_i)$ represents its corresponding penalty value; |TP| represents the path length, and d is a value used to tune the weight of path length in the metric(detail described in [11]). Denominator |TP| is the trade-off between path length and penalty value since a longer path may contains more definition-p-use transition pairs, which has higher probability that it can result in a larger penalty value of a transition path. However, if

$\sum_{i=0}^{k} v(df_i) = 0$, it means that there is no definition-p-use interdependence among transitions, and this path has best feasible probability. In this case, the penalty value is assigned $0-|TP|$ due to a longer path should get a lower penalty value to cover more transitions. Thereafter, the transition path set is sorted by f in ascending order. However, the final feasible path subset which attempt to achieve all-transitions criterion or other coverage criteria will be generated in the test data generation process dynamically.

Step3: Test Data Generation Process

In this step, a dynamic analysis method, which is based on meta-heuristic search algorithm, is proposed to detect infeasible paths in the test data generation process. In general, a common strategy of dynamic analysis is to limit the depth or iterations of search. We build an executable model by expressions semantic analysis technique, therefore the dynamic run-time feedback information can be collected as a fitness function and scatter search algorithm is introduced to guide the test data generation process directly. In addition, the corresponding expected outputs of the generated test data are also calculated to construct the oracle information automatically. Finally, feasible path subset, test data and oracle information are combined into complete test cases.

3.2 Multi-objective Optimization Approach

The formula 1 is designed carefully to trade-off path length and feasible evaluation value, since a longer path has higher probability that it contains more conflict Guards and Actions which can result in a path infeasible. However, the formula 1 is still not precise enough. Therefore, in this study, the multi-objective optimization approach is presented to find a path set which achieves the trade-off between path length and path feasibility to meet the specified coverage criterion(longer paths may contain more transitions to achieve specified coverage criterion). In brief, we present a Multi-objective Optimization approach to improve the path feasibility evaluation strategy.

3.2.1 Solution Encoding and Fitness Functions Design

The important issues of multi-objective optimization problem are the encoding of the solution and the design of the fitness function. Before a Multi-objective optimization algorithm can be applied to solve the problem, we need to encode potential solutions to that problem in a certain form which can be processed. In addition, the fitness function has an important influence on finding the final solution to meet the requirements. The fitness value of a solution is evaluated by the fitness function. A solution with better fitness value is selected for next iteration in the Pareto evolution process. In context of our approach, the fitness values is the measurement that whether a path set with certain path ordering can facilitate test data and oracle information generation.

In this study, we use the permutation encoding method, therefore, the paths in a set are encoded as a sequence of integer. In permutation encoding, the individuals in the population are the string of numbers. Thus, for a given transition path set, every transition path is encoded as a different number(named path ID). Please note that, when using permutation encoding, the evolution process cannot generate any new number in the individual but change its ordering. In addition, the property of a transition path, such as path length and count of distinct transitions, can be obtain by its ID. The individual structure is described in **Figure 2**.

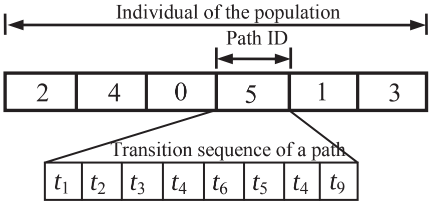


Figure 2. Individual structure of a population

For ordering the path set that generated by candidate path set generation algorithm to obtain Pareto front, we designed two fitness functions f_1 and f_2, where f_1 is utilized to find the path ordering to obtain a path subset with small number of longer paths and higher coverage ratio, and f_2 is utilized to find the path ordering to obtain a feasible path subset more efficiently in dynamic process(Step 3 in section 3.1). the fitness functions are designed as follows:

$$f_1 = \sum_{j=0}^{n-1} \frac{C_1}{(\frac{C_2}{\gamma}+1)\times(n-j+1)} \tag{2}$$

$$f_2 = \sum_{j=0}^{n-1} \frac{C_1}{(\sum_{i=0}^{k} v(df_i)+1)\times(n-j+1)} \tag{3}$$

where n is the count of the paths, j represents the position of a path in path set, γ is the count of the distinct transitions in a path, $\sum_{i=0}^{k} v(df_i)$ is same as formula 1. Number C_1 and C_2 as numerator are constant value since smaller fitness value means better solution in selection process. In our study, C_1 and C_2 are assigned 1000 and 100, respectively. In general, according to the fitness function f_1, the longer paths with more distinct transitions will be arranged in the front of path set, this kind of path ordering has better fitness. In terms of f_2, the paths with better feasible probability will be arranged in the front of path set, this kind of path ordering has better fitness.

3.2.2 Multi-objective GA Algorithm

For implementing our method, the non-dominated sorting genetic algorithm II (NSGA-II) presented by Deb et al. [14] is utilized. NSGA-II uses an elitist approach and a crowed comparison mechanism, improving both quality and diversity of Pareto solutions. Hence, in our case, NSGA-II is used to solve the problem of multi-objective optimization.

In every generation, NSGA-II utilizes crossover and mutation to create a new population. The next generation is given by a Pareto-optimal selection from both the new offspring and their parents. Therefore, NSGA-II is elitist. In this study, NSGA-II was given two fitness function:f_1 and f_2.

At the beginning, a population P_0 is created by means of stochastic method. The population is sorted based on the

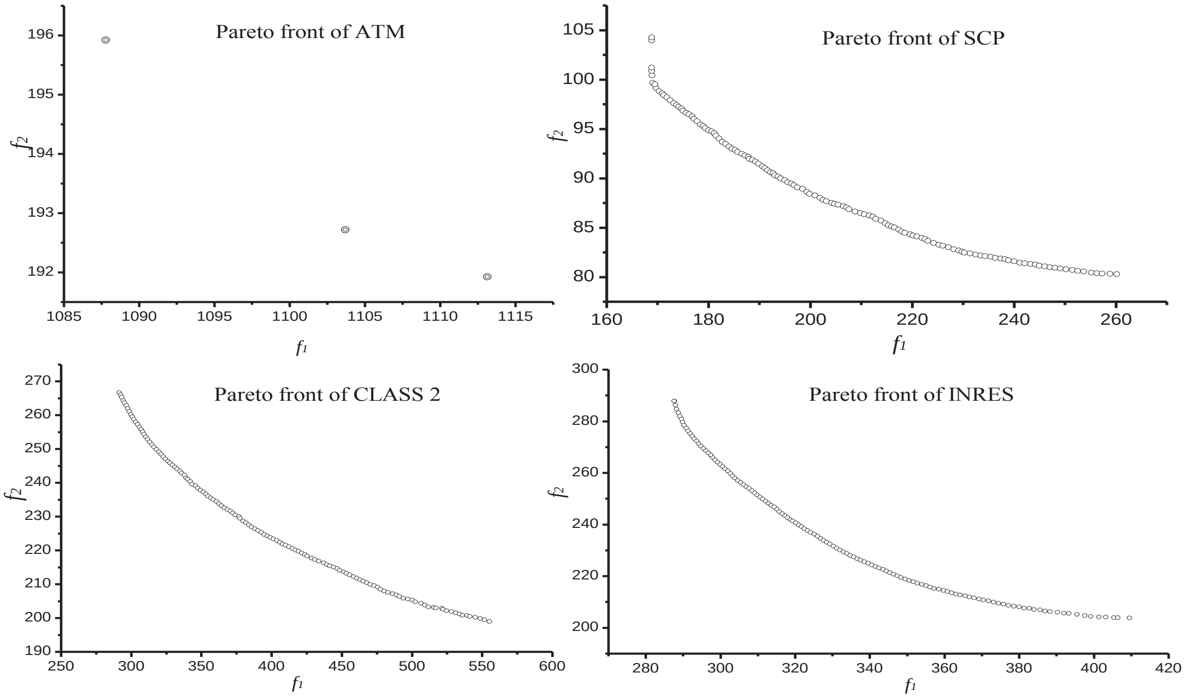


Figure 3. The front of Pareto–optimal solutions of four EFSM models

non-domination. To fit the permutation encoding method, we adopt the BinaryTournament selection, TwoPointsCrossover and SwapMutation operators to create a offspring population Q_0 (size is N).

After that, the algorithm enters into iteration process: first the population $R_t = P_t \cup Q_t$ (R_t size is $2N$) is formed by combination method. Afterwards, the population of R_t is sorted according to non-domination, and the ranks are assigned to each Pareto front with fitness F_i. Then, from Pareto front of fitness F_1, each Pareto front is added to new population P_{t+1} until it reaches the size N. The new population P_{t+1} will be used for selection, crossover and mutation to create the new population Q_{t+1}. The above process is repeated until all individuals are assigned to a pare to front.

IV. Experimental Study

In this section, in order to determine whether test case generation with specified coverage criterion for an EFSM can benefit from proposed multi-objective optimization technique, we present a detailed experimental study. More specifically, the study is designed to get the final Pareto fronts of the transition path ordering of the EFSM models. Thereafter, Pareto-optimal solution is used to test data generation process to find whether this approach is more effective than our previous metric.

Four popular EFSM models are utilized for our experimental study: Class 2 transport protocol[3], Automated Teller Machine (ATM)[15], INRES protocol[16], and SCP protocol[17]. Class 2, INRES and SCP model are free of exit states while ATM contains both a start and exit state. For generality, in our experiment, we do not considered that the path must end in an exit state in the model with exit state.

4.1 Experimental Setup

In order to achieve the experiment fairly, the experiment environment and the main configurations of test data generation algorithm are same as our previous work [11].

The main parameters configuration of NSGA-II algorithm we set in this study are listed as follows:

1. population size =128;
2. Max Evaluations times=20000;
3. Crossover Operator= TwoPointsCrossover;
4. Crossover probability=0.9;
5. Mutation Operator= SwapMutation;
6. Mutation probability=0.2;

Since permutation encoding method is utilized in this study to solve path ordering problem. The individuals in the population are the string of numbers that represent a path ID in a sequence. When crossover and mutation operation are used to create the new population, the operation cannot generate any new path ID in the individual but change its ordering. That is to say, the elements of a individual remain unchanged, crossover and mutation operation only change the ordering of the elements of an individual. Therefore, special crossover and mutation operator are needed. In this study, TwoPointsCrossover operator and SwapMutation operator are adopted to achieve permutation encoding method. Crossover probability is 0. 9 where Mutation probability is 0.2. The population size for evolution is 128. While generation count reached Max iteration times, NSGA-II algorithm stopped.

TABLE I. TEST GENERATION RESULTS WITH DIFFERENT TECHNIQUES

Model & Method	Subject Model	Executed Path Number	FTP Number	Iteration Times	Execute Time(sec.)
Feasibility Evaluation Using Formula 1	ATM	17	16	1600	4
	CLASS 2	30	16	2016	57
	INRES	10	10	7650	2
	SCP	10	2	816	13
Multi-objective Optimization	ATM	15	14	1400	4
	CLASS 2	26	12	3456	58
	INRES	11	9	20936	9
	SCP	10	2	1126	14

4.1 Experimental Results

In this section, the experimental results of the path ordering problem on four EFSM models are presented. **Figure 3** shows the front of Pareto–optimal solutions of four EFSM models. In the figure, Y-axis represents the fitness value of the path sequence that calculated by fitness function f_2. The number of X-axis represents the fitness value of the path sequence that calculated by fitness function f_1. That is to say, a dot in the coordinate plane represents a kind of path sequence that has fitness value f_1 and f_2. The count of dot less than or equals to 128 since population size equals 128. The special case in the figure is that the dot of ATM model relative few as a result of this model has relative simple transition correlation, also the Pareto-optimal solutions is relative few.

In order to evaluate the efficiency of our approach, we need to choose a solution in Pareto–optimal front(an ordering path set), this path set is used to generate test data, meanwhile, a subset of feasible path which achieves all-transitions coverage strategies will be generated dynamically in this process. In the experiment, the solution with minimum value of f_1+f_2 is selected to generate test data.

Since the execution time of the test data generation of the same model exists tiny difference, the program is executed 30 times for every model and calculate its average execution time for the purpose of illustrating the experimental results precisely. In order to illustrate the advantage of this approach, the results of test data generation are used to compare with previous results in the[11]. The comparison results are listed in **Table I**, where Executed Path Number is the count of transition path that be chosen to generate test data. FTP Number is the feasible transition path number that are selected for test data generation to achieve all-transitions coverage. Iteration Times is the sum of the Iteration Times of every feasible path. It shows that the final feasible path number(FTP Number) reduced obviously compared with previous technique.

In the mass, the count of transition path that be chosen to generate test data(executed path number) also decreases. More infeasible paths are avoided in the process of test data generation(the value of Executed Path Number minus FTP Number of a model means the count of infeasible path that encountered in the process of test data generation). In the other hand, Iteration Times of test data generation increased in some degree, Execute Time is the same as Iteration Times. The reason is that the feasible transition paths become longer and more difficult to generate test data to traverse these paths. Through observing, we found that the longer path contains more correlation among transitions, and test data generation process becomes more difficult, even the path becomes infeasible. However, we think that the quality of test paths and test data is more important than iteration times. Hence, this approach is more effective by comparing it to previous metric.

V. RELATED WORK

In terms of EFSM-based testing, some test sequence generation methods of EFSM models have been proposed in [18][19]. In these methods, test sequence generation with data flow criteria is considered, and control flow testing is ignored or considered separately. Duale et al.[7][21] transformed an EFSM to one that has no infeasible paths, the approach converted a EFSM into consistent EFSM, and they utilized simplex algorithm to solve the infeasible problem. However, this method only can be utilized for EFSMs in which all operations and guards are linear. In addition, there are some other methods used FSM-based techniques to test an EFSM model[16][20], however, these methods may suffer the state explosion problem. In order to generate feasible test paths from EFSM models, Derderian et al.[8] gave a fitness function to estimate how easy it is to trigger a path. Kalaji et al.[9] proposed a GA-based approach to generate feasible transition paths that are easy to trigger. However, the path length should be determined in advance and all paths have same length. In terms of test data generation on EFSM, Lefticaru et al.[22] first introduced the genetic algorithm to derive test data from a FSM. The design of fitness function is based on literature [23]. Yano et al.[24] presented a multi-objective evolutionary approach for test sequence generation from an EFSM. However, the generated test data may have redundancy. Kalaji et al.[25] used a genetic algorithm whose fitness function is based on a combination of a branch distance function[23] and approach level [26] to generate test data for specified path. J. Zhang et al.[27] proposed a method to obtain a deterministic EFSM from a program written in a subset of C program, and they attempted to find test data by means of the symbolic execution technique. Ngo et al.[28] proposed a heuristics-based approach for code-based testing to detect infeasible path for dynamic test data generation by the observation of some common properties of program paths. J. Zhang et al.[29]presented the detail of test data generation from EFSM by using scatter search and execute information feedback technique. The method showed good effectiveness by the experiments of comparing with the random algorithm.

VI. CONCLUSIONS AND FUTURE WORK

This paper presents a new approach that utilizes Multi-objective Pareto optimization technique to explore the relationship between path length and path feasibility on EFSM models. Two fitness functions of an individual are designed to solve the path ordering problem. The optimization process is given to find a path set which achieves the trade-off between path length and path feasibility to meet the specified test criterion. This idea also can be extended to other path ordering problem in path-oriented testing. An experiment was designed with four popular EFSM models. The experimental results show that the approach is more effective by comparing it to previous metric.

In our future study, we intend to improve design of the fitness function to get better quality of test case by further exploring the property in the transition path. The tuning of the parameters and operators of multi-objective optimization algorithm to get higher efficiency also will be implemented.

VII. ACKNOWLEDGMENTS

The work described in this article was partially supported by the National Natural Science Foundation of China (90818027, 61003024, 61170067). The authors would like to thank anonymous reviewers for their valuable comments.

REFERENCE

[1] A. Petrenko, S. Boroday and R. Groz, “Confirming configurations in EFSM,” FORTE, pp.5-24, 1999.

[2] D. Hedley and M. A. Hennell, “The Causes and Effects of Infeasible Paths in Computer Programs,” ICSE, pp. 259-266, 1985.

[3] T. Ramalingom, K. Thulasiraman, and A. Das, “Context independent unique state identification sequences for testing communication protocols modelled as extended finite state machines,” Computer Communications, Vol. 26(14), pp. 1622-1633, Sep 2003.

[4]C.M. Huang, M.Chiang, and M.Y.Jang. “UIO_E: A protocol test sequence generation method using the transition executability analysis (TEA), ” Computer Communications, vol.21(16), pp.1462-1475, 1998.

[5] A. Petrenko, S. Boroday and R. Groz, “Confirming configurations in EFSM testing,” IEEE Transactions on Software Engineering, Vol.30(1), pp.29-42, 2004.

[6] S. T. Chanson and J. Zhu, “A unified approach to protocol test sequence generation,” In Proceedings of the International Conference on Computer Communications , IEEE INFOCOM , pp. 106-114, 1993.

[7] A. Y. Duale and M. Ü. Uyar, “A Method Enabling Feasible Conformance Test Sequence Generation for EFSM Models,” IEEE Transactions on Computers, Vol. 53(5), pp. 614-627, 2004.

[8] K. Derderian, R. M. Hierons, M. Harman, and Q. Guo, “Estimating the feasibility of transition paths in extended finite state machines,” Automated Software Engineering, Vol. 17(1), pp. 33-56, Nov 2009.

[9]A. S. Kalaji, R. M. Hierons, and S. Swift, “Generating Feasible Transition Paths for Testing from an Extended Finite State Machine (EFSM),” ICST, pp. 230-239, Apr 2009.

[10]T. Yano, E.Martins, and F.L.Sousa. “Generating Feasible Test Paths from an Executable Model Using a Multi -objective Approach,” ICST Workshops, pp.236-239, 2010.

[11]R. Yang, Z. Chen, B. Xu, W. Eric Wong, and J. Zhang “Improve the Effectiveness of Test Case Generation on EFSM via Automatic Path Feasibility Analysis,” IEEE HASE, pp.17-24, 2011.

[12]G. Fraser and F. Wotawa. “Redundancy Based Test-Suite Reduction,” Lecture Notes in Computer Science, 4422, pp. 291-305, 2007.

[13] C. Shih, J. Huang, and J. Jou, “Stimulus generation for interface protocol verification using the non-deterministic extended finite state machine model,” IEEE HLDVT, pp. 87-93, 2005.

[14]K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, “A fast and elitist multiobjective genetic algorithm: NSGA-II,” IEEE Transactions on Evolutionary Computation, Vol. 6(2), pp. 182–197, 2002.

[15]B. Korel, I. Singh, L. Tahat, and B. Vaysburg, “Slicing of state-based models,” ICSM, pp. 34-43, 2003.

[16]C. M. Huang, M. Y. Jang, and Y. C. Lin, “Executable EFSM-based data flow and control flow protocol test sequence generation using reachability analysis,” Journal of the Chinese Institute of Engineers, Vol. 22(5), pp. 593-615, Jul 1999.

[17]A. Cavalli, C. Gervy, and S. Prokopenko, “New approaches for passive testing using an Extended Finite State Machine specification,” Information and Software Technology, Vol. 45(12), pp. 837-852, Sep 2003.

[18]H. Ural and B. Yang, “A Test Sequence Selection Method for Protocol Testing,” IEEE Transactions on Communication,Vol.39(4), pp.514-523, 1991.

[19]R. Miller and S. Paul, “Generating Conformance Test Sequences for Combined Control and Data Flow of Communication Protocols,” In Protocol Specifications, Testing and Verification, pp.13-27, 1992.

[20]R.M.Hierons, T.H.Kim, and H.Ural, “Expanding an extended finite state machine to aid testability,” COMPSAC, pp.334-339, 2002.

[21]M. Ü. Uyar and A. Y. Duale, “Test generation for EFSM models of complex army protocols with inconsistencies,” 21st Century Military Communications. Architectures and Technologies for Information Superiority, Vol 0(C), pp. 340-346, 2000.

[22]R. Lefticaru and F. Ipate, “Automatic State-Based Test Generation Using Genetic Algorithms,” SYNASC 2007, pp. 188-195, Sep 2007.

[23]N. Tracey, J. Clark, K. Mander, and J. McDermid, “An automated framework for structural test-data generation,” ASE, pp. 285-288, 1998.

[24]T. Yano, E.Martins, and F.L.Sousa, “MOST: A Multi-objective Search-Based Testing from EFSM, ” ICST Workshops, pp.164-173, 2011.

[25]A. S. Kalaji, R. M. Hierons, and S. Swift, “An integrated search-based approach for automatic testing from extended finite state machine (EFSM) models,” Information and Software Technology, Vol.53, pp.1297 - 1318, Dec 2011.

[26]P. McMinn and M. Holcombe, “Evolutionary testing of state-based programs,” GECCO, pp.1013-1020, 2005.

[27] J. Zhang, C. Xu and X. Wang, “Path-Oriented Test Data Generation Using Symbolic Execution and Constraint Solving Techniques,” SEFM, pp.242-250, 2004.

[28]M. Ngo, and H. Tan, “Heuristics-based infeasible path detection for dynamic test data generation,” Information and Software Technology, Vol. 50(7-8), pp. 641-655, Jun 2008.

[29]J. Zhang, R. Yang, Z. Chen, Z. Zhao and B. Xu, “Automated EFSM-Based Test Case Generation with Scatter Search,” ICSE Workshop AST, Accepted, 2012.

Structural Testing for Multithreaded Programs: An Experimental Evaluation of the Cost, Strength and Effectiveness

Silvana M. Melo, Simone R. S. Souza, Paulo S. L. Souza
ICMC/USP, University of São Paulo
São Carlos, São Paulo, Brazil 668–13.560-970
{morita, srocio, pssouza}@icmc.usp.br

Abstract

Concurrent program testing is a complex activity, due to factors do not present in sequential programs, such as non-determinism. In this context, techniques related to static analysis are proposed, but its applicability is not often proven through theoretical or empirical study. This paper aims to contribute in this direction, by presenting the results of an experimental study to evaluate cost, effectiveness and strength of structural test criteria defined for multithreaded programs, implemented with Pthreads (POSIX Threads). The ValiPThread testing tool was used to support the experiment. The programs were selected considering the benchmarks Inspect, Helgrind and Rungta programs, that are commonly used to evaluate techniques and testing criteria in the context of concurrent software. The results indicate that is possible the development of an incremental strategy for application of these testing criteria with low application cost and high effectiveness.

1 Introduction

Empirical studies have been developed in context of sequential programs, showing information about the effectiveness, cost and strength of sequential testing criteria, where the results indicate that structural testing criteria are effective to find faults in programs [3, 8]. The effectiveness is measured calculating the amount of faults revealed for each testing criterion in relation to injected faults. The application cost of the testing criteria can be measured in terms of the test set size and the number of required elements. The strength of a testing criterion C_1 in relation to C_2 is measured evaluating the coverage of a test set T, adequate to C_2, in relation to C_1. If the coverage of T in relation to C_1 is low, it means that C_1 has a high strength in relation to C_2. Otherwise, if the coverage of T in relation to C_1 is high, it means that, in a practical way, C_2 includes C_1.

Concurrent programming became an essential paradigm to reduce the computational time in many application domains. However, 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 testing.

Several works have proposed structural criteria to test concurrent applications, demonstrating that is promising to translate sequential testing criteria to the context of concurrent programs [1, 10, 2, 11]. In the same vein, we have developed structural testing criteria for the validation of concurrent programs, applicable to both message-passing software [6] and multithreaded software [5].

The empirical evaluation of the testing criteria for concurrent programs is an important activity to find evidences about the cost and effectiveness of these testing criteria. This paper contributes in this scenario, where we present an experimental study to evaluate the effectiveness, cost and strength of a set of testing criteria for multithreaded programs, defined by Sarmanho et al. [5]. Considering this objective, the following research question were defined:

Which is the effectiveness, the application cost and the strength of the testing criteria for multithreaded programs?

Based on the research question, the study hypotheses are:

Null Hypothesis 1 (NH1): The application cost is the same for all structural testing criteria analyzed.

Alternative Hypothesis 1 (AH1): The application cost is different for at least one structural testing criterion for multithreaded programs.

Null Hypothesis 2 (NH2): The effectiveness is the same for all structural testing criteria analyzed.

Alternative Hypothesis 2 (AH2): The effectiveness is different for at least one structural testing criterion for multithreaded programs.

Null Hypothesis 3 (NH3): In relation to strength, there is not one multithreaded testing criterion that subsumes another.

Alternative Hypothesis 3 (AH3): There is at least one multithreaded testing criterion that subsumes another.

This paper is organized as follows. In Section 2 is described the structural testing for multithreaded programs studied in this paper. In Section 3 is presented the planning and conduction of the experimental study. Section 4 discusses the results obtained with the experimental study and, finally, in Section 5 is presented the conclusions and future work.

2 Structural Testing of Concurrent Programs

In this section, we present the structural testing criteria for multithreaded programs investigated in our experimental study. More details about the definition of these testing criteria are presented in Sarmanho et al. [5].

Based on previous work, the definition of the structural testing for multithreaded programs uses a test model that captures relevant information for the testing, such as data, control, synchronization and communication flow. A *Parallel Control Flow Graph for Shared Memory* (PCFGsm) is proposed to represent the information flow of the multithreaded program under test, composed of a Control Flow Graph for each thread and edges related to syncrhonization among threads. In this model the communication happens implicitly, by means of shared variables and the synchronization happens explicitly by means of semaphores. Based on this test model and on the sequential testing criteria, the following testing criteria were defined:

All-nodes: requires the execution of all nodes of the PCFGsm;

All-p-nodes: requires the execution of all nodes thats contains the synchronization primitive post;

All-w-nodes: requires the execution of all nodes thats contains the synchronization primitive wait;

All-s-edges: requires the execution of all edges thats contains synchronizations instructions related to semaphores;

All-edges: requires the execution of all edges of the PCFGsm.

All-def-comm: requires the coverage of definition-clear paths that execute all definitions of shared variables associated for at least one communicational use;

All-def: requires the coverage of definition-clear paths that execute all definition of shared and local variables associated for at least one use (computational, communicational or predicative use).

All-comm-c-uses: requires the execution of definition-clear paths that execute all computational uses of each definition of shared variable;

All-com-p-use: requires the execution of definition-clear paths that execute all predicative uses of each definition of shared variable;

All-c-uses: requires the execution of definition-clear paths that execute all computational uses of each definition of local variable;

All-p-uses: requires the execution of definition-clear paths that execute all predicative uses of each definition of local variable;

All-sync-uses: requires the execution of definition-clear paths that execute synchronization uses for all definition of semaphore variable.

A testing tool, named ValiPThread, was implemented to support these test model and testing criteria[5]. This tool is configured to test PThreads/ANSI C programs, allowing the following functionalities: static analysis, generation of required elements from multithreaded program, execution of test cases, controlled execution of test cases and evaluation of the test cases coverage. This tool was used in the experimental study described in the next section.

3 Experimental Study: Description and Planning

The experimental study was performed according to the Experimental Software Engineering Process, proposed by Wholin et al. [9]. Considering this process, the subjects of the experiment are the structural testing criteria for multithreaded programs (Section 2). The goal is the evaluation of three factors: application cost, effectiveness and strength.

Aiming this goal, the programs were selected considering some benchmarks commonly used for the evaluation of testing criteria in the context of concurrent software. We have selected programs from three different benchmarks: Inspect (12 programs), Helgrind (12 programs) and Rungta benchmark (4 programs) [12, 7, 4]. Furthermore, we have selected more five programs that implement solutions to concurrent classical problems.

Each program was executed in the ValiPThread tool following a similar strategy: 1) generation and execution of the test cases; 2) analysis of the coverage in relation to each criterion; 3) generation of new test cases until to obtain the maximum coverage, where the maximum coverage corresponds to execute all feasible required elements of the testing criterion; 4) analysis of the required elements to determine those infeasible. An element is infeasible if there is no set of values for the parameters (considering the input and global variables) that cover that element.

After the application of this strategy, we obtained an adequate test set for each testing criterion. These adequate test sets are used to evaluate the strength of the criteria. This analysis is performed by applying each test set T, adequate for a given criterion, to other criteria, evaluating the coverage obtained for T. The strength is a measure that evaluate the difficulty to satisfy one given criterion with a test set adequate to another criterion. This measure is important to evaluate empirically the inclusion relation among criteria.

During the generation of test cases was observed the effectiveness of each criteria to reveal the existent faults. Some programs of the benchmarks Inspect, Helgrind and Rungta have injected fault. If a test case belonging to an adequate test set is able to reveal fault, this criterion is considered effective to reveal this kind of fault.

4 Experimental Study: Results

The analysis and interpretation of the results are made based on principles of the descriptive statistics and hypothesis testing. Using the analysis of variance (ANOVA) [9], it was verified whether it is possible to reject the null hypothesis based on collected data set and statistical tests. The descriptive analysis is useful to describe and to show graphically interesting aspects of the study. There are different perspectives to evaluate the cost of a testing criterion. In this study we choose the size of the adequate test set and the number of required elements. Table 1 presents information of cost for some programs of the experiment, showing the size of the adequate test set and the number of feasible required elements for each testing criterion. The cost data obtained for all programs of the experiment can suggest an order to apply these criteria, considering initially the criteria with minor cost: All-w-nodes (ANW), All-p-nodes (ANP), All-p-uses (APU), All-c-uses (ACU), All-nodes (AN), All-comm-c-uses (ACCU), All-comm-p-uses (ACPU), All-sync-uses (ASU), All-s-edges (AES) and All-edges (AE).

Criteria	MMult	Lazy01	Jacobi	Stateful06	Effectiveness
All-nodes	12/340	2/26	4/260	1/26	75%
All-p-nodes	12/54	1/11	2/31	1/7	50%
All-w-nodes	12/49	1/9	3/27	1/6	50%
All-edges	52/425	5/19	11/149	2/13	100%
All-s-edges	52/328	5/15	11/75	2/8	100%
All-c-uses	12/247	1/3	2/67	1/8	50%
All-p-uses	12/209	-	2/64	1/6	66%
All-sync-uses	52/270	4/9	-	2/6	75%
All-comm-c-uses	24/320	2/2	6/100	1/1	75%
All-comm-p-uses	-	4/6	7/61	-	100%

Table 1. Costs and effectiveness for some programs in the experiment.

The test set size was used to evaluate the cost and to perform the hypothesis testing. Based on the ANOVA analysis, the null hypothesis (NH1) is rejected because the *p-value* obtained for a pair of different criteria was 0.0009, less than the significance level of 0.05. This result suggests that there exist differences among the costs of these testing criteria. In this case, the alternative hypothesis (AH1) is accepted.

The effectiveness is calculated by the following equation:

$$effectiveness = \frac{number\ of\ faults\ found}{number\ of\ faults\ injected} * 100$$

In this analysis was considered the programs with injected faults, totaling 23 programs. The results are presented in Table 1. The criteria ACPU, AE and AES are the most effective to reveal the faults, however there are some kind of faults that are revealed for few testing criteria and, in some cases, testing criteria with less effectiveness are responsible to identify specific kind of faults. The results of the ANOVA analysis suggest that is possible reject the null hypothesis (NH2) and accept the alternative hypothesis (AH2), because the *p-value* obtained is 0.007, less than 0.05. These results indicate that there is a considerable difference of effectiveness among the testing criteria.

The equation below is used to calculate of *strength* of a criterion C_1:

$$strength_{C_1} = \frac{Number\ of\ elements\ covered\ by\ T_{C2}}{Total\ of\ required\ elements\ -\ infeasible\ elements}$$

Data analysis related to the *strength* was performed using the statistical method of cluster analysis in order to identify if there is an inclusion relation among the criteria. Dendrogram charts are used to illustrate the results about inclusion relation among criteria. If a criterion $C1$ includes another criteria $C2$ by applying the adequate test set T_{C1}, these criteria are at same level in the graph. Analysing the Dendogram 1 we can conclude that the ACCU criterion includes the ANW criterion because both are at the same level in the graph. Based on this result the null hypothesis (NH3) can be rejected and the alternative hypothesis (AH3) is accepted, indicating that the criteria can be complementary.

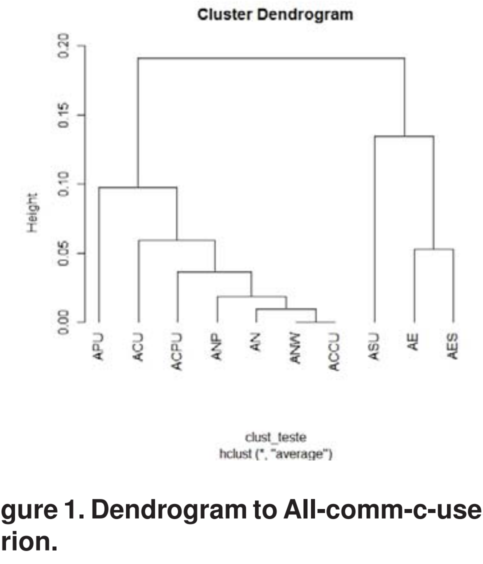


Figure 1. Dendrogram to All-comm-c-uses criterion.

The cluster analysis for the remaining criteria indicates the following inclusion relation among the criteria: ACCU, APU, AE, AES include the ANW criterion; the APU criterion includes the ACU criterion and the ANW includes the ACU criterion. These results suggest a test strategy to apply these testing criteria: 1) generate test cases adequate to ACCU criterion; 2) in the sequence, select new test cases to cover the APU criterion and 3) since there is not relationship between the other criteria pairs, it can be used information about the cost and effectiveness to generate new test cases, assuring that relevant aspects of the application under test will be properly tested.

4.1 Threats to Validity

The external validity of the experiment refers to the ability to generalize the results. Since the programs used are small, they may not be representative of the population. However these benchmarks are largely used to compare other testing techniques for multithreaded programs and they include programs that contain faults commonly found in these applications. The generalization can be achieved if the experiment is replicated considering other domain of concurrent applications.

The construct validity threats concern the relationship between theory and observation. In our case, this threat is related to knowledge about the used programs and inserted faults, where this knowledge can influence the application of the testing criterion. To avoid this threat, the order of application of the criteria was different for each program, preventing that the knowledge of the programs and inserted faults would influence in the generation of test cases.

5 Conclusion

This paper presents an experimental study to evaluate a family of structural testing criteria for multithreaded programs, where the objective was to find evidence about the cost, effectiveness and strength of these testing criteria, aiming to define a testing strategy to apply them.

The results of hypothesis analysis indicate that the testing criteria present different cost and effectivess. The comparison among the sequential criteria: AN, AE, ACU and APU, and the respective concurrent criteria: ANP and ANW, AES, ACCU and ACPU, resulted that the sequential criteria sequential are the most expensive in terms of application cost. In respect to effectiveness, the sequencial testing criteria obtained better results, where only the AES criterion had effectiveness similar to the AE criterion. The results related to the strength indicated the complementary relationship among the testing criteria.

According to Wohlin et al. [9], an experiment will never provide the final answer to a question, hence, it is important to facilitate its replication. In this sense, we are packing the material used and generated in this experiment to provide a lab package, which it is available for public access [1]. We believe that is important to evaluate these aspects considering other testing criteria for multithreaded programs, but using the same benchmarks to allow the comparison between the results obtained.

Acknowledgment

The authors would like to thank FAPESP, Brazilian funding agency, for the financial support (process number: 2010/04042-1).

References

[1] O. Edelstein, E. Farchi, E. Goldin, Y. Nir, G. Ratsaby, and S. Ur. Framework for testing multi-threaded java programs. *Concurr. Comput.: Pract. Exper.*, 15:485–499, 2003.

[2] P. V. Koppol, R. H. Carver, and K. C. Tai. Incremental integration testing of concurrent programs. *IEEE Trans. Softw. Eng.*, 28:607–623, June 2002.

[3] N. Li, U. Praphamontripong, and J. Offutt. An experimental comparison of four unit test criteria: Mutation, edge-pair, all-uses and prime path coverage. In *ICSTW09 - International Conference on Software Testing, Verification and Validation Workshops*, pages 220 –229, april 2009.

[4] N. Rungta and E. G. Mercer. Clash of the titans: tools and techniques for hunting bugs in concurrent programs. In *Proc. of PADTAD '09*, pages 9:1–9:10, New York, NY, USA, 2009. ACM.

[5] F. S. Sarmanho, P. S. Souza, S. R. Souza, and A. S. Simão. Structural testing for semaphore-based multithread programs. In *Proc. of ICCS '08*, pages 337–346, Berlin, Heidelberg, 2008.

[6] 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. *Concurr. Comput. : Pract. Exper.*, 20:1893–1916, November 2008.

[7] Valgrind-Developers. Valgrind-3.6.1. http://valgrind.org. Accessed: jun/2011.

[8] E. J. Weyuker. The cost of data flow testing: An empirical study. *IEEE Trans. Softw. Eng.*, 16:121–128, February 1990.

[9] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in software engineering: An Introduction*. Kluwer Academic Publishers, 2000.

[10] W. E. Wong, Y. Lei, and X. Ma. Effective generation of test sequences for structural testing of concurrent programs. In *ICECCS*, pages 539–548, 2005.

[11] C. S. D. Yang, A. L. Souter, and L. L. Pollock. All-du-path coverage for parallel programs. *SIGSOFT Softw. Eng. Notes*, 23:153–162, March 1998.

[12] Y. Yang, X. Chen, and G. Gopalakrishnan. Inspect: A runtime model checker for multithreaded C programs. Technical report, 2008.

[1] `http://stoa.usp.br/silvanamm/files`

Towards a Unified Source Code Measurement Framework Supporting Multiple Programming Languages

Reisha Humaira, Kazunori Sakamoto, Akira Ohashi, Hironori Washizaki, Yoshiaki Fukazawa
Dept. Computer Science and Engineering
Waseda University
Tokyo, Japan
(reisha@fuji, kazuu@ruri, akira-radiant@akane).waseda.jp / (washizaki, fukazawa)@waseda.jp

Abstract—**Software metrics measure various attributes of a piece of software and are becoming essential for a variety of purposes, including software quality evaluation. One type of measurement is based on source code evaluation. Many tools have been developed to perform source code analysis or to measure various metrics, but most use different metrics definitions, leading to inconsistencies in measurement results. The metrics measured by these tools also vary by programming language. We propose a unified framework for measuring source code that supports multiple programming languages. In this paper, we present commonalities of measurable elements from various programming languages as the foundation for developing the framework. We then describe the approach used within the framework and also its preliminary development. We believe that our approach can solve the problems with existing measurement tools.**

Keywords-source code measurement; metrics; framework; multiple languages

I. INTRODUCTION

Quantitative measurements have become essential in software development for a variety of purposes, especially the evaluation of software quality. Software metrics as a measure of properties of a piece of software are commonly used by software developers and are based on software processes, products, or resources [1]. There are various kinds of measurements that can be conducted based on the goal of the metric.

One type of measurement is performed based on source code. There are already a large number of source code measurement tools, but most of them are only able to handle one programming language [2], [3], [4], [5], [6]. Some, such as [7] and [8], are able to handle multiple programming languages, but we still need to specify which programming language should be measured, which means these tools cannot handle software written in more than one programming language. In addition, some measurement tools cover only one metric category. For example, [9] and [10] only assess size by measuring lines of code, so if we want to measure complexity we have to find another measurement tool. When different tools are used together, there can be inconsistency in measurement results due to differences in measurement definitions and standards among the tools [11].

It would be worthwhile to have a unified source code measurement system for multiple programming languages. However, the implementation would not be easy since processing each programming language and metric would entail significant expense. In this paper, we present the preliminary efforts related to our proposed measurement framework. We extract the commonalities of various elements needed to measure source code metrics regarding the features of a particular programming language. The results are the foundation for developing a lower-cost framework, and it will be easier to define or modify the measurement for use with multiple programming languages.

The remainder of this paper is organized as follows. Section 2 discusses problems with existing measurement tools, and also metrics that can be measured from source code. Section 3 discusses commonalities of metrics for several programming languages. Section 4 provides an overview of the proposed framework for source code measurement supporting multiple programming languages. Section 5 discusses a case study. Section 6 considers related works. Finally, Section 7 addresses conclusions and future work.

II. BACKGROUND

A. *Problems with Existing Approaches*

The following summarizes the main problems with existing tools.

1) Cost of New Development

The implementation of a measurement tool supporting multiple programming languages would be very expensive, mainly due to the difficulty in implementing an analyzer to obtain information from the source code of every language. In addition, since developers increasingly use object-oriented programming languages, metric tools are built mainly to support that paradigm (in addition to the basic measurement of lines of code) [11]. Tools are often unavailable for other paradigms. However, procedural or functional languages are still popular, and tools compatible with these languages are therefore necessary.

2) Incomprehensive Measurement

Most existing source code measurement tools only assess software size and cannot evaluate software written in multiple languages. JSUnit [12] is an example of a program that uses more than one programming language. JSUnit is a client-server-type test framework that performs tests on web browsers and displays the results on the client side. The client side of JSUnit is implemented in JavaScript and the server side is implemented in Java. With existing tools, the measurement of JSUnit metrics requires that the Java and JavaScript code be separated and that two measurements be performed.

3) Inconsistent Measurement

Different measurement tools have different metrics definitions that may produce different results [11]. For example, we evaluated several Java code files within the package `net.jsunit.model` and some JavaScript code files within the `app` directory of JSUnit. We obtained the source code of JSUnit from its Github project page[1]. Table I shows the measurement results using different tools. In this example, lines of code (LOC) in Metrics 1.3.6 is defined as non-blank lines of code, but Source Monitor also counts blank lines when measuring LOC. Furthermore, Count Lines of Code (CLOC) skips blank lines and comment lines when measuring code lines.

TABLE I. DIFFERENCES BETWEEN METRICS TOOLS

File Name	**Metrics 1.3.6 [5]**	**Source Monitor [7]**		**CLOC [10]**	
	Lines	**Lines**	**Statement**	**Blank**	**Code**
AbstractResult.java	73	88	57	15	73
BrowserResult.java	197	248	147	51	197
BrowserSource.java	6	10	5	4	6
TestCaseResult.java	128	160	97	32	128
TestPageResult.java	45	61	32	16	45
TestRunResult.java	184	223	147	39	184
jsUnitCore.js	n/a	977	283	107	504
jsUnitParams.js	n/a	117	62	24	93
jsUnitTracer.js	n/a	48	33	9	39

B. Source Code Metrics

This section reviews the source code measurement systems that are most commonly used. We focus on size and complexity because they are the most important measurements [13].

1) Size Measurement

Lines of code (LOC) are the traditional measurement of software size. We define: the number of physical lines of target source code for measurement (*total lines of code*); the number of lines that contain the instructions necessary to execute a program (*effective lines of code*); the number of lines that contain only comments (*comment lines of code*); and the number of the lines that contain only spaces, tabs, or newline character(s) (*blank lines of code*). Measuring the number of "statements", which are logical measures based on the specification of the programming language, could also be a component of size measurement.

2) Complexity Measurement

The following are several complexity measurements [13].

- Cyclomatic complexity: the complexity measurement proposed by McCabe, which is a measure of the number of control flows within a module.
- Halstead's software science: complexity metrics based on the number of operators and operands in a program.
- Information flow metrics: these measure the information flow into and out of modules, and indicate the cohesion of the program.
- CK metrics: a set of object-oriented design metrics by Chidamber and Kemerer, consisting of six metrics that measure class size and complexity, use of inheritance, coupling between classes, class cohesion, and collaboration between classes.

III. ASSESSING SOURCE CODE METRICS AND PROGRAMMING LANGUAGES

To define the base of our framework, we conducted assessments of source code metrics and programming language features. These assessments produce the commonalities of source code's measurable elements. From these measurable elements, we can define a metric more easily and even customize a new metric by utilizing the predefined measurable elements.

We analyzed the metrics described in subsection II-B to determine what elements of a program are used by these metrics. The size measurements are simply related to the

TABLE II. MEASURABLE ELEMENTS USED BY METRICS

Measurable Elements	**STMT[a]**	**Halstead[b]**	**CK Metrics[c]**					
			WMC	**DIT**	**NOC**	**CBO**	**RFC**	**LCOM**
statement	√							
operator		√						
operands		√						
Entities								
class			√	√	√	√	√	
method			√				√	√
attribute								√
Relationships								
method calls method						√	√	
class inherits from class				√				
class is a child of class					√			
class uses instance of other class						√		
class is parent to class					√			
class/method uses attribute								√

a. Number of statements. b. Halstead's software science.

c. The CK metrics: weighted methods per class (WMC), depth of inheritance tree (DIT), number of children (NOC), coupling between object (CBO), response for class (RFC), and lack of cohesion in methods (LCOM)

[1] https://github.com/pivotal/jsunit

physical lines of source code. The number of statements is based on the "statement" definition in the programming language. The cyclomatic complexity is measured based on control flow, and Halstead's software science is related to operators and operands. Reference [14] has developed a data model of information from which most of the proposed object-oriented design metrics can be computed. The data model lists 12 entities and 17 relationships that are used by the metrics. We found that at the implementation level, the "class sends message to class/method" relationship is similar to "method calls method", so we combined them. In addition, we also combined the "class uses attribute of class" and "class/method uses attribute" relationships. Table II summarizes the measurable elements that are used by number of statements, Halstead's software science metrics, and CK Metrics.

We expanded this preliminary measurable elements list so we could assess other features of programming languages. We also wanted to cover measurable elements provided by non-object-oriented languages.

We included 20 programming languages in our analysis, based on [15]. Table III summarizes the measurable elements that could be quantified in each programming language. Table III consists of two parts. The first part describes the paradigm used by each programming language, based on the definition for each paradigm as presented in [16]. The checkmarks ("√") show the supported paradigms.

The second part depicts the measurable elements. A checkmark indicates that the measurable element could be quantified in the corresponding programming language. Every programming language has statements, operators, and operands that we could count. Therefore, we inferred that the number of statements and Halstead's software science metrics could be calculated for each language. For other elements, the results varied by programming language.

Non-object-oriented programming languages do not utilize the class and inheritance concepts. Alternatively, they use other concepts that can be measured, such as modules, functions, and procedures. We can possibly use these elements to produce complexity measurements such as numbers of functions, numbers of procedures, etc.

In comparison, languages designed mainly for object-oriented programming such as Java, C#, C++, and Python incorporate measurable elements that permit us to assess object-oriented metrics. Nevertheless, historically imperative languages that have been extended with some object-oriented features such as Perl and Ada yield different results, especially in association with the class concept. JavaScript is also a distinctive case.

Measurable elements are readily distinguished based on language syntax. In Perl and Ada, we can declare a class using the `package` keyword. However, the same keyword is also used to declare namespace, subprogram, or data types. Therefore, we cannot determine that all `package` keywords indicate the presence of classes. In spite of this, Perl has the special keyword `@ISA` and Ada uses `tagged` to indicate inheritance, both of which allow us to count the measurable elements related to inheritance. We could possibly designate packages associated with the inheritance keyword as "classes" instead of "packages", but we would perhaps incorrectly measure the number of classes since classes without inheritance would not be counted. In other words, we could partially count the "class" number. We note such cases with dots ("●") in Table III.

JavaScript is actually a prototype-based scripting language. Although it supports the object-oriented paradigm, the identification of classes or inheritance is complicated by the fact that we utilize `function` to declare a class. Therefore, in JavaScript we interpreted `function` syntax as representing a "function" although semantically it could possibly represent a "class".

The above results show that object-oriented design metrics could not be applied to every object-oriented programming language due to the availability of the measurable elements. Based on Table II, we required the "class" entity in order to measure CK Metrics. We are able to measure WMC, DIT, NOC, CBO, and RFC for Java, C#, C++, Objective-C, PHP, (Visual) Basic, Python, Delphi, and Ruby, but we cannot measure them for JavaScript since we are unable to precisely identify its classes.

IV. Overview of Proposed Framework

We propose a framework to measure source code that supports multiple languages and will diminish the problems previously described in subsection II-A. The following are the main features of the measurement framework:

- The framework should provide an easy way to measure a metric.
- The framework should be able to evaluate the metrics of software written in more than one programming language.
- The framework should use the same metric definition standard.

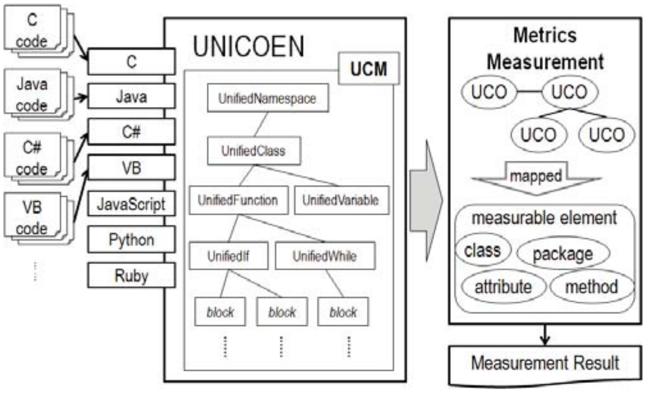


Figure 1. Overview of proposed framework.

Figure 1 shows the overview of our proposed framework. In order to reduce the cost of developing the framework, we used UNICOEN, a unified framework for code engineering, supporting multiple programming languages currently being developed by [17]. UNICOEN supplies a common

TABLE III. MEASURABLE ELEMENTS IN EACH PROGRAMMING LANGUAGE

Programming Languages		Java	C	C#	C++	Objective-C	PHP	(Visual) Basic	Python	Perl	JavaScript	Delphi	Ruby	Lisp	Pascal	Transact-SQL	PL/SQL	Ada	Logo	R	Lua
Paradigm	imperative	√	√	√	√		√	√	√	√	√	√	√	√	√	√	√	√	√	√	√
	object-oriented	√		√	√	√	√	√	√	√	√	√	√					√		√	
	functional										√			√					√	√	√
	logic															√	√				
Measurable elements: entities																					
statement		√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√
operator		√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√
operand		√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√
namespace				√	√		√	√				√								√	
package		√								√							√	√			
class		√		√	√	√	√	√	√	●		√	√					●		√	
module			√												√						√
method		√		√	√	√		√	√	√		√	√							√	
procedure			√												√	√	√	√			
function			√				√				√			√	√	√	√	√			√
structure/record			√	√	√	√				√		√			√			√			
union			√		√	√												√			
attribute/variable		√	√	√	√	√	√	√	√	√	√	√	√		√			√		√	
method arguments		√		√	√	√		√	√	√		√	√							√	
procedure arguments			√												√	√	√	√	√		
function arguments			√				√				√			√	√	√	√	√	√		√
return value		√	√	√	√	√	√	√	√	√	√	√	√	√		√	√	√		√	√
global variable							√		√				√								
abstract class		√		√	√		√	√	√									√			
interface		√		√			√											√			
Measurable elements: relationships																					
method calls method		√		√	√	√		√	√	√		√	√							√	
class inherits from class		√		√	√	√	√	√	√	√		√	√					√		√	
class is a child of class		√		√	√	√	√	√	√	√		√	√					√		√	
class uses instance of other class		√		√	√	√	√	√	√	√		√	√					√		√	
class is parent to class		√		√	√	√	√	√	√	√		√	√					√		√	
class/method uses attribute		√		√	√	√	√	√	√	√		√	√					√		√	
procedure/function calls procedure/function			√				√				√			√	√	√	√	√			√

representation of source code for different programming languages called the unified code model (UCM), and objects generated from the source code according to the UCM are called unified code objects (UCOs). At present, UNICOEN supports seven programming languages: C, Java, C#, VB, JavaScript, Python, and Ruby.

The basic purpose of the framework was to define the measurable elements we obtained in Section III. These elements were obtained by mapping UCOs generated by UNICOEN to our definitions. UNICOEN generates UCOs from the input source code. After the mapping process, we could easily count each element and then utilize it to evaluate the corresponding metrics.

The mapping process is described as follows. The partial structure of the UCM is depicted in Figure 1. Each node in the UCM will constitute one UCO block, and relationships among the blocks will form a tree structure. A UCO block contains primary information such as UnifiedNamespace, UnifiedClass, UnifiedVariable, UnifiedFunction, etc., whose names refer to the types of entities they are. There is also additional information such as modifier, type, extend-constrain, identifier, etc., which describe the details of each entity.

Based on this information, for example, we can map a UnifiedNamespace to the "package" entity for Java code or to the "namespace" entity for C# code. A UnifiedFunction is then mapped to "method" for Java code. For C code, UnifiedFunction could be mapped to "function" if the block contains UnifiedReturn information or "procedure" if it contains no UnifiedReturn. Currently the UCOs can cover all of the measurable elements for the seven programming languages mentioned previously.

V. CASE STUDY

In this section, we illustrate the evaluation of metrics using our approach. For the case study, we analyzed the same files listed in Table I. We describe the measurement of total and blank lines of code, number of statements, and number of children.

A. *Measuring Total and Blank Lines of Code*

These measurements are directly related to the physical lines of source code, so did not require any mapping from UCOs. The framework read each file line-by-line and counted the number of times a line was read. This resulted in total lines of code (TLOC). Blank lines of code (BLOC) was defined as the number of lines that contained only spaces, tabs, or newline character(s). It was calculated by reading each line and tallying if the line has only these character types. The results of our measurements are summarized in Table IV. To obtain the number of lines excluding blank lines, we can easily subtract BLOC from TLOC.

B. *Measuring Number of Statements*

While the lines of code are generically called "the physical lines of code", the number of statements (NOS) is called "the logical lines of code". NOS is measured based on UCOs. Each block in each UCO is basically mapped to a statement. Therefore, the NOS value is exactly the same as the total number of blocks in the UCOs.

```
package net.jsunit.model;
import java.util.List;
public interface BrowserSource {
    Browser getBrowserById(int id);
    List<Browser> getAllBrowsers();
}
```

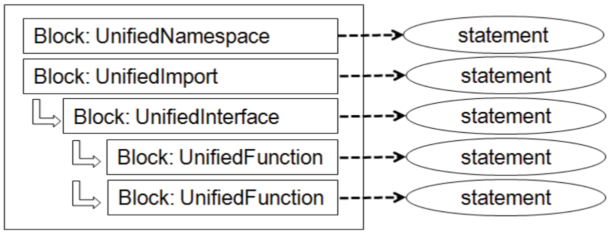


Figure 2. Mapping of BrowserSource.java to UCO and then to measurable elements (statements).

TABLE IV. TLOC, BLOC, AND NOS MEASUREMENT RESULTS

File Name	TLOC	BLOC	NOS
AbstractResult.java	89	16	56
BrowserResult.java	249	52	145
BrowserSource.java	11	5	5
TestCaseResult.java	160	32	95
TestPageResult.java	62	17	32
TestRunResult.java	224	40	142
jsUnitCore.js	978	108	361
jsUnitParams.js	118	25	62
jsUnitTracer.js	48	9	29

Figure 2 describes how the code in BrowserSource.java is represented using UCOs. Each block is considered as one statement, so the NOS for BrowserSource.java is 5. The mapping mechanism for JavaScript code is the same as for Java code. The NOS results are summarized in Table IV.

C. *Measuring Number of Children*

The number of children (NOC) is defined as the number of immediate subclasses subordinated to a class in the class hierarchy. To measure the NOC we distinguished the relationships "class is a child of class" and "class is parent to class". Determining these relationships based on UCOs is not as simple as mapping UCOs to entities such as packages, classes, etc. A class that inherits another class will contain extend-constrain information in its UnifiedClass block. To find the inheritance relationship, we traced whether the "extend-constrain" name of a class was identical to that of any class name within the project. Using our sample code, this approach is demonstrated in Figure 3.

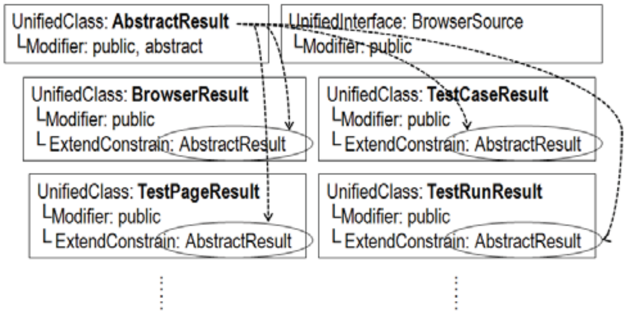

UnifiedClass: AbstractResult
Modifier: public, abstract
UnifiedInterface: BrowserSource
Modifier: public
UnifiedClass: BrowserResult
Modifier: public
ExtendConstrain: AbstractResult
UnifiedClass: TestCaseResult
Modifier: public
ExtendConstrain: AbstractResult
UnifiedClass: TestPageResult
Modifier: public
ExtendConstrain: AbstractResult
UnifiedClass: TestRunResult
Modifier: public
ExtendConstrain: AbstractResult

Figure 3. Tracing the children of the AbstractResult class.

In this case, we obtained the following relationships. The TestPageResult, TestRunResult, TestCaseResult, and BrowserResult classes were children of the AbstractResult class. The AbstractResult class was the parent of the TestPageResult, TestRunResult, TestCaseResult, and BrowserResult classes. We could infer, then, that the NOC for the AbstractResult class was 4.

Using this approach makes it easier to map the UCOs to measurable elements and to determine relationships. Although currently UNICOEN supports only seven languages, it is easy to add new languages or update existing languages, so we could easily add measurements for other languages, thus reducing the cost of development. In addition, this approach allows us to fully measure various metrics for programs written in more than one programming language using only one framework. Regarding the problem with measurement inconsistencies demonstrated in Table I, we cannot say whether our results are more appropriate since there are no international standards prescribing the optimal way to measure a metric. However, our approach used the same metric definition for each programming language supported by our framework. Hence, we could obtain consistent results among all the languages, for instance between Java and JavaScript code as shown in Table IV.

VI. Related Works

This section discusses the related research approaches of Baroni et al. [18], Higo et al. [19], and Maneva et al. [20]. The goals of these research groups were similar to our own in terms of measuring software metrics.

Baroni et al. developed MOOSE as a language-independent platform that outputs object-oriented design metrics. MOOSE analyzes OCL and UML metamodel representations. Their approach is limited because the amount of information obtained from their representation is less than that which can be directly extracted from source code. The UCOs generated by UNICOEN contain more information than the representations used by MOOSE. For instance, with UCOs we can trace the "method calls method" and "class/method uses attribute" relationships, but MOOSE is unable to recognize them.

Higo et al. built MASU as a plug-in for measuring metrics of source code written in multiple programming languages. Their work focused primarily on developing a source code analyzer for each programming language. This analyzer built AST from the input source code of object-oriented programming languages. Currently MASU handles Java and C# source code. Our approach simplifies the measurement by assessing only the measurable elements. In addition, MASU was designed to work only with object-oriented languages, whereas our framework can handle non-object-oriented language such as C.

Maneva et al. proposed using a framework to evaluate source code. The key ideas are that a base set of metrics must be defined and the computation of the measures are distributed into distinct modules. They implement preprocessor functions to filter the source code artifacts from elements that render the computation of metric values inaccurate in some contexts. Their research does not mention a means of processing multiple languages, which is the main contribution of our approach.

VII. Conclusion and Future Work

This paper described our proposed unified source code measurement framework for multiple programming languages. We reported three main problems with existing source code metric tools: cost of new development, incomprehensive measurement, and inconsistent measurement. Based on size and complexity measurements, we assessed source code metrics and various programming languages and obtained the commonalities of measurable source code elements. The assessment results showed that we were able to measure size and some complexity metrics for every programming language. However, in some cases we could not measure object-oriented design metrics even though a language such as JavaScript supports the object-oriented paradigm, while in other cases, such as Perl and Ada, we could only partially conduct measurements. The next step in our approach is to identify measurable elements by mapping the representations produced by UNICOEN. As a case study, we explained the way our framework measures the total and blank lines of code, number of statements, and number of children.

We limit our work to several size and complexity metrics, so some other metrics were not considered. In addition, our framework is still early in development. Further work is needed to evaluate whether it can properly solve the problems we mentioned previously. But we believe that our approach can contribute to reducing the cost of developing a source code measurement tool, even for software written in more than one programming language. In addition, most current tools only perform static analysis of one version of software, and we will also therefore consider identifying distinctions between two versions of source code.

References

[1] N. Fenton, "Software measurement: a necessary scientific basis," IEEE Transactions on Software Engineering, vol. 20, pp199–206, March 1994.

[2] Virtual Machinery, "JHawk," http://www.virtualmachinery.com.

[3] Semantic Designs, Inc., "Java Source Code Metrics," http://www.semanticdesigns.com.

[4] Clarkware Consulting, inc., "JDepend," http://www.clarkware.com/software/JDepend.html.

[5] F. Sauer, "Eclipse Metrics plugin 1.3.6," http://metrics.sourceforge.net.

[6] Chr. Clemens Lee, "JavaNCSS - A Source Measurement Suite for Java," http://javancss.codehaus.org.

[7] Campwood Software, "SourceMonitor," http://www.campwoodsw.com.

[8] M Squared Technologies, "Resource Standard Metrics," http://msquaredtechnologies.com.

[9] D. A. Wheeler, "SLOCCount," http://www.dwheeler.com/sloccount.

[10] Northrop Grumman Corporation, "CLOC - Count Lines of Code," http://cloc.sourceforge.net.

[11] R. Lincke, J. Lundberg, and W. Lowe, "Comparing software metrics tools," Proc. of International Symposium on Software Testing and Analysis, pp. 131–142, July 2008.

[12] E. Gamma and K. Beck, "JSUnit," http://www.jsunit.net.

[13] L. M. Laird and M. C. Brennan, Software measurement and estimation: a practical approach, IEEE Computer Society, 2006.

[14] J.R. Abounader and D. A. Lamb, A data model for object-oriented design metrics. Kingston, ON: Queen's University, 1997.

[15] TIOBE Software, "TIOBE Programming Community Index for January 2012," http://www.tiobe.com.

[16] A. B. Tucker and R. E. Noonan, Programming Languages Principles and Paradigms, 2nd ed, McGraw-Hill, 2007.

[17] K. Sakamoto, A. Ohashi, D. Ota, and H. Iwasawa, "UNICOEN," http://www.unicoen.net.

[18] A. L. Baroni and F. B. Abreu, "An OCL-based formalization of the MOOSE metric suite," Proc. of 7th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering, July 2003.

[19] Y. Higo, A. Saitoh, G. Yamada, T. Miyake, S. Kusumoto, and K. Inoue, "A pluggable tool for measuring software metrics from source code," Proc. of the Joint Conference of the 21th IWSM/MENSURA, November 2011.

[20] N. Maneva, N. Grozev, and D. Lilov, "A framework for source code metrics," Proc. of 11th International Conference on Computer Systems and Technologies, June 2010.

A Tiny Specification Metalanguage

Walter Wilson, Yu Lei
Dept. of Computer Science and Engineering,
The University of Texas at Arlington
Arlington, Texas 76019, USA
wwwilson1@sbcglobal.net, ylei@cse.uta.edu

Abstract— **A logic programming language with potential software engineering benefit is described. The language is intended as a specification language where the user specifies software functionality while ignoring efficiency. The goals of the language are: (1) a pure specification language – "what, not how", (2) small size, and (3) a metalanguage – able to imitate and thus subsume other languages. The language, called "axiomatic language", is based on the idea that any function or program can be defined by an infinite set of symbolic expressions that enumerates all possible inputs and the corresponding outputs. The language is just a formal system for generating these symbolic expressions. Axiomatic language can be described as pure, definite Prolog with Lisp syntax, HiLog higher-order generalization, and "string variables", which match a string of expressions in a sequence.**

Keywords- specification; metalanguage; logic programming; Prolog; HiLog; Lisp; program transformation

I. Introduction

Programming languages affect programmer productivity. Prechelt [1] found that the scripting languages Perl, Python, Rexx, and Tcl gave twice the productivity over conventional languages C, C++, and Java. A factor of 4-10 productivity gain has been reported for the declarative language Erlang [2]. Both studies showed that lines of source code per hour were roughly the same, regardless of language.

With current programming languages programmers generally keep efficiency in mind. Even in the logic programming field, which is supposed to be declarative, the programmer must understand the execution process in order to write clauses that terminate and are efficient. Program transformation potentially offers a way of achieving the goal of declarative programming where one can write specifications without considering their implementation and then have them transformed into efficient algorithms. One may thus ask what kind of programming language would we want if we could ignore efficiency and assume that a smart translator could transform our specifications?

This paper describes a logic programming language called "axiomatic language" [3] that is in tended to be such a programming/specification language. T he goals of the language are as follows:

1) a pure specification language – There are no language features to control program execution. T he programmer writes specifications while ignoring efficiency and implementation. P rogram transformation is thus required. We assume that a translator can be built that can transform the specifications into efficient programs.
2) small, but extensible – The language should be as small as possible with semantics as simple as possible. Nothing is built-in that can be defined. T his means that arithmetic would not be built-in. (We assume the smart translator can "recognize" the definitions of arithmetic and replace them with the corresponding hardware operations.) Of course, any small language must also be highly extensible, so that those features which are not built-in can be easily defined and used for specification as if they had been built-in.
3) a metalanguage – There are many ways in which one might want to specify software: first-order logic, set operations, domain specific languages, even a procedural language, etc. An ideal language would be able to define within itself other language features and paradigms, which the user could then use for software specification. With such a metalanguage capability, one could not only define functions, but also define new ways of defining functions.

We also have the goal of beauty and elegance for the language.

Axiomatic language is based on the idea that the external behavior of a program – even an interactive program – can be specified by a static infinite set of symbolic expressions that enumerates all possible inputs – or sequences of inputs – along with the corresponding outputs. A xiomatic language is just a formal system for defining these symbolic expressions.

Section II defines the language and section III gives examples. Section IV shows how symbolic expressions can be interpreted as the input and output of real programs. Section V discusses the novel aspects of the language and section VI gives conclusions. This paper extends an earlier publication [4] with syntax extensions and higher-order examples.

II. The Language

This section defines axiomatic language, which is intended to fulfill the objectives identified in the introduction. Section A introduces the language informally with examples. The semantics and syntax of the core language are defined in section B, and section C gives some syntax extensions.

A. An Overview

Axiomatic language can be described as pure definite Prolog with Lisp syntax, HiLog [5] higher-order generalization, and “string variables”, which match a string of expressions in a sequence. A typical Prolog predicate is represented in axiomatic language as follows:

```
  father(sue,X)   ->   (father Sue %x)
```

Predicate and function names are moved inside the parentheses, commas are replaced with blanks, “expression” variables start with %, and both upper and lowercase letters, digits, and most special characters can be used for symbols.

Clauses, or “axioms” as they are called here, are essentially the same as in traditional logic programming, as shown by the following definitions of natural number operations in successor notation:

```
  (number 0).          ! set of natural numbers
  (number (s %n))< (number %n).

  (plus 0 % %)< (number %).       ! addition
  (plus (s %1) %2 (s %3))< (plus %1 %2 %3).

  (times 0 % 0)< (number %).   ! multiplication
  (times (s %1) %2 %3)< (times %1 %2 %x),
                        (plus %x %2 %3).
```

The symbol < replaces the Prolog symbol :- and comments start after !. These axioms generate “valid expressions” such as (number (s (s (s 0)))) and (plus (s 0) (s (s 0)) (s (s (s 0)))), which are interpreted as the statements “3 is a number” and “1 + 2 = 3”, respectively.

The two main features of axiomatic language in comparison to Prolog are its higher-order property and string variables. The higher-order property comes from the fact that predicate and function names can be arbitrary expressions, including variables, and that variables can represent entire predicates in axioms. Section III.A gives some higher-order examples.

The other major feature of axiomatic language is its string variables. A string variable, beginning with $, matches zero or more expressions in a sequence, while an expression variable %... matches exactly one. String variables enable more concise definitions of predicates on lists:

```
  (append ($1) ($2) ($1 $2))   ! concatenation

  (member % ($1 % $2)).    ! member of sequence

  (reverse () ()).      ! reversing a sequence
  (reverse (% $) ($rev %))<
    (reverse ($) ($rev)).
```

Some example valid expressions are (append (a b) (c d) (a b c d)) and (member c (a b c)). String variables can be considered a generalization of Prolog's list tail variables. For example, the Prolog list [a, X | Z] would be represented in axiomatic language as the sequence (a %X $Z). But string variables can occur anywhere in a sequence, not just at the end. Note that sequences in axiomatic language are used both for data lists and term arguments. This single representation is convenient for representing code as data.

B. *The Core Language*

This section gives the definitions and rules of axiomatic language, while the next section gives some syntax extensions. In axiomatic language, functions and programs are specified by a finite set of “axioms”, which generate a (usually) infinite set of “valid expressions”, analogous to the way productions in a grammar generate strings. An **expression** is

an **atom** – a primitive, indivisible element,
an **expression variable**,
or a **sequence** of zero or more expressions and **string variables**.

The hierarchical structure of expressions is inspired by the functional programming language FP [6].

Atoms are represented syntactically by symbols starting with the backquote: `abc, `+. (The non-variable symbols seen previously are not atoms, as section C explains.) Expression and string variables are represented by symbols beginning with % and $, respectively: %expr, %, and $str, $1. A sequence is represented by a string of expressions and string variables separated by blanks and enclosed in parentheses: (`xyz %n), (`M1 $ ()).

An **axiom** consists of a **conclusion** expression and zero or more **condition** expressions, in one of the following formats:

```
  conclu < cond1, …, condn.     ! n>0
  conclu.           ! an unconditional axiom
```

Axioms may be written in free format over multiple lines and a comment may appear on the right side of a line, following an exclamation point. Note that the definition of expressions allows atoms and expression variables to be conclusion and condition expressions in axioms.

An axiom generates an **axiom instance** by the substitution of values for the expression and string variables. An expression variable can be replaced by an arbitrary expression, the same value replacing the same variable throughout the axiom. A string variable can be replaced by a string of zero or more expressions and string variables. For example, the axiom,

```
  (`a (%x %y) $1)< (`b %x $w), (`c $w %y).
```

has an instance,

```
  (`a ((` %) `y))< (`b (` %) () `v),
                   (`c () `v `y).
```

by the substitution of (` %) for %x, `y for %y, the string ‘() `v’ for $w and the empty string for $1.

The conclusion expression of an axiom instance is a **valid expression** if all the condition expressions of the axiom instance are valid expressions. By default, the conclusion of an unconditional axiom instance is a valid expression. For example, the two axioms,

```
  (`a `b).
  ((%) $ $)< (% $).
```

generate the valid expressions (`a `b), ((`a) `b `b), (((`a)) `b `b `b `b), etc. Note that the semantics of axiomatic language is based on definitions that enumerate a set of hierarchical expressions and not on the operation of a resolution algorithm. Note also that valid expressions are just abstract symbolic expressions without any inherent meaning. Their association with real computation and inputs and outputs is by interpretation, as described in section IV.

C. *Syntax Extensions*

The expressiveness of axiomatic language is enhanced by adding syntax extensions to the core language. A single printable character in single quotes is syntactic shorthand for an expression that gives the binary code of the character:

```
'A'  =  (`char (`0 `1 `0 `0  `0 `0 `0 `1))
```

This underlying representation, which would normally be hidden, provides for easy definition of character functions and relations, which are not built-in.

A character string in single quotes within a sequence is equivalent to writing the single characters separately:

```
(… 'abc' …)  =  (… 'a' 'b' 'c' …)
```

A character string in double quotes represents a sequence of those characters:

```
"abc"  =  ('abc')  =  ('a' 'b' 'c')
```

A single or double quote character is repeated when it occurs in a character string enclosed by the same quote character: ″′ ″″″ = (′′′″′) = (′′′′ ′″′).

A symbol that does not begin with one of the special characters ` % $ ( ) ′ ″ ! is equivalent to an expression consisting of the atom ` and the symbol as a character sequence:

```
abc  =  (` "abc")
```

This is useful for higher-order definitions, decimal number representation, and for defining inequality between symbols, which is not built-in.

Other syntax extensions might be useful, such as in-line text, macros, or indentation-based syntax. Any syntax extension that has a clear mapping to the core language could be considered as an addition to the language. We view the definitions and rules of the core language as fixed and permanent, but its syntactic realization and any syntax extensions are open to refinement and enhancement.

III. Examples

This section gives some examples of axiomatic language and discusses its features. Section A gives examples of higher-order definitions and section B shows the metalanguage capability of the language.

A. *Higher-Order Definitions*

In axiomatic language, variables can be used for predicate names and for entire predicates. These higher-order constructs can be powerful tools for defining sets and relations. For example, a predicate for a finite set can be defined in a single expression, as follows:

```
(%set %elem)<(finite_set %set ($1 %elem $2)).
    ! used to define finite sets:
(finite_set day
    (Sun Mon Tue Wed Thu Fri Sat)).
```

These axioms generate valid expressions such as (day Tue). Similarly, instead of defining facts in separate axioms, as would be done in Prolog, we can generate them from a single expression, as follows:

```
% < (valid $1 % $2).
  ! specify multiple facts in one expression:
(valid (father Sue Tom)
       (father Bill Tom)
       (father Jane Bill)).
```

From this we get valid expressions such as (father Jane Bill).

The higher-order capability allows valid expressions to be combined into a single expression. The following axioms form lists of expressions that are all valid:

```
(all_valid).
(all_valid % $)< % , (all_valid $).
    ! all expressions in list are valid:
(grandparent %x %z)<
   (all_valid (parent %x %y) (parent %y %z)).
```

This expression represents the conjunction of valid expressions. We can also have a condition that asserts that at least one expression in a list is valid:

```
(one_valid $1 % $2)< % .
  ! at least one expression in list is valid:
(parent %x %y)<
   (one_valid (mother %x %y) (father %x %y)).
```

This represents valid expression disjunction.

Axioms themselves can be represented as expressions. We can represent a single axiom in an expression as follows:

```
%conclu < (axiom %conclu $conds),
          (all_valid $conds).
   ! specify axiom in a single expression:
(axiom (sort %1 %2)
       (permutation %1 %2) (ordered %2)).
```

This is easily extended to represent a set of axioms in a single expression:

```
(axiom $axiom)< (axiom_set $1 ($axiom) $2).
   ! a set of axioms in a single expression:
(axiom_set ((length () 0))  ! sequence length
    ((length (% $) (s %n)) (length ($) %n))).
```

This last axiom generates valid expressions such as (length (a b c) (s (s (s 0)))).

The mapping of a relation or function to lists of arguments is easily defined. First we need a utility that generates sequences of zero-or-more copies of an expression:

```
(zero_or_more % ()).
(zero_or_more % (% $))< (zero_or_more % ($)).
```

We also need a utility that distributes a sequence of elements over the fronts of sequences:

```
(distr () () ()).   ! distr elems over seqs
(distr (%el $els) (($seq) $seqs)
       ((%el $seq) $seqsx))<
   (distr ($els) ($seqs) ($seqsx)).
```

Now we make use of symbol representation to map previously-defined functions and relations to sequences of arguments:

```
((` ($rel '*')) $nulls)<   ! empty arg seqs
  (zero_or_more () ($nulls)).
```

```
((` ($rel ’*’)) $argseqsx)<  !non-empty seqs
  ((` ($rel ’*’)) $argseqs),
  ((` ($rel)) $args),      ! relation to map
  (distr ($args) ($argseqs) ($argseqsx)).
```

The expression (` ($rel)) matches the symbol for the relation name and (` ($rel '*')) represents that symbol with an asterisk appended. These axioms generate valid expressions such as (day* (Sat Tue Tue)) and (append* ((a b) ()) ((c) (u v)) ((a b c) (u v))). Note that our mapping definition automatically applies to predicates of any arity. The higher-order property of axiomatic language is essentially the same as that of HiLog. That is, the language has higher-order syntax but the semantics are really first-order.

B. Metalanguage Examples

The higher-order property and string variables along with the absence of commas give axiomatic language its metalanguage property – its ability to imitate other languages. In this section we define the evaluation of nested functions in Lisp format. First we need decimal number representation:

```
(finite_set digit ”0123456789”).   ! digits
(index %set %elem %index)<  ! index for elems
  (finite_set %set ($1 %elem $2)),
  (length ($1) %index).
     ! -> (index digit ’2’ (s (s 0)))
(dec_sym (` (%digit)) %value)<
  (index digit %digit %value).
     ! -> (dec_sym 1 (s 0))  -- single digit
(dec_sym (` ($digits %digit)) %valuex)<
  (dec_sym (` ($digits)) %value),
  (index digit %digit %n),
  (length (* * * * *  * * * * *) %10),
  (times %value %10 %10val),
  (plus %10val %n %valuex).  ! multi digits
```

These axioms generate valid expressions, such as `(dec_sym 325 (s (s … (s 0)…)))`, which give the natural number value for a symbol of decimal digits. We use the length function to hide the representation for the natural number 10. Now we define the evaluation of nested expressions:

```
(eval (quote %expr) %expr).    ! identity fn
(eval %dec_sym %value)<        ! decimal num
  (dec_sym %dec_sym %value).
(eval (%fn $args) %result)<    ! eval func
  (eval* ($args) ($results)),  ! eval args
  (%fn $results %result).      ! func result
```

These axioms turn some of the previously-defined predicates into functions which can be applied in a Lisp-like manner:

```
(eval (times (length (append (quote (a b c))
                     (reverse (quote (e d)))))
             (plus 3 17)
      ) %value)
```

When this expression is used as a condition, the variable %value will get instantiated to the natural number representation for 100. Of course, these nested expressions only make sense when formed from predicates that are functions that yield a single result as the last argument.

A contrasting procedural-language example can be found in the original paper [4].

IV. Specification by Interpretation

We want to specify the external behavior of a program using a set of valid expressions. A program that maps an input file to an output file can be specified by an infinite set of symbolic expressions of the form

```
(Program <input> <output>)
```

where <*input*> is a symbolic expression for a possible input file and <*output*> is the corresponding output file. For example, a text file could be represented by a sequence of lines, each of which is a sequence of characters. A program that sorts the lines of a text file could be defined by valid expressions such as the following:

```
(Program (”dog” ”horse” ”cow”)   ! input
         (”cow” ”dog” ”horse”))  ! output
```

Axioms would generate these valid expressions for all possible input text files.

An interactive program where the user types lines of text and the computer types lines in response could be represented by valid expressions such as

```
(Program <out> <in> <out> <in> …
                    <out> <in> <out>)
```

where <*out*> is a sequence of zero or more output lines typed by the computer and <*in*> is a single input line typed by the user. Each Program expression gives a possible execution history. Valid expressions would be generated for all possible execution histories. This static set of symbolic expressions is interpreted to represent real inputs and outputs over time. This is a completely pure approach to the awkward problem of input/output in declarative languages [7] and avoids Prolog's ugly, non-logical read/write operations. Example programs can be found at the language website [3].

V. Novelty and Related Work

This section discusses some of the more novel aspects of axiomatic language in comparison to Prolog and other languages:

(1) specification by interpretation – We specify the external behavior of programs by interpreting a static set of symbolic expressions. Even languages with "declarative" input/output [9] have special language features, but axiomatic language has no input/output features at all – just interpretation of the generated expressions.

(2) definition vs. computation semantics – Axiomatic language is just a formal system for defining infinite sets of symbolic expressions, which are then interpreted. Prolog semantics, in contrast, are based on a model of computation.

(3) Lisp syntax – Axiomatic language, like some other logic programming languages (MicroProlog [10], Allegro [11]), uses Lisp syntax. This unified representation for code and data supports metaprogramming.

(4) higher-order – Predicate and function names can be arbitrary expressions, including variables, and entire predicates can be represented by variables. This is the same as in HiLog, but with Lisp syntax. The XSB [12] implementation of HiLog,

however, does not allow variables for head predicates in clauses [13]. Higher-order programming can be done in standard Prolog, but requires special features, such as the 'call' predicate. [14]

(5) non-atomic characters – Character representation is not part of the core language, but defined as a syntax extension. There are no built-in character functions, but instead these would be defined in a library.

(6) non-atomic symbols – Non-atomic symbols eliminate the need for built-in decimal numbers, since they can be easily defined through library utilities.

(7) flat sequences – Sequences in axiomatic language are completely "flat" compared with the underlying head/tail "dot" representation of Prolog and Lisp.

(8) string variables – These provide pattern matching and metalanguage support. String variables can yield an infinite set of most-general unifications. For example, ($ a) unifies with (a $) with the assignments of $ = '', 'a', 'a a', ...

(9) metalanguage – The flexible syntax and higher-order capability makes axiomatic language well-suited to meta-programming, language-oriented programming [15], and embedded domain-specific languages [16]. This language extensibility is similar to that of Racket [17], but axiomatic language is smaller.

(10) no built-in arithmetic or other functions – The minimal nature and extensibility of axiomatic language means that basic arithmetic and other functions are provided through a library rather than built-in. But this also means that such functions have explicitly defined semantics and are more amenable to formal proof.

(11) explicit definition of approximate arithmetic – Since there is no built-in floating point arithmetic, approximate arithmetic must also be defined in a library. But this means symbolically defined numerical results would always be identical down to the last bit, regardless of future floating point hardware.

(12) negation – In axiomatic language a form of negation-as-failure could be defined on encoded axioms.

(13) no non-logical operations such as cut – This follows from there being no procedural interpretation in axiomatic language.

(14) no meta-logical operations such as var, setof, findall – These could be defined on encoded axioms.

(15) no assert/retract – A set of axioms is static. Modifying this set must be done "outside" the language.

VI. CONCLUSIONS

A tiny logic programming language intended as a pure specification language has been described. The language defines infinite sets of symbolic expressions which are interpreted to represent the external behavior of programs. The programmer is expected to write specifications without concern about efficiency.

Axiomatic language should provide increased programmer productivity since specifications (such as the earlier sort definition) should be smaller and more readable than the corresponding implementation algorithms. Furthermore, these definitions should be more general and reusable than executable code that is constrained by efficiency. Note that most of the axioms of this paper could be considered reusable definitions suitable for a library. Axiomatic language has fine-grained modularity, which encourages the abstraction of the general parts of a solution from specific problem details and minimizes boilerplate code. The metalanguage capability should enable programmers to define a rich set of specification tools.

The challenge, of course, is the efficient implementation of the programmer's specifications. [18] Higher-order definitions and the metalanguage capability are powerful tools for software specification, but make program transformation essential. The difficult problem of transformation should be helped, however, by the extreme simplicity and purity of the language, such as the absence of non-logical operations, built-in functions, state changes, and input/output. Future work will address the problem of transformation.

REFERENCES

[1] L. Prechelt, "An empirical comparison of seven programming languages," IEEE Computer, vol. 33, no. 10, pp. 23-29, October 2000.

[2] U. Wiger, "Four-fold increase in productivity and quality", Ericsson Telecom AB, 2001.

[3] http://www.axiomaticlanguage.org

[4] W. W. Wilson, "Beyond Prolog: software specification by grammar," ACM SIGPLAN Notices, vol. 17, #9, pp. 34-43, September 1982.

[5] W. Chen, M. Kifer, D. S. Warren, "HiLog: a foundation for higher-order logic programming," in J. of Logic Programming, vol. 15, #3, pp. 187-230, 1993.

[6] J. Backus, "Can programming be liberated from the von Neumann style? A functional style and its algebra of programs," CACM, vol. 21, #8, pp. 613–641, August 1978.

[7] S. Peyton Jones, "Tackling the Awkward Squad: monadic input/output, concurrency, exceptions, and foreign-language calls in Haskell", Engineering Theories of Software Construction, ed. T. Hoare, M. Broy, R. Steinbruggen, IOS Press, pp. 47-96, 2001.

[8] A. Pettorossi, M. Proietti, R. Giugno, "Synthesis and transformation of logic programs using unfold/fold proofs," J. Logic Programming, vol. 41, 1997.

[9] P. Wadler, "How to declare an imperative," ACM Computing Surveys, vol. 29, #3, pp. 240-263, September, 1997.

[10] K. L. Clark, Micro-Prolog: Programming in Logic, Prentice Hall, 1984.

[11] Allegro Prolog, http://www.franz.com/support/documentation/8.2/doc/prolog.html.

[12] The XSB Research Group, http://www.cs.sunysb.edu/~sbprolog/index.html.

[13] D. S. Warren, K. Sagonas, private communication, 2000.

[14] L. Naish, "Higher-order logic programming in Prolog," Proc. Workshop on Multi-Paradigm Logic Programming, pp. 167-176, JICSLP'96, Bonn, 1996.

[15] M. Ward, "Language oriented programming", Software Concepts and Tools, vol. 15, pp. 147-161, 1994.

[16] P. Hudak, "Building domain-specific embedded languages," ACM Computing Surveys, vol. 28, #4es, December 1996.

[17] S. Tobin-Hochstadt, V. St-Amour, R. Culpepper, M. Flatt, M. Felleisen, "Languages as Libraries", PLDI'11, 2011.

[18] P. Flener, Achievements and Prospects of Program Synthesis, LNAI 2407, pp. 310-346, 2002.

SciprovMiner: Provenance Capture using the OPM Model

Tatiane O. M. Alves, Wander Gaspar, Regina M. M. Braga, Fernanda Campos
Master's Program in Computer Science
Department of Computer Science
Federal University of Juiz de Fora, Brazil
{tatiane.ornelas, regina.braga, fernanda.campos}@ufjf.edu.br, andergaspar@gmail.com

Marco Antonio Machado, Wagner Arbex
National Center for Research in Dairy Cattle
Brazilian Enterprise for Agricultural Research - EMBRAPA
Juiz de Fora, Brazil
{arbex, machado}@cnpgl.embrapa.br

Abstract—**Provide historical scientific information to deal with loss of knowledge about scientific experiment has been the focus of several researches. However, the computational support for scientific experiment on a large scale is still incipient and is considered one of the challenges set by the Brazilian Computer Society for the period 2006 to 2016. This work aims to contribute in this area, presenting the SciProvMiner architecture, which main objective is to collect prospective and retrospective provenance of scientific experiments as well as apply data mining techniques in the collected provenance data to enable the discovery of unknown patterns in these data**.

Keywords-web services; ontology; provenance; OPM; OPMO; data mining.

I. Introduction

The large-scale computing has been widely used as a methodology for scientific research: there are several success stories in many domains, including physics, bioinformatics, engineering and geosciences [1].

Although scientific knowledge continue to be generated in a traditional way, in-vivo and in-vitro, in recent decades scientific experiments began to use computational procedures to simulate their own execution environments, giving origin to in-virtuo scientific experimentation. Moreover, even the objects and participants in an experiment can be simulated, resulting in in-silico experimentation category. These large-scale computations, which support a scientific process, are generally referred to as e-Science [1].

The work presented in this article, SciProvMiner, is part of a project developed in Federal University of Juiz de Fora (UFJF), Brazil, with emphasis on studies to building an infrastructure to support e-Science projects and has as its main goal the specification of an architecture for the collection and management of provenance data and processes in the context of scientific experiments processed through computer simulations in collaborative research environments geographically dispersed and interconnected through a computational grid. The proposed architecture must provides an interoperability layer that can interact with SWfMS (Scientific Workflows Management Systems) and aims to capture retrospective and prospective provenance information generated from scientific workflows and provides a layer that allows the use of data mining techniques to discover important patterns in provenance data.

The rest of the paper is structured as follows: In Section 2 we present the theoretical foundations that underpin this work. Section 3 describes some related work. Section 4 presents the contributions proposed in this work, the architecture of SciprovMiner is detailed, showing each of its layers and features. Finally, section 5 presents final considerations and future work.

II. Conceptual Background

Considering scientific experimentation, data provenance can be defined as information that helps to determine the historical derivation of a data product with regards to its origins. In this context, data provenance is being considered an essential component to allow reproducibility of results, sharing and reuse of knowledge by the scientific community [2].

Considering the necessity of providing interoperability in order to exchange provenance data, there is an initiative to provide a standard to facilitate this interoperability. The Open Provenance Model (OPM) [3] defines a generic representation for provenance data. The OPM assumes that the object (digital or not) provenance can be represented by an annotated causality graph, which is a directed acyclic graph, enriched with notes captured from other information related to the execution [3].

In the OPM model, provenance graphs are composed of three types of nodes [3]:

- Artifacts: represent an immutable state data, which may have a physical existence, or have a digital representation in a computer system.
- Processes: represent actions performed or caused by artifacts, and results in new artifacts.
- Agents: represent entities acting as a catalyst of a process, enabling, facilitating, controlling, or affecting its performance.

Figure 1 adapted from [3], illustrates these three entities and their possible relationships, also called dependencies. In Figure 1, the first and second edge in the right column express that a process used an artifact and that an artifact was generated by a process. These two relationships represent data derivation dependencies. The third edge in the right column indicates that a process was controlled by an agent. Unlike the other two mentioned edges, this is a control relationship. The first edge in the left column is used in situations where we do not know exactly which artifacts were used by a process, but we know that this process has used some artifact generated by another process. Thus, it can be said that the process is initialized by another process.

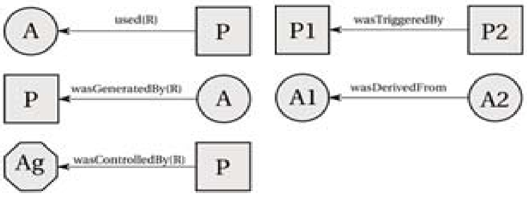


Figure 1. Edges in OPM: origins are effects and destinations are causes [adapted from [3]]

Similarly, the second edge in the left column is used in situations where we do not know what process generated a particular artifact, but we know that this artifact was derived from another artifact. These last two relationships are recursive and can be implemented considering inference rules. So, from them, it is possible to determine the sequence of execution of processes or the historical derivation that originated a data.

Some provenance models use Semantic Web technology to represent and to query provenance information. Semantic Web languages such as RDF and OWL provide a natural way to model provenance graphs and have the ability to represent complex knowledge, such as annotations and metadata[7]. One of the benefits of the semantic web approach is the ability to integrate data from any source in the knowledge base.

In [3] is defined an OWL ontology to capture concepts in OPM 1.1 version and the valid inferences in this model. Furthermore, this OWL ontology specifies an RDF serialization of the OPM abstract model. This ontology is called Open Provenance Model Ontology (OPMO).

III. RELATED WORKS

Recently, some works have been conducted to address challenges in data provenance considering scientific domain.

In [5], the author proposes an architecture for data provenance management called ProvManager. The SciProvMiner architecture as well as ProvManager is focused on capturing provenance in workflows orchestrated from heterogeneous and geographically distributed resources based on the invocation of web services. However SciProvMiner provides an infrastructure based on semantic web for provenance metadata representation and query. This feature gives to SciProvMiner a greater power of expression to infer new knowledge. ProvManager collects both prospective and retrospective provenance. SciProvMiner also collects prospective and retrospective provenance, with the differential that it uses OPM model to capture both retrospective and prospective provenance. The advantage of this approach is the interoperability of captured data, since it uses a standard model. Considering the data mining layer of SciProvMiner, ProvManager does not have.

In [4] is proposed an extension to OPM in order to model prospective provenance, in addition to retrospective provenance already supported in the native OPM model. The SciProvMiner also uses an OPM extension for supporting prospective provenance, however, the proposed architecture in [4] does not provide infrastructure based on semantic web - RDF, OWL ontologies, and inference engines, as SciProvMiner. This feature allows that the query engine of SciProvMiner can provides results that combine the power of data mining techniques together with inference engines acting over ontologies.

In [6] the authors present a mining method based on provenance data for creating and analyzing scientific workflows. SciProvMiner also uses mining methods applied to provenance data, but the focus is on unexpected knowledge discovery. Also, SciProvMiner uses ontologies together with data mining techniques.

IV. SCIPROVMINER- SCIENTIFIC WORKFLOW PROVENANCE SYSTEM MINER

In the context of e-Science, the emphasis on the conception and construction of SciProvMiner architecture consists of using semantic web resources to offer to researchers an environment based on interoperability for heterogeneous and distributed provenance management and query.

The SciProvMiner architecture representation considering a typical scenario is presented in Figure 2. In a collaborative environment interconnected through a computational grid, the experiments can be conducted jointly by groups of scientists located on remote research centers.

Within this scenario, researchers can model scientific workflows from different SWfMS and whose execution requires access to heterogeneous repositories and distributed in a computational grid. In this environment, the information stored lack mechanisms that contribute to a better interoperability between the data and processes generated and orchestrated within collaborative research projects. The SciProvMiner aims to provide the necessary infrastructure to make this interoperability possible.

The initial responsibility of SciProvMiner is to provide an instrumentation mechanism for the several components of the scientific workflow involved in conducting the collaborative experiment whose provenance data need to be collected for further analysis. In this context, an instrumentation mechanism is implemented using web services technology and manually configured for each component whose provenance must be collected. This mechanism aims to capture information generated during the workflow execution and send the metadata to a provenance repository managed by SciProvMiner.

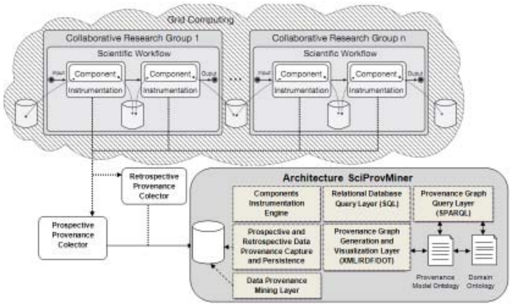

Grid Computing
Collaborative Research Group 1
Collaborative Research Group n
Scientific Workflow
Scientific Workflow
Input
Component
Instrumentation
Component
Instrumentation
Output
Input
Component
Instrumentation
Component
Instrumentation
Output
Retrospective Provenance Colector
Prospective Provenance Colector
Architecture SciProvMiner
Components Instrumentation Engine
Relational Database Query Layer (SQL)
Provenance Graph Query Layer (SPARQL)
Prospective and Retrospective Data Provenance Capture and Persistence
Provenance Graph Generation and Visualization Layer (XML/RDF/DOT)
Data Provenance Mining Layer
Provenance Model Ontology
Domain Ontology

Figure 2. SciProvMiner Architecture

Considering Figure 2, "Provenance Graph Generation and Visualization Layer" has several interrelated tasks. From the relational database, the information persisted according to OPM model are processed in order to obtain an in-memory representation of the provenance graph corresponding to each execution of the scientific workflow and also of the execution models of this workflow. This layer allows the construction of a visual representation of the generated provenance graph, considering both retrospective and prospective provenance. The "Provenance Graph Query Layer" is intended to provide a mechanism and an interface for the user to formulate queries. This layer is associated with the OPM ontology and with the ontology of the studied domain (for example, dairy cattle and its parasites domain). This resource gives to SciProvMiner the possibility to process queries using inference machines that provides the capability to make deductions about SciProvMiner's knowledge bases and gain important results considering that it extracts additional information beyond those that are explicitly recorded in the generated provenance graphs.

The "Data Provenance Mining Layer" performs the search for unknown and useful patterns in provenance data using data mining techniques together with the ontologies and inference mechanisms. With this layer we aim to achieve a higher quality of provenance data, identifying possible errors and increasing the reliability of the results in the execution of an experiment.

A. *Provenance Capture*

As emphasized, the SciProvMiner presents an approach for managing provenance data independently of SWfMS. This implies that SciProvMiner is responsible for gathering of generated provenance data. An important feature of the instrumentalization model of SciProvMiner refers to the adoption of a single web service to wrap any original component of an experiment modeled from a scientific workflow. The SciProvMiner web service can be invoked one or more times for each component of the original workflow. In a typical scenario, a first instance of the web service is configured to gather the input data (artifacts, according to the OPM model) to be processed on the original component. In a second instance, it can capture information related to the original component (a process, according to the OPM model) such as start and end of the run, causal dependencies related to input and output artifacts (needed for the explicit characterization of which component used the input data and generated the output data).

It is important to observe that can be used a greater number of instrumentalization web service instances for the same original component, depending on the need to collect other entities and causal dependencies of the OPM model identified in a workflow. By hypothesis, the instrumentalization in the form adopted by this work is possible in any Scientific Workflow Management System that supports web services based on SOAP standard.

Starting from an approach of provenance collection at a process level (component), it opens the possibility of control over the data granularity to be captured. In the domain of scientific experimentation, may be important for the researcher to analyze information and prepare data lineage queries at various levels of detail. In this scenario, it becomes important that the provenance model may be able to deal with provenance information at different levels of granularity. (The OPM provides this resource in a convenient from the entity called "description" (account).

It can also be observed that different provenance descriptions relating to the same execution of a scientific workflow, represent varied lineage semantics. Therefore, by provenance data collection at different levels of granularity is possible to make analyzes and queries over the provenance data, according to the particularities of the research in progress. In this scenario, it should be noted that the strategy of provenance collection and management at various levels of abstraction can be considered as an important differential of our work allowing the parameterization of the provenance query according to the specific focus of research.

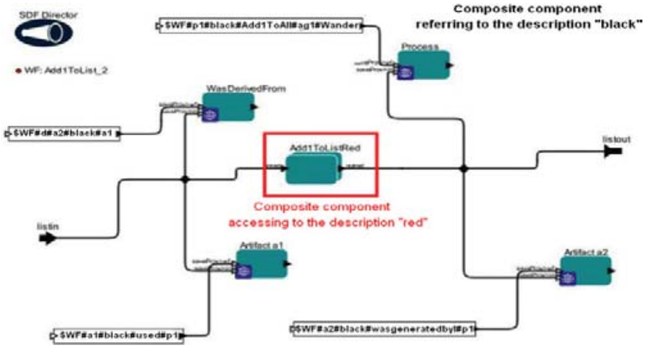

SDF Director
WF: Add1ToList_2
$WF#p1#black#Add1ToAll#ag1#Wander
Composite component referring to the description "black"
Process
WasDerivedFrom
$WF#d#a2#black#a1
Add1ToListRed
listout
listin
Composite component accessing to the description "red"
Artifact a1
Artifact a2
$WF#a1#black#used#p1
$WF#a2#black#wasgeneratedby#p1

Figure 3. Composite component modeled in SWfMS Kepler

Taking an example of a workflow modeled in Kepler and the capture of retrospective provenance, shown in Figure 3, the instrumentation mechanism built manually can be used to capture the provenance data according to the OPM model in a specific level of granularity, defined according to the interests of the researcher or of the research group involved with the scientific experiment under study.

If the scientific workflow is instrumentalized in more than one level of granularity from different descriptions, the subgraph for each account (description) is displayed in a different color in the generated provenance graph.

Considering the prospective provenance context and the importance to obtain all necessary information about provenance, the SciProvMiner architecture uses an OPM model extension as proposed in [6]. With this extension becomes possible to collect information such as the description of the workflow, the computational tasks that are part of the workflow, the subtasks of a task, the agent who performs a task and the input and output ports of a task, where the output port of a task can be connected to input port of another task, characterizing the flow of data. These informations are persisted in Relational Database Management System (RDBMS), together with information about retrospective provenance.

Due to the fact that one of the main features of SciProvMiner is the use of Semantic Web resources for representation and query of provenance data, an extension of the OPMO ontology that includes prospective provenance is currently being detailed.

V. DATA MINING

The task of data mining seeks to extract information that are hidden in large databases, which are previously unknown and potentially useful. Generally, the knowledge discovered by data mining processes is expressed in the form of rules and patterns.

According to [6], mining processes in scientific workflows are very valuable, because of scientific experiments are exploratory in nature and changes are constant.

Considering the context of SciProvMiner, data mining techniques can be applied for the following categories:

- Descriptive: whose objective is to find patterns that describe characteristics of a segment or a group of data. The main descriptive tasks are Extraction of Association Rules and Clustering, In the context of the prospective provenance data, this category is useful in order to help in the validation of workflow models and assist in the replacement of workflow models that follow similar pattern in the context of a given experiment.
- Predictive: make inferences about the existing data to build models that will be used as predictive tools. Classification and Regression may be cited as the main predictive tasks. This category is particularly interesting in the context of retrospective provenance, considering the OPMO ontology and the possibilities of connection between the data.

In this context, data mining layer of SciProvMiner acts regarding as these two categories, providing a query mechanism where data mining techniques are considered, together with OPMO and inference mechanisms. Currently, OPMO ontology is being extended to encompass specific needs of data mining techniques in order to improve data treatment.

Whereas provenance in scientific experimentation is important to help the scientist to evaluate the quality of the data, the validity of the results and allow reproducibility of the experiments, it is important to provide tools to scientist that offers more knowledge about the experiments.

VI. FUTURE WORK

In 1995, Embrapa Dairy Cattle has initiated a research project that has as main objective to obtain, by means of biological approaches, zootechnical statistics and more recently, through bioinformatics, genetic markers that, among other things, explain the existence of genetically resistant parasites and also seeking information about genetic characteristics that confer low-irritability to animals.

These studies are motivated by the importance they have and the expected production of these animals, since they are responsible for phenotypical characteristics related to animal behavior, and also the influence of this behavior on their production.

In tropical areas, the infestation of bovine animals by endo and ectoparasites causes a reduction in the productivity of animals, leading, in extreme cases, to the death of them. This issue is particularly important for Brazil, since the country has the largest commercial cattle in the world, recording in 2010 about 209 million cattles [8].

In this context, it is indispensable the collection and management of provenance data and processes of scientific experiments which will assist in the perception of historical data derivation, being necessary to the reproducibility of results obtained.

This way, the use of SciProvMiner is being inserted into this project because it is considered important to assist in the management and discovery of new patterns considering the data generated by the project.

REFERENCES

[1] S. C. Wong, S. Miles, W. Fang, P. Groth, and L. Moreau, "Provenance-based Validation of E-Science Experiments," In: 4th International Semantic Web Conference (ISWC), Galway, Ireland, 2005, pp. 801-815.

[2] J. Freire, D. Koop, E. Santos, C. T. Silva,"Provenance for Computational Tasks: A Survey", Computing in Science and Enginnering, vol. 10, n. 3, pp. 11–21, 2008.

[3] L. Moreau et al, "The Open Provenance Model core specification (version 1.1)", Future Generation Computer Systems, vol. 27 Issue 6, pp.743 –756, 2011.

[4] C. Lim, S. Lu, A. Chebotkot, F. Fotouhi,"Prospective and retrospective provenance collection in scientific workflow environments", Proceedings - 2010 IEEE 7th International Conference on Services Computing, SCC 2010, art. n. 5557202, pp. 449-456, 2010.

[5] A. Marinho et al., "Integrating Provenance Data from Distributed Workflow Systems with ProvManager", in International Provenance and Annotation Workshop - IPAW, Troy, NY, USA, pp. 0-3, 2010.

[6] R. Zeng, X. He, J. Li, Z. Liu, V.D. Aalst, "A Method to Build and Analyze Scientific Workflows from Provenance through Process Mining", 3rd USENIX Workshop on the Theory and Practice of Provenance, 2011.

[7] W. Gaspar, R. Braga, R. Campos, "SciProv: An architecture for semantic query in provenance metadata on e-Science context", 2nd International Conference on Information Technology in Bio- and Medical Informatics, ITBAM 2011, Toulouse, pp 68-81, 2011.

[8] IBGE,http://www.sidra.ibge.gov.br/bda/tabela/listabl.asp?c=73&z=t&o=24, 2011.

Engineering Graphical Domain Specific Languages to Develop Embedded Robot Applications

Daniel B. F. Conrado and Valter V. de Camargo
Computing Department – Federal University of São Carlos (UFSCar)
São Carlos, Brazil
{daniel_conrado,valter}@dc.ufscar.br

Abstract—**Over the last years, little attention has been paid in software engineering techniques for improving the productivity of embedded robot application development. The complexity of these applications has been continuously growing and they are presenting challenges that are uncommon to information systems' development. Therefore, any technique that can support their development is a great contribution. Domain specific languages (DSLs) are one technique that aims at improving productivity by reusing concepts and abstractions from a specific domain. In this paper we present a DSL engineering process for embedded robot applications. The aim is to make the activity of creating DSLs to this domain more systematic and controlled. As a result, one can build embedded robot applications in a more productive way. An important characteristic of our process is that it asks for just one application to reach a first version of a running DSL. We also present a generic language model that may serve as a foundation for future DSLs. In order to validate our process, we have applied it to the development of a DSL from an aisle monitoring robot application.**

Keywords-component; Domain-Specific Languages; Mobile Robots; DSL Engineering

I. INTRODUCTION

Domain-specific languages (DSLs) are small programming languages which targets a specific domain. This domain may be vertical, i.e. application domains like healthcare, engineering, home security, etc. or horizontal, that is, technical domains that provide structures to build applications, like persistence, signal processing, security, among others [1].

One of the main advantages of DSLs is the possibility of writing applications using domain expressions instead of programming languages' constructs. It raises the abstraction level and also improves code reuse. DSLs can be differentiated by their appearance (textual or graphical) and their origin (internal or external). The difference between external and internal DSLs is that the latter are embedded into another general purpose language (GPL) called the host language, and that's why such DSLs are often called embedded languages.

In general, DSLs increases expressiveness significantly when compared to a GPL [3]. It allows the developers to write applications with fewer instructions that are easier to comprehend. DSLs are also used in the context of Model-Driven Development (MDD) by specifying high-level models that are transformed into source code artifacts (Model-to-Code transformation) or detailed models (Model-to-Model transformation) [1,4].

Robots are electromechanical machines that perform repetitive and/or dangerous tasks in an environment. They extract environment's information using sensors; calculate actions based on that information by means of processing units and perform those actions with end effectors (devices like gripper and arm). In short, robots are composed by sensors, actuators and processing units. The latter generally are a platform-specific software embedded into one or more microcontrollers [6,7,8].

In this context, one may note that the way the robot wheels are configured, its sensors, microcontrollers, physical structure and even its environment influences its programming. Furthermore, there are a lot of non-functional properties that must be considered such as power consumption, response time, safety, among others. As we said, most of these issues are uncommon to or quite different from information systems.

Although there are some processes to build DSLs [3,9,10,11], we didn't find one that takes into consideration specific characteristics of mobile robots. With this lack, these processes may yield poorly designed DSLs which compromises the quality of generated applications. Using DSLs in the context of robot applications may enhance their overall development, since robotics is a complex domain and high-level representations combined with model/code generation would contribute for better understanding and code reuse.

In this paper we present a process for building DSLs which target to the r obot application domain. Moreover, although domain engineering is generally performed by analyzing at least three applications of a certain domain, there may be situations where there is an interest in building DSLs even though such applications don't exist yet. In this context, our aim is to c ontribute to robotic software development with an alternative process that provides guidelines for building initial DSLs based on a single application. The main idea is that such initial DSLs may eventually be evolved to the extent that new applications are developed and new abstractions are identified. As a proof of concept, we developed a DSL based on security applications implemented with a LEGO Mindstorms kit and the LeJOS API [5].

This paper is organized as follows: section II presents our process; section III presents our proof of concept; section IV presents related works, and section V concludes the paper.

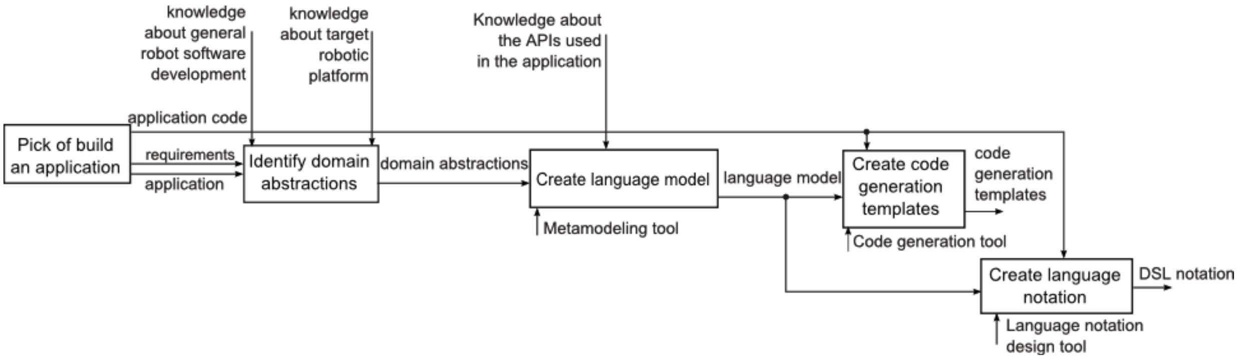

knowledge about general robot software development
knowledge about target robotic platform
Knowledge about the APIs used in the application
application code
Pick of build an application
requirements
application
Identify domain abstractions
domain abstractions
Create language model
Metamodeling tool
language model
Create code generation templates
code generation templates
Code generation tool
Create language notation
DSL notation
Language notation design tool

Figure 1. The process for engineering DSLs for mobile robot applications

II. A Process For Engineering DSLs

In this section we present our process, which has five activities. We detail them below. The process overview can be seen as a SADT diagram in Fig. 1. The main focus of this paper is on the second and third activities because of space limitations.

Pick or build an application. In this activity, the domain engineer shall builds a robot application or take an existi ng one. He/she must have knowledge about its requirements and the software libraries used in the implementation.

Identify domain abstractions. This activity is divided in two steps which are explained below.

Extract domain abstractions from requirements. In this step, the domain engineer shall analyze the application's requirements to extract domain abstractions, which will comprise the DSL abstractions. This activity consists of structuring the application's requirements as a tree, as can be seen in Fig. 2, where the first node represents the application's main objective. The first level of children nodes is called "What to do" level, since they state what the robot must perform in order to achieve the main objective. The second level is called "How to do" level and its nodes describe what the robot does to perform what is stated in their parent nodes. The "How to do" nodes shall be created considering how actuators and sensors are utilized to perform such action. For example, if the main objective is following a wall, there may be a first level node stating "walk near the wall", and it could have children nodes stating "walk" and "keep parallel to wall". A node may also have children nodes in the same level.

After creating this tree, which is the resulting artifact from this activity, the domain engineer shall choose which abstractions will comprise the intended DSL. If he/she chooses the leaf nodes, the resulting DSL will be more flexible but more verbose and less related to the application domain. But if the "What to do" nodes is chosen, the DSL will be closely related to the application domain and will be more expressive but too rigorous and limited. It's also possible to choose abstractions from different node levels—usually, the more abstract ones results in less flexible DSL elements.

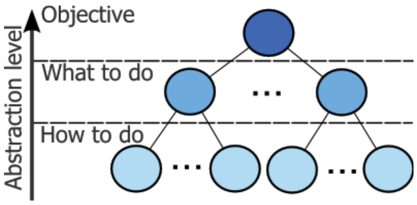


Figure 2. Illustration of requirements structured as a tree

Classify domain abstractions into concerns. In this step, one shall create a table categorizing the chosen abstractions into *concerns*. These concerns must be related to the mobile robots' domain. The most common concerns are domain-independent, for example, *Movement*, *Communication* and *Recognition*. However, the domain engineer can elect domain-specific concerns as well. For example, abstractions that state the robot must move, like following a line or a path, should belong to Movement concern, yet abstractions such as finding the best path should be classified into the Planning concern, and so on. The resulting artifact of this activity is called Table of Domain Abstractions and it is the input for the next activity.

Create language model. In this activity, the domain engineer shall iterate the concerns and for each of them he/she identifies common and variant parts of its abstractions. Then these parts are added to t he *Language model*. Here, the engineer must know the APIs used in the implementation of the application, as we mentioned earlier. It is important to know how the physical parts of the robot, such as sensors and motors, are programmatically represented, because the Language Model should be aware of implementation details, even though they will not be visible to developers. These details are important for code generation issues.

As a r esult of our study, we created a generic language model that could be used as a foundation for new mobile robot DSLs. That is, the domain engineer could simply extend our generic language model to fit his/her needs.

Our generic language model is shown in Fig. 3. The main entity is Robot. It contains behaviors, belief states, sensors and variables. This model is based on the Behavior model [6] and its way of programming robots consists roughly of specifying behaviors for the robot, and each of them has a condition to be executed. Just one behavior will run at a time—the one whose condition is satisfied at the moment. In our model, the conditions are specified as belief states; the expression property is intended to contain comparisons of sensors' data and variable values. The expression may use functions of sensors and variables. For example, an ultrasonic sensor may have the "distance" function which returns a float value. The domain engineer may also create other specialized entities having algorithms such as the Gaussian filter, Proportional-Integral-Derivative (PID) controller, etc.

A DSL may have multiple diagram kinds. When creating the language model, the domain engineer must know what kind of diagrams the DSL should have. The engineer should look at the Language Model in order to analyze if all entities can be addressed into only one diagram. If some entities cannot stay together, either because they are totally independent from each other or they could yield polluted diagrams, then the domain engineer should separate them into different diagrams. When separating entities into diagrams, there should be a balance between correlated entities and diagram pollution.

Create code generation templates. In this activity, the domain engineer shall implement templates that are roughly fixed code with variation points. A domain framework could be created as well. It serves as a foundation for the generated code and provides artifacts that are common to all applications that can be generated.

The process' control flow may go back to the prior activity "Create Language Model", because changes may be required in the language model depending on the code generation technology used. In Java Emitter Templates framework (JET), for example, a model is traversed as a graph. Commands that traverse instances of a particular entity cannot guarantee the order they are processed. If the order is a must, the engineer should change the language model to assure it, for example, by adding an initial node that connects to the first instance and adding a relationship between instances that specifies the *next* to be processed. The process' control flow may also go back to the activity "Identify domain abstractions", because during template confection, it is common to note missing or even incorrect domain abstractions.

Create language notation. The language notation is its concrete syntax, that is, how it visually represents its elements, relationships and properties. These representations must be meaningful to the users of the DSL. Choosing the most appropriate visual representations contributes significantly to the DSL success; however, this is beyond the scope of this paper.

III. Proof of Concept: A DSL for the LEGO Mindstorms

In this section, we describe how we conducted our process to create a DSL that generates code for LEGO Mindstorms.

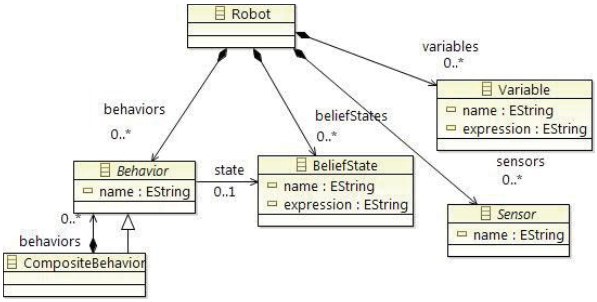


Figure 3. The generic language model

Pick or build an application. Our mobile robot application has the following requirements: i) It must seek for opened doors along an aisle; ii) a sound alert shall be played when it finds one; and iii) when it reaches the end of the aisle, the robot should turn back and begin seeking again. We implemented our application using the LeJOS API [5].

Identify domain abstractions. We extracted the domain abstractions and built the tree shown in Fig. 4. We divided the main objective into three nodes: "walk along aisle", "detect open doors" and "sound alert if door is open".

After extracting the domain abstractions, we shall choose which ones will comprise the DSL. As mentioned before, one could pick abstractions from different levels—at the bottom lies the less abstracted ones and hence more flexible. We chose all from the "How to do" level but the "play sequence of tones". Instead we chose its parent node. The next step is to categorize the chosen abstractions into concerns. We classified into Movement those that manipulate the motors, and into Detection those that use the sensors. The one that left was classified into the Communication concern since its goal is to notify someone about an opened door.

Create language model. We created the language model shown in Fig. 5. There is the generic language model extended with our DSL entities, which are in bold. We also added properties to the generic entities, which are underlined.

Create code generation templates. We used the Java Emitter Templates Framework (JET) to create the templates for code generation. The model built with the DSL is passed as an input to the templates and their tags traverse its elements to extract and process information, producing application's code. We also created a domain framework that implements all the behaviors, for the generated code only instantiates and configures what is modeled. The process of creating the domain framework reuses a significant part of the application developed in the first activity.

Create language notation. In this activity, we created the notations of the DSL. To make it simple, choosing the best notations is out of the scope of our proof of concept.

IV. Related Work

Günther et al. [9] show an agile and lightweight process for engineering DSLs embedded in dynamic languages. The process is divided into three phases: *Domain Design*, *Language Design* and *Language Implementation*. They also present several DSL-engineering patterns to be used in the last phase, which are strongly related to the following DSL engineering concerns: *Language Modeling* (what language constructs implement domain concepts), *Language Integration* (how to easily integrate the DSL with other components) and *Language Purification* (how to optimized e.g. readability).

Robert et al. [10] define a lightweight process for designing UML profiles. The process has three activities: *Problem description*, *Refinement restriction* and *Profile definition*. Their resulting artifacts are respectively a *Problem model*, a *Domain model* and a *Profile*. There are two actors: the *domain expert* and the *language expert*. The former is responsible for the first activity, and the latter, for the last one; they both work in the

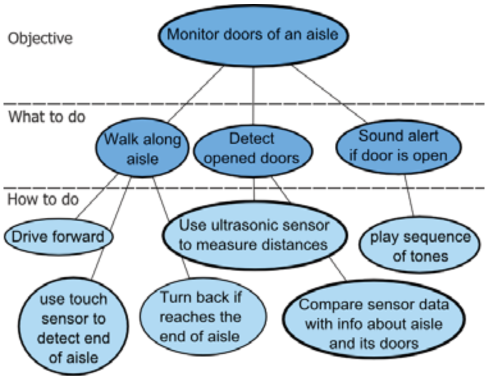

Objective
Monitor doors of an aisle
What to do
Walk along aisle
Detect opened doors
Sound alert if door is open
How to do
Drive forward
Use ultrasonic sensor to measure distances
play sequence of tones
use touch sensor to detect end of aisle
Turn back if reaches the end of aisle
Compare sensor data with info about aisle and its doors

Figure 4. The abstractions extracted from the applications' requirements

remaining (middle) activity. The proposed process also employs a set of predefined heuristics to automatically generate an incomplete profile from the domain model. This profile is then optimized using several guidelines defined by the authors. These guidelines are intended to ensure the correctness of profiles and to optimize them.

Based on the experience gained from multiple different DSL development projects and prototyping experiments, Strembeck and Zdun [11] have proposed a systematic DSL development process composed by a set of activities and sub-activities. The process has four main ones: *Defining the DSL's core language model*, *Defining the behavior of DSL language elements*, *Defining the DSL's concrete syntax(es)* and *Integrating DSL artifacts with the platform/infrastructure*. These activities typically outputs the following artifacts, respectively: the core language model, the behavior definition, the concrete syntax(es) and the transformation rules. The first three artifacts compound the DSL while the last one is related to the development platform. Those activities have sub-activities and they all have (not rigidly) defined control flows. They also present activities to tailor the process to "fit into the corresponding organization's standard development approach", relating project type, involved people, budget, among other factors.

The main differences between our work and those described above are twofold: our work focuses on graphical and external DSLs instead, and it is specific to mobile robots domain. Moreover, the aforementioned processes are too generic regarding to application domains.

V. Final Remarks

In this paper we present our efforts towards a process for engineering DSLs in the context of mobile robots. The resulting DSLs are initial running version since the process input is just one application. However, such DSLs are powerful enough to model other different ones. As long as a DSL is in use, missing abstractions may be identified, and they could be added to the language model. We also provide a generic language model, where the domain engineer can simply extend by adding entities representing (possibly domain specific) behaviors, specialized functions to sensors, among others. As our generic language model is based on the well-known Behavior model, the created DSLs may evolve easily.

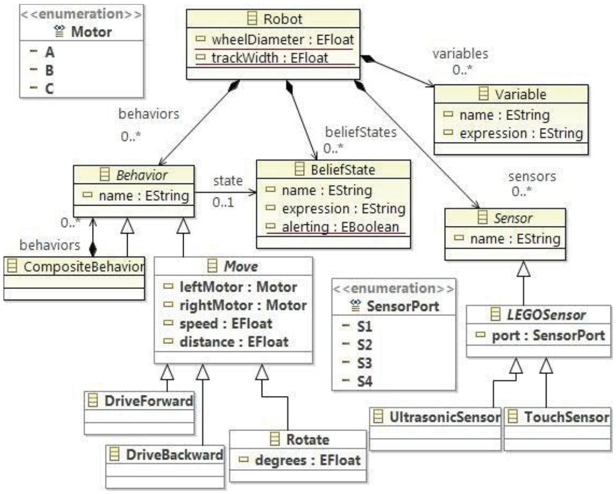


Figure 5. The Language Model of the DSL for LEGO Mindstorms

Finally, we present a proof of concept where we applied our process to obtain a DSL for developing robot applications to the LEGO Mindstorms platform. We believe our DSL is more expressive than code since the developer does not need to be aware of implementation details, like libraries, unity conversion, programming languages issues, etc.; he/she just manipulates concepts from the target domain.

As a future work, we are aiming to improve the process by applying it on different applications implemented in different robotic platforms. The next step will be to implement different applications into the Pioneer 3-DX mobile robot platform, and then emerge DSLs with our process. We are aiming to improve the identification of abstractions to address more mobile robot concepts. Another future work is to design a consistent process or adapt the existing one to address the evolution of those initial DSLs while they are in use and new abstractions arise.

References

[1] S. Kelly and J. P. Tolvanen, Domain-specific modeling. IEEE Computer Society, 2008.

[2] OMG, "SysML: Systems Modeling Language", www.omgsysml.org.

[3] M. Mernik, J. Heering and A. M. Sloane, "When and how to develop domain-specific languages", ACM Computing Survey, 2005.

[4] T. Stahl and M. Völter, Model-Driven Software Development. Wiley, 2006.

[5] LeJOS, "LeJOS, Java for Lego Mindstorms", lejos.sourceforge.net.

[6] R. C. Arkin, Behavior-based robotics. Series: Intelligent robots and autonomous agents. MIT Press, 1998.

[7] T. Bräunl, Embedded robotics: mobile robot design and applications with embedded systems. 3rd ed. Springer, 2006.

[8] R. Siegwart and I . Nourbakhsh, Introduction to autonomous mobile robots. MIT Press, 2004.

[9] S. Günther, M. Haupt and M. Splieth, "Agile engineering of internal domain-specific languages", In: "Proceedings of the fifth international conference on software engineering advances", 2010.

[10] S. Robert, S. Gérard, F. Terrier and F. Lagarde, "A lightweight approach for domain-specific modeling languages design", In: "Proceedings of the 35th Euromicro conference on software engineering and advanced applications", 2009.

[11] M. Strembeck and U. Zdun, "An approach for the systematic development of domain-specific languages", "Software: Practice and Experience", vol. 39, pp.1253-1292, 2009.

Dynamically recommending design patterns

S. Smith, D. R. Plante
Department of Mathematics and Computer Science
Stetson University
421 N. Woodland Boulevard
DeLand, FL 32723, USA
Email: sarahgrace89@gmail.com, dplante@stetson.edu

***Abstract*—Recommendation Systems for Software Engineering are created for a variety of purposes, such as recommending sample code or to call attention to bad coding practices (code smells). We have created a system to recommend the use of design patterns. While many programmers have knowledge of design patterns, whether rushed to meet deadlines, inexperienced in their implementations, or unaware of a particular pattern, pattern implementation may be overlooked. We have developed a tool to dynamically search for signs that a programmer would benefit by using a particular design pattern and make the appropriate recommendations to the programmer during code development.**

I. Introduction

Code reuse is a common practice used to improve the development process by providing well tested elements which the programmer can incorporate into his or her system. Similarly, design patterns encourage the reuse of object oriented ideas [6]. They provide design solutions to common problems, but must be implemented specifically for each project.

Anti patterns attempt to prevent common mistakes which can degrade the quality of an object oriented system [2]. Each pattern defines a "bad way" of structuring code and suggests methods for refactoring. Anti patterns are not directly related to design patterns in that there are not necessarily refactorings for the anti patterns that turn them into design patterns. The suggestions simply create more object oriented code.

Design pattern and anti pattern discovery assists programmers in the software development process and is currently an active area of research. By identifying design patterns in a developed system, future programmers are encouraged to maintain those patterns. Discovering anti patterns can alert programmers to problem areas so that they can be fixed.

We have developed a tool which, instead of finding instances of either type of pattern, recommends the use of design patterns based on an unfinished project. We determine that a programmer is trying to solve a common problem in a way that could be improved using a design pattern and dynamically make recommendations. We have created a framework for detection and a format of storing requirements for each of these anti design patterns. The requirements will then be processed and our tool, developed as an Eclipse plugin, will search for instances within the current project. The plugin will also determine what to make recommendations about and when to make them. Although the current tool only recommends a few patterns, it is designed so that the set can be easily expanded when new "anti design patterns" are included.

II. Related Work

A. Design Pattern Detection

Brown [1] made the first attempt at automatically detecting design patterns. Since then, research has focused on both defining design patterns in a manner that is programmatically useful and identifying matches in source code. Some methods only look for structural characteristics, such as the relationships between classes, while others focus more on specific behaviors and how the classes actually interact. Dong *et al.* [11] provide an excellent review of design pattern mining techniques presently implemented. As noted previously, we focus on matching anti patterns.

B. Anti Pattern and Code Smell Detection

Anti patterns were originally defined by Brown *et al.* [2] as recurring solutions to common problems which have negative consequences. Similarly, Fowler [5] informally defines code smells, which are also examples of bad programming but with less complexity. Searching for anti patterns (or any type of code "smells") is very useful to programmers who do not have the time or resources to analyze large systems.

Most anti pattern detection methods use metrics. Once the metrics are defined for a specific pattern, objects in a system are tested and potential matches are returned. Marinescu [14][15] created detection strategies for finding anti patterns. However, a strategy was created separately for each pattern, making it difficult to expand to new patterns. Munro [16] formally defined rule cards for nine design flaws and tested each one. Salehie *et al.* [21] found "hot spots," in addition to known anti patterns, when certain metric values were outside the range found in good code. These areas could either match to specific design smells or be indicators of a problem that was not formally defined. Moha *et al.* [19] also developed a system of rule cards and tested it on four anti patterns.

Exact matching using metrics does not allow for a high level of flexibility and leaves the software analyst with a list of possible design anti patterns and no method of prioritizing which to consider first. Khomh *et al.* [4] investigated the use of Bayesian belief networks for anti-pattern detection. This produced a probability that a given class or set of classes followed an anti pattern, which was a more realistic way to detect something which could not be exactly defined. Similarly, Oliveto *et al.* [20] used B-splines to detect anti patterns and gave results in terms of probabilities.

These methods focus on metrics of individual classes and do not take into account the relationships between classes or specific behavior. This is logical for the anti patterns described by Brown, as the majority of them reflect a failure to follow object-oriented standards, and therefore have minimal relationships to consider.

C. *Recommendations Systems for Software Engineering*

Recommendation systems for software engineering aim to assist programmers in the software development process by making recommendations based on written code and/or dynamic analysis [18]. They can make recommendations dynamically or by the request of the programmer. These systems offer assistance on a variety of topics, from what to consider changing next [23] to examples of and suggestions for what call sequence to make [8][22]. Guéhéneuc *et al.* [7] created a design pattern recommender based on words chosen to describe the programmer's needs.

III. Methods

Any code pattern can be defined in terms of three characteristics: structure, behavior, and semantics [11]. The structure refers to the types of classes and relationships between them. A UML class diagram, for example, contains mostly structural information about a system, specifying the inheritance, association, and other relationships between classes. These relationships are found by looking for certain types of references; for example, a field declaration is an aggregation, an "extends" in the class signature is a generalization, and any other reference is considered to be an association.

The behavioral characteristics used to define a pattern are more complex and often abstract. They define how objects are created, methods are invoked, and information is shared. They describe what a piece of code actually does, not just the type of relationships or objects. We match behavioral characteristics by defining specific ways they could be implemented and searched for similar code structures.

The actual choice of names for classes, methods, and other parts of a system make up the semantic data. Semantic data can be used to look for repetition of names or the use of actual words and their synonyms to name or describe parts of the system. Our system does not use semantic data in searching for matches as we are not matching design patterns but rather anti patterns.

In order to limit recommendations to code which the programmer is currently working on, our matching algorithm is triggered when a class has been modified, using only the modified class and the classes "close" to it. Here we define classes to be "close" to the modified class if they may be reached by traversing at most N relations from it, where N is chosen by the developer as a configuration setting. We first look for a structural match to a given anti pattern. If we find a match, we then test the behavioral characteristics for those classes. If all the behavioral requirements are met, we make a recommendation to the programmer.

A. *Intermediate Code Representation*

As noted by Dong *et al.* [11], rather than working with source code directly, most pattern search algorithms use some form of intermediate code representation. The Abstract Syntax Tree (AST) is a directed acyclic graph, where each node represents a programming element and its children are the elements which are part of it [13]. The Abstract Semantic Graph (ASG) is a higher level representation where nodes represent source code entities and edges represent relationships between them. It is similar to the AST but, for example, instead of a node with the name of the referenced object, the ASG contains an edge from the first node to the referenced node [3]. A matrix may be used to provide a simplified representation of the relationships between classes, as will be described later.

For this work, we use the Eclipse JDT's `ASTParser` to create an Abstract Syntax Tree [13]. Each Java file is parsed and each element traversed by the `ASTVisitor`. To catch elements we are interested in, we must extend `ASTVisitor` and override the `visit()` methods for each type and store the node information in a data structure.

B. *Structural Matching*

For structural matching of anti patterns, we use the matrix representation developed by Dong *et al.* [10], with prime numbers encoding relationships between classes. The use of prime numbers allows multiple relations to be encoded and decoded unambiguously. With the mapping defined in Table I, we note that the UML class diagram shown in Figure 1 may be represented by the matrix given in Figure 2.

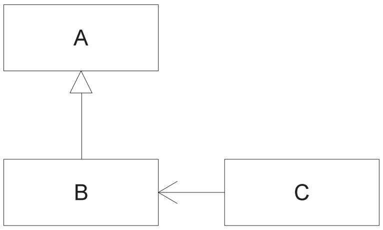


Fig. 1. Three classes and the relationships between them, shown using UML.

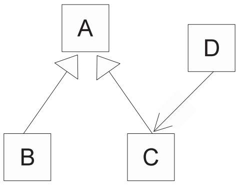
TABLE I
Prime number values for different relationships

Relationship	Prime Number Value
Association	2
Generalization	3
Aggregation	5

The process of actually searching for an instance of the pattern matrix within the (larger) system matrix is a relative of the subgraph isomorphism matching problem in graph

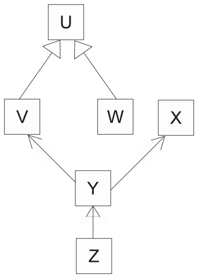
$$\begin{array}{cc} & \begin{array}{ccc} A & B & C \end{array} \\ \begin{array}{c} A \\ B \\ C \end{array} & \begin{pmatrix} 0 & 0 & 0 \\ 3 & 0 & 0 \\ 0 & 2 & 0 \end{pmatrix} \end{array}$$

Fig. 2. The matrix for Figure 1 using prime numbers.



$$\begin{array}{cc} & \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{array}{c} A \\ B \\ C \\ D \end{array} & \begin{pmatrix} - & - & - & - \\ 3 & - & - & - \\ 3 & - & - & - \\ - & - & 2 & - \end{pmatrix} \end{array}$$

Fig. 3. An example pattern



$$\begin{array}{cc} & \begin{array}{cccccc} U & V & W & X & Y & Z \end{array} \\ \begin{array}{c} U \\ V \\ W \\ X \\ Y \\ Z \end{array} & \begin{pmatrix} - & - & - & - & - & - \\ 3 & - & - & - & - & - \\ 3 & - & - & - & - & - \\ - & - & - & - & - & - \\ - & 2 & - & 2 & - & - \\ - & - & - & - & 2 & - \end{pmatrix} \end{array}$$

Fig. 4. An example system to which the pattern in Figure 3 is matched.

theory. We explore both a brute force algorithm and a second algorithm which is similar to a breadth-first search. Before using either of these techniques, we reduce the size of the search space by including only classes “close” to the modified class.

1) “K-Steps” shrinking: A developer working with a large system comprised of many classes will often only work with a few classes at a time. Only those classes being modified, along with those “close” to them, should be examined for pattern matches. When the relationships between all classes in the system are found, we can think of the entire system as a graph where each class is a node and each relationship an edge. We define the distance between two classes as the distance in the graph, ignoring the direction of the edges. In order to limit matching, we start from the edited class and find all classes within k steps, where k is the max distance between any two classes in the pattern we want to match.

2) Permute-and-Match: We first consider a brute force method which checks for every permutation of the nodes in the graph and whether or not it matches the anti pattern matrix. Since the system is usually larger than the pattern, we must first find all the subsets of the graph before permuting their order to perform matching. Also recall that multiple types of relationships can be stored in this single graph by using different values for different types of relationships. So when matching the relationship values, the system value must be divisible by the pattern value in order to be considered a match. For example, if the pattern requires a generalization relationship (given the value 3) and the system relationship we are matching has a value 6, this is still considered a match because $6 = 2 * 3$ and therefore it has both a generalization and an association relationship. Because of this, a sub matrix which has extra relationships is still considered a match.

Figure 3 shows a graph and representative matrix for a pattern, and likewise Figure 4 represents the system we are matching to. (A dash for cell i, j indicates that there is no relationship from i to j. Internally this is stored as the value one). This process will find that the relationships between the set of vertices {U,W,V,Y}, in that order, will create a match to the matrix in Figure 3.

Finding every permutation and matching it to the pattern is very inefficient. It requires the value of $P(m, n)$ where m is the number of classes in the system and n is the number of nodes in the pattern. If our basic operation is the array comparison during matching, then in the worst case this algorithm will require

$$\frac{m!}{(m-n)!} * n^2$$

operations, which approaches order $O(m! * n^2)$ for $m \approx n$. With the help of the K-Steps algorithm (Section III-B1), the system matrices are relatively small, but even for a system matrix reduced to 7 nodes and a pattern of size 4, the worst case will require over 13,000 comparisons.

3) Tree Matching: Consider a situation where the pattern has a class with two classes inheriting from it, while the system has a matching class but with three classes inheriting from it. The permute-and-match process will find a match for each permutation of two of those three classes. Firstly, this results in multiple matches which are essentially the same match. Secondly, we may actually want to know about all three of these inheriting classes, even if the pattern only requires two. In order to recognize which classes are part of a set defined in the pattern and pass them on for behavioral matching, we developed an algorithm to perform matching in a way similar to a breadth-first search.

We choose a node in the pattern tree and try to match it to every one of the nodes in the system tree as described in Algorithm 1. For example, in Figures 3 and 4, we would try to find a match for pattern node A by first comparing it to the system node U. The algorithm looks both at the relationships from A and to A, checking if there are a sufficient number of like relationships to and from U. If it finds a matching relationship, for example that V inherits from U, it recursively checks V for the required relationships before determining that U actually matches the pattern.

C. Behavioral Matching

Structural information can define a large part of any pattern. However, in order to match an element of a pattern, it is often

Algorithm 1: Algorithm to recursively check if there is a match between a pattern element and a particular class.

```
Data: patternIdxCheck, uncheckedPatternIndices,
      systemIdxMatch,
      unmatchedSystemIndices,currentMatch
Result: A boolean representing whether there was a
        match found. currentMatch will be updated
for pattIndex in uncheckedPatternIndices do
    if there is a relationship between patternIdxCheck
    and pattIndex then
        numMatchesFound = 0
        for sysIndex in unmatchedSystemIndices do
            if there is a matching relationship between
            systemIdxMatch and sysIndex then
                aMatch = a node representing this match
                if match(pattIndex, uncheckedPat-
                ternIndices.remove(pattIndex), sysIndex,
                unmatchedSystemIndices.remove(sysIndex)
                then
                    currentMatch.addChild(aMatch)
                    numMatchesFound++
                end
            end
        end
        if numMatchesFound ≥ required number of
        matches then
            continue
        end
        else
            return false
        end
    end
    return true
end
```

necessary to define not only the categorized relationships to other elements but also how they relate. Therefore, once a structural match is found, behavioral matching is performed on each of those elements.

Each behavioral requirement will be stored as a tree with a root node to specify the pattern element that should contain it, element nodes which match to certain types (i.e. switch statements, object instantiations, etc.), and references to other pattern elements. For example, if the pattern element named Client should contain a switch statement with references to all of the Product elements, the behavioral definition stored in the file would be

```
ROOT:Client {
   ELEM:SwitchStatement {
      REF:Product
   }
}
```

Our algorithm steps through each root node defined in the pattern and pulls out the class that is paired with it during structural matching. In our case, using an `AST` representation, it then searches for the first element, a `SwitchStatement`, using an `ASTVisitor` object which stores only that type. Each matched `ASTNode` is then revisited, using a new `ASTVisitor`, and the process continues until the entire tree has been matched. Before beginning this process, each behavior node is passed a list of all pairings so that the references can be set to actual class names in the current system. Therefore, when a `REF` node is matched, it searches for references to the appropriate class names. This structure allows for considerable flexibility in defining elements to search for. By defining requirements in a tree-like fashion, references can be searched for within constructors and method calls, or more generally within if and switch statements.

Finally, we must consider that behavioral information need not always be defined by specific syntax. We want to allow a pattern to be defined in terms of different options (such as a switch statement or a series of if-then-else statements). Therefore, after defining several different behavioral structures, the pattern also contains a statement declaring what is required and what is optional. For example, a pattern with three different behavioral structures could state that element three is required, along with either one or two:

```
3 AND (1 OR 2)
```

This logic sentence is converted into postfix using Dijkstra's Shunting-yard algorithm [9] and then evaluated with each index replaced by the boolean result of that behavioral match.

D. Pattern Definition Format

Each of the patterns created is defined in its own file and stored with the other patterns. As shown in Figure 5, a series of parameters are set, followed by labels for each element. If a particular class can be matched to multiple elements (e.g., we are looking for all the inheriting children of a particular class), the minimum number of required elements is listed with the class. The next part of the file specifies the matrix representing the relationships between elements. Following this is a list of behavioral descriptions, made into tree structures and indexed in order starting at zero, and a logic phrase to specify which are required (the & symbol is used for AND and | for OR). Finally, each file contains a paragraph to display to the user about the recommended Design Pattern. Note that in Figure 5, `ElementLabel1` should match to one element, `ElementLabel2` to at least one element, and `ElementLabel3` to two or more. Also, when `ElementLabel3` is listed as a reference in the behavioral requirements, this means that all matches to `ElementLabel3` should be referenced.

E. Dynamic Recommendations

Happel *et al.* [12] discuss the issues of what and when to recommend in a recommendation system for software engineering. For a programmer working on a large system, it would be useless to recommend the use of a design pattern in

```
Name of Pattern
NUM_CLASSES = <number classes in pattern>
NUM_BEHAVIOR = <number behavioral elements>
MAX_STEPS = <max number steps
             between any two classes>
ElementLabel1
ElementLabel2(1)
ElementLabel3(2)
. . .
ElementLabelN
1  1  2 . . . 1
3  1  1 . . . 1
1  1  1 . . . 1
   .    .    .
   .     .   .
   .     .   .
1  1  1 . . . 1
ROOT:ElementLabel1{
   ELEM:<ASTNode type>{
      REF:ElementLabel3
   }
}
ROOT:ElementLabel1 {
   . . .
}
ROOT:ElementLabel2 {
   . . .
}
. . .
(0 | 1)&2
Paragraph explaining design pattern that
will be recommended to the user.
```

Fig. 5. The configuration for files defining anti design patterns.

a package they are not working on and are not responsible for. Therefore, in addition to being able to detect these patterns, it is important to consider where to search for them and how often to make recommendations. A programmer who is constantly interrupted with suggestions is likely to begin ignoring them or turn off the tool. Therefore, once a recommendation has been ignored, a dynamic recommendation system must remember and not revisit it.

Every time a user makes a modification to a file, our tool checks to see if there has been a change in relationships between that class and the others in the system. Because our tool only looks for matches when there is a change, the user will only receive recommendations about the part of code which he is currently editing. Once a match is discovered, the tool presents it to the user automatically and without requiring the user to make any requests, as suggested by Murphy-Hill *et al.* [17]. The plugin has its own window, where current recommendations will be displayed. When the user sees that there is a recommendation, he can click on it to obtain more information, which is in the form of a popup describing the recommended pattern and pointing out which files are involved.

F. Sample Definition of Singleton Pattern

In this section we provide the definition of the Singleton creational pattern. We explain the pattern and also the indicators that a programmer should use it.

The Singleton design pattern is used when only one instance of an object is to be created during a particular execution. The Singleton object keeps track of one instance of itself. Instead of instantiating it with the *new* keyword, a `getInstance()` method is called each time it is needed by another object.

A programmer who is not following this design pattern may attempt alternative implementations to accomplish the same goal, which are the rules we are looking for in order to recommend the use of the Singleton design pattern. To discuss these “bad” solutions, we will refer to the “single” class (the class which should have only one instantiation) and the using class or classes.

Many times a Singleton pattern is needed when the single class may or may not be instantiated, depending on whether certain code fragments are reached. In the anti pattern, in order to guarantee that it is only instantiated once, the programmer may set up a conditional check before instantiating the single class with the `new` operator.

```
Singleton
NUMCLASSES = 2
NUMBEHAVIOR = 1
MAXSTEPS = 2
User(2)
1,1
5,1
ROOT:User {
   ELEM:IfStatement {
      ELEM:ClassInstanceCreation {
         REF:Single
      }
   }
}
```

Fig. 6. Whether or not the object exists is checked before instantiating the single class.

The structural tests are very simple; if two or more objects have an aggregation with another object (the singleton), it fits the structural requirements. The behavioral check involves looking for an existence of a conditional containing an instantiation which refers to the single class. This is specific but flexible enough to allow for either checking if the object equals `null` or some boolean value set up by the programmer. The structural and behavioral definition is shown in Figure 6.

IV. Conclusions and Future Work

We have focused on developing a framework for defining and detecting anti design patterns and making dynamic recommendations to the programmers. We have developed a format for representing both the structural and behavioral requirements of these patterns.

In order to expand the patterns that our system can recommend, future work requires the determination and inclusion of more anti patterns to provide recommendations for more design patterns. One possibility for doing so would be to obtain the input of experts on design patterns who have the experience to know what the common design mistakes are that developers make. Of particular utility would be a corpus of examples to test and perfect not only our system but those of other researchers in this field. We hope that future research in this area will encourage the development of such examples.

As we are working from the idea of proof of concept, our format is flexible but somewhat limited in the types of behaviors and even structures that it can check. Due to the tree nature of our matching algorithm, we are not able to deal with cyclical structures. Subgraph isomorphism matching is a similar problem and research in this area may provide more insight, but any solution to the basic graph problem would need to be modified for our situation, due to the existence of different edge types in our model.

Since behavioral matching can only find a matching node and then drill down into it, there are certainly more complicated structures which cannot be defined. It would also be useful to understand the uncertainty of a particular match, and tell the user this information. Better behavioral matching and uncertainty would require looking for multiple matching to certain behavioral constructs and a method of probabilistic reasoning (such as Bayes reasoning).

Finally, an advanced version of this plugin could help programmers refactor their code into the recommended design pattern. However, this would require a much more extensive programmatic understanding of each design pattern and its anti pattern.

References

[1] K. Brown, "Design Reverse-Engineering and Automated Design Pattern Detection in Smalltalk," Technical Report TR-96-07, Dept. of Computer Science, North Carolina State Univ., 1996.

[2] W. J. Brown, R. C. Malveau, W. H. Brown, H. W. McCormick III, and T. J. Mowbray, emphAnti Patterns: Refactoring Software, Architectures, and Projects in Crisis. John Wiley and Sons, 1st edition, March 1998.

[3] P.T. Devanbu, D. S. Rosenblum, A.L. Wolf. "Generating Testing and Analysis Tools with Aria", *ACM Transactions on Software Engineering and Methodology*, vol 5 no. 1, January 1996.

[4] F. Khomh, S. Vaucher, Y.-G. Guéhéneuc, and H. Sahraoui, "A Bayesian Approach for the Detection of Code and Design Smells," in *2009 Ninth International Conference on Quality Software*, qsic, pp.305-314, 2009.

[5] M. Fowler, *Refactoring – Improving the Design of Existing Code*, 1st ed. Addison-Wesley, June 1999.

[6] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns, Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[7] Y.-G. Guéhéneuc and R. Mustapha. "A simple recommender system for design patterns", in *Proceedings of the 1st EuroPLoP Focus Group on Pattern Repositories*, July 2007.

[8] R. Holmes, R.J. Walker, and G.C. Murphy, "Approximate Structural Context Matching: An Approach for Recommending Relevant Examples," in *IEEE Trans. Software Eng.*, vol. 32, no. 1, 2006, pp. 952970.

[9] E. W. Dijkstra, "Making a Translator for ALGOL 60," in *APIC-Bulletin*, no. 7, 1961.

[10] J. Dong, D. S. Lad, and Y. Zhao, "DP-Miner: Design Pattern Discovery Using Matrix," in *14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)*. (Tuscon, Arizona, March 26-29, 2007, pp.371-380).

[11] J. Dong, Y. Zhao, and T. Peng, "A review of design pattern mining techniques," *International Journal of Software Engineering and Knowledge Engineering (IJSEKE)*, vol. 19, no. 6, pp. 823-855, 2009.

[12] H.-J. Happel and W. Maalej, "Potentials and challenges of recommendation systems for software development", in *Proceedings of the 2008 international workshop on Recommendation systems for software engineering*, (Atlanta, Georgai, November 09-15, 2008, pp.11-15).

[13] T. Kuhn and O. Thomann, "Abstract Syntax Tree," *Eclipse Corner Articles*, 20 Nov 2006. [Online]. Available WWW:http://www.eclipse.org/articles/article.php?file= Article-JavaCodeManipulation_AST/index.html.

[14] R. Marinescu, "Detection Strategies: Metrics-Based Rules for Detecting Design Flaws," in *20th IEEE International Conference on Software Maintenance (ICSM'04)*, (September 11-14, 2004, pp.350-359).

[15] R. Marinescu, "Measurement and Quality in Object-Oriented Design," *21st IEEE International Conference on Software Maintenance (ICSM'05)*, (Budapest, Hungary, September 25-30, 2005, pp.701-704).

[16] M. J. Munro, "Product metrics for automatic identification of 'bad smell' design problems in java source-code," in *Proceedings of the 11th International Software Metrics Symposium*, (September 19-22, 2005, pp.15).

[17] E. Murphy-Hill and A. P. Black, "Seven habits of a highly effective smell detector", in*Proceedings of the 2008 international workshop on Recommendation systems for software engineering*, (Atlanta, Georgia, November 9-15, pp.36-40).

[18] M. Robillard, R. Walker, T. Zimmermann, "Recommendation Systems for Software Engineering," *IEEE Software*, vol. 27, no. 4, pp. 80-86, July/Aug. 2010.

[19] 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," *Transactions on Software Engineering (TSE)*, vol.36, no.1, pp.20-36, 2009.

[20] R. Oliveto, F. Khomh, G. Antoniol, and Y.-G. Guéhéneuc, "Numerical signatures of antipatterns: An approach based on b-splines". In *Proceedings of the 14th Conference on Software Maintenance and Reengineering*. IEEE Computer Society Press, March 2010.

[21] M. Salehie, S. Li, L. Tahvildari, "A Metric-Based Heuristic Framework to Detect Object-Oriented Design Flaws," in *14th IEEE International Conference on Program Comprehension (ICPC'06)*, (Athens, Greece, June 14-15, 2006, pp.159-168).

[22] S. Thummalapenta and T. Xie, "PARSEWeb: A Programming Assistant for Reusing Open Source Code on the Web," in *Proc. IEEE/ACM International Conference on Automated Software Eng. (ASE 07)*, (Atlanta, GA, November 5-9, 2007, pp. 204213).

[23] T. Zimmermann, P. Weißgerber, and S. Diehl "Mining Version Histories to Guide Software Changes," in *IEEE Trans. Software Eng.*, vol. 31, no. 6, 2005, pp. 429445.

Towards a Novel Semantic Approach for Process Patterns' Capitalization and Reuse

Nahla JLAIEL
RIADI Research Laboratory
National School of Computer Science
La Manouba, Tunisia
Nahla.Jlaiel@riadi.rnu.tn

Mohamed BEN AHMED
RIADI Research Laboratory
National School of Computer Science
La Manouba, Tunisia
Mohamed.Benahmed@riadi.rnu.tn

Abstract—**Process patterns are widely used by the software engineering's community as an excellent mechanism for communicating software development knowledge (experiences and best practices) that has proven to be effective in practice. As a consequence, several description languages and formats have been proposed in the literature dealing with process patterns, in which patterns are described with different terminologies and reused in informal manner. These disparities make capitalization and/or reuse of process patterns, difficult to be achieved. This is why, in this paper, we propose a new semantic approach for process patterns capitalization and reuse, named SCATTER, which aims to disseminate software process best practices by making process patterns described in a unified and formal form. The proposed approach is based on two main processes namely, process patterns warehousing and process patterns mining.**

Keywords-software process patterns; patterns capitalization; patterns reuse; patterns unification; patterns warehousing; patterns mining; ontologies; information extraction; text mining; ontologies' population

I. INTRODUCTION

Nowadays, patterns are increasingly being recognized by software development communities, as an effective method to reuse knowledge and best practices gained during software development processes [1] [2].

In fact, software patterns now exist f or a wide range of topics including process patterns, requirement, analysis, design, implementation or code patterns, test patterns and even maintenance patterns.

Concerning process patterns, whose main role is to capitalize good specifications or implementations of a method to be followed to achieve a goal [3], they become commonly used by software development communities as an excellent medium to share software development knowledge that is often encapsulated in experiences and best practices [4] [5] [6].

Indeed, process patterns are growingly being adopted by different development processes such as A gile processes [7], Object-oriented Software Development processes [8], Component Based Software Development processes [9], Service-Oriented Development processes [10] as well as Aspect-oriented Development processes [11]. As co nsequence to the huge proliferation of the process patterns practice, these latter are being used in an informal manner, through traditional textbooks or better with modest hypertext systems providing weak semantic relationships. In addition to the huge number of process patterns that are available in books or Web-based resources [2], they significantly differ in format, coverage, scope, architecture and terminology used [12].

All of these observations conspire to create barriers to the efficient use of process patterns. In fact, patterns users are expected to investigate different patterns resources such as books, magazines, papers and Web collections to find the most appropriate patterns. This investigation really needs cognitive efforts, abilities and time to identify, understand, select, adapt and apply relevant ones.

For these reasons, we argue that efforts are needed to more formally capitalize patterns knowledge in order to help software development communities use, reuse and create process patterns during any given software development process. The overall goal of our research is to build up an intelligent framework supporting process patterns capitalization and reuse. Process patterns capitalization should enable patterns' integration, validation, evolution and/or creation through patterns' reuse. In other words we aim to cr eate a framework for process patterns' knowledge management

In this paper, we outline the proposed approach SCATTER, acronym for "*SemantiC Approach for sofTware process paTErns capitalization and Reuse*" which is a ne w knowledge management approach [13] adapted for software process patterns.

The remainder of this paper is organized as follows: section II deals with the context and motivation of our research work. Section III provides background information on the process patterns' literature review that we carried out and the resulting meta-model forming the building block of the proposed approach. Section IV gives details on how the proposed approach is performed to better disseminate knowledge and best practices within software development communities. Section V concludes the paper by giving a discussion of our contributions as well as an overview of our work in progress.

II. CONTEXT AND MOTIVATION

This section is devoted to a brief description of our research context in a first subsection and then to the illustration of our research motivation as a second subsection.

A. Pattern Reuse

In patterns reuse state of the art and practice, there are two complementary processes: a process for the reuse and a process by the reuse [14]. The software engineering process for re use concerns patterns engineering and aims to identify, specify and organize patterns that should be used during the process. The software engineering process by reuse is devoted to patterns engineering allowing the search, selection, adaptation and the integration of patterns for software engineering.

As conclusion to the literature review that we carried out on patterns reuse, we noted that a great deal of interest is addressed to product patterns and more precisely design patterns in research as well as in industry. In addition, we derived that most of the deployed efforts to improve pattern reuse are specific to design patterns and could not be adopted or adapted for other pattern types. Among these, we cite the efforts deployed by [15] to automatically detect design patterns' problem.

On the other hand, process patterns are increasingly being described within software development communities but in different forms. In fact, in our literature review, we identified eleven process patterns' description models that are being adopted separately and independently. In other words, every community creates and manages its own patterns' collection or system. This would in our view, cripple patterns' knowledge reuse especially when we find communities using different models to describe their patterns. As a consequence, patterns' users could only have access to patterns that are maintained by their own community.

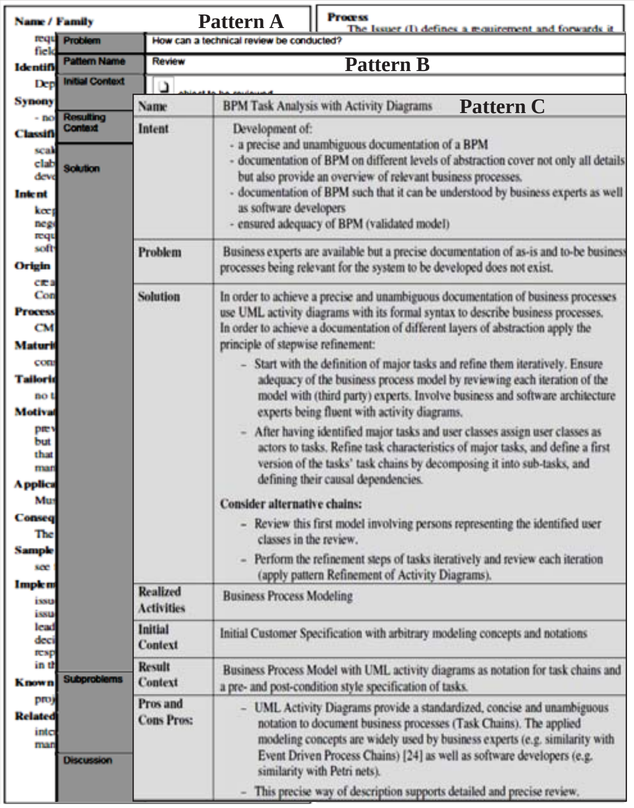
So, we argue that a holistic approach is required to better reuse and capitalize process patterns' knowledge. This latter should support the two complementary processes mentioned above, namely: patterns for reuse and patterns by reuse

B. Motivating Example

To motivate our position, we try to state the problem of model diversity with regard to process patterns description by showing some patterns' samples. Figure 1 illustrates this by presenting three different process patterns. Pattern A, proposed by Störrle [16], deals with requirements administration using its own vocabulary to describe patterns attributes such as *Intent* to express the pattern's problem. Pattern B, described by Dittman [17], handles the technical review process using the term *subproblem* as a pattern's problem. Pattern C which is created by Gnatz [18], deals with task analysis and employs another different vocabulary such as *Pros and Cons Pros* to express pattern's discussion.

As far as we can see, these patterns are not only informally and differently defined but also described in an unstructured manner. We believe that all of these observations conspire to create barriers to patterns share and reuse. This matter of fact leads us to think to unify and to formalize pattern descriptions in order to make them machine- processable so as to more efficiently share and capitalize them.

Section III presents the unification meta-model that we created to express process patterns knowledge after the mapping efforts that we made on the different patterns description models.


Pattern A
Name / Family
Identifi
Synony
Classifi
Intent
Origin
Process
Maturi
Tailori
Motivat
Applica
Conseq
Sample
Implem
Known
Related
Process
The Issuer (I) defines a requirement and forwards it
Pattern B
Pattern C

Problem	How can a technical review be conducted?
Pattern Name	Review
Initial Context	object to be reviewed
Resulting Context	
Solution	
Subproblems	
Discussion	

Name	BPM Task Analysis with Activity Diagrams
Intent	Development of: - a precise and unambiguous documentation of a BPM - documentation of BPM on different levels of abstraction cover not only all details but also provide an overview of relevant business processes. - documentation of BPM such that it can be understood by business experts as well as software developers - ensured adequacy of BPM (validated model)
Problem	Business experts are available but a precise documentation of as-is and to-be business processes being relevant for the system to be developed does not exist.
Solution	In order to achieve a precise and unambiguous documentation of business processes use UML activity diagrams with its formal syntax to describe business processes. In order to achieve a documentation of different layers of abstraction apply the principle of stepwise refinement: – Start with the definition of major tasks and refine them iteratively. Ensure adequacy of the business process model by reviewing each iteration of the model with (third party) experts. Involve business and software architecture experts being fluent with activity diagrams. – After having identified major tasks and user classes assign user classes as actors to tasks. Refine task characteristics of major tasks, and define a first version of the tasks' task chains by decomposing it into sub-tasks, and defining their causal dependencies. **Consider alternative chains:** – Review this first model involving persons representing the identified user classes in the review. – Perform the refinement steps of tasks iteratively and review each iteration (apply pattern Refinement of Activity Diagrams).
Realized Activities	Business Process Modeling
Initial Context	Initial Customer Specification with arbitrary modeling concepts and notations
Result Context	Business Process Model with UML activity diagrams as notation for task chains and a pre- and post-condition style specification of tasks.
Pros and Cons Pros:	– UML Activity Diagrams provide a standardized, concise and unambiguous notation to document business processes (Task Chains). The applied modeling concepts are widely used by business experts (e.g. similarity with Event Driven Process Chains) [24] as well as software developers (e.g. similarity with Petri nets). – This precise way of description supports detailed and precise review.

Figure 1. Heterogeneity of process patterns' descriptions.

III. Background

This section is intended to provide background information for the proposed approach. The first subsection is dev oted to sum up results of the study that we carried out in order to assess process patterns representation and reuse within software development communities. The next subsection will exhibit the general structure of the proposed process pattern meta-model as the building block of the proposed approach.

A. Process Patterns Literature: Review's Results

Different works have been carried out in the literature of patterns dealing with process patterns' description and formalization. These are classified into description models such as Ambler [8], RHODES[19], Gnatz [18], P-Sigma [3], Störrle [16] and other as languages, such as PROMENADE [20], PPDL [5], PROPEL [21], PLMLx [22], UML-PP [6] and PPL [23].

Based on eleven evaluation criteria that we fixed in a previous work [24], we assessed the aforementioned works and revealed several lacks, detailed in [12] and [24], creating barriers to patterns' knowledge capitalization and reuse. Among these, we notice the lacks of architectural as well as terminological consent in patterns descriptions.

- The lack of architectural consent means that different process pattern descriptions have been proposed using disparate architectures. In fact, when comparing the eleven selected works from the literature, we identified eleven different pattern description facets, namely:

identification, classification, problem, context, solution, role, artifact, relationship, guidance, management and evaluation [12]. In addition, these are differently covered by process patterns descriptions as it is shown in Fig. 2 most of them pay attention to the four main facets: context, solution, problem and relationships (15%, 14% and 13%).

- The lack of terminological consent refers to the problems of polysemy and synonymy addressed in labels used to describe patterns. Indeed, we find terms such as *Consequences* used to express a *Resulting Context* in PPL as well as a *Guideline* in Gnatz. Moreover, others different terms are being used to address the same concept such as *Intention* in RHODES to describe a pattern *Problem,* instead, the term *Intent* is used in Störrle.

B. Unified Conceptualization of Process Patterns

To overcome these lacks, a first step was to create a unified conceptualization of process patterns. Mappings efforts [12] were necessary to achieve this goal leading to a process patterns' meta-model unifying patterns knowledge representations.

Fig. 3 provides an abstract view of the meta-model structure in which we consider a process pattern information description from six facets:

- The identification facet encapsulates a set of properties identifying a pa ttern such as pattern name, author(s), keywords, pattern's classification (type, category, abstraction level, and aspect) as well as pattern origin (project and participants) and pattern artifacts (used and/or produced).
- The core information is the main pattern facet embodying details about the well-known triplet: problem, context and solution.
- The relationships facet expresses how a pattern could interact with other patterns (e.g. similar patterns, refinement patterns, subsequent patterns, and anti-patterns)
- The guidance facet refers to the support level provided by a pattern to be comprehended and used (e.g. known uses, example, literature, illustration, etc.)
- The evaluation facet provides feedbacks on pattern application (e.g. discussion, confidence, maturity, etc.)
- The management facet provides general information about a given pattern (e.g. version, creation-date)

In order to validate the proposed meta-model, we adopted an ontology-based approach providing common and shared architecture and terminology for better patterns' capitalization and reuse. The proposed ontology [12], named MetaProPOS and acronym for Meta Process Patterns' Ontology for Software Development, aims to unify all th e proposed process patterns descriptions, providing thus semantic interconnection and composition of relevant and unified bodies of patterns knowledge.

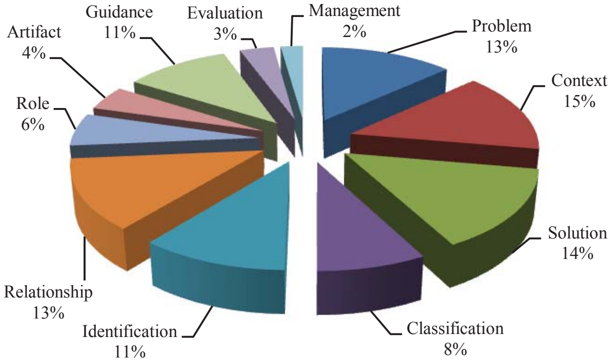


Figure 2. Architectural dissent in patterns descriptions [12].

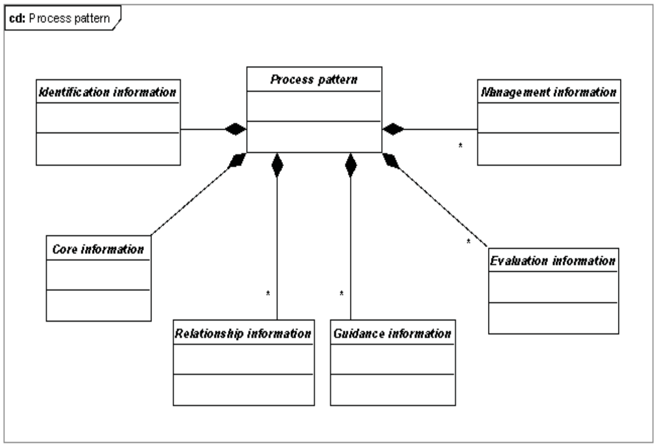


Figure 3. Abstract view of a unified process pattern.

Because of space limitations, we could not give details of all the modeled facets. So we choose to illustrate this work by the relationships facet which is qu iet important for the pattern reuse process. Fig. 4 shows the relationship hierarchy considered by the proposed meta-model. An alternative pattern is a pattern that could be applied as an alternative solution which can be different from a similar pattern. A refinement pattern should give details to an a bstract one. A consequent pattern is a pattern that is implied or that could be applied after a given pattern. An anti-pattern is a pattern that could not be applied with another given pattern.

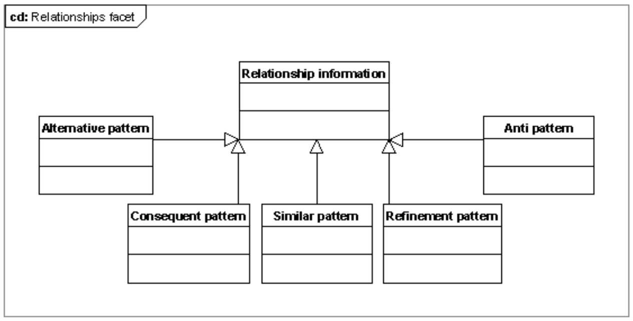


Figure 4. The relationship's facet.

IV. SCATTER

In order to enhance patterns capitalization and reuse, a first step in this direction was to build up a unified conceptualization of process patterns via the proposed ontology MetaProPOS, as it is stated before. The next step is to demonstrate how it can be performed to fulfill our goal through the proposed approach SCATTER which, as the name implies, aims to help improve process patterns' knowledge dissemination by means of a formal and semantic technique of patterns' capitalization and reuse. As illustrated by Fig. 5, SCATTER comprises two major processes:

A. *Terminological, Semantic and Architectural Unification Process*

Given different collections of process patterns forming a patterns' corpus, this main process consists of a triple unification effort.

1) *Terminological unification:* It aims to map between terms used as labels for a given pattern and the corresponding meta labels in the proposed meta model. This is ensured by means of MetaProPOS on the one hand, and a text mining tool on other. The purpose of this phase is to recognize the terminology employed by a given pattern through key terms extraction. The unification's result is an annotated pattern whose format is XML. In order to reach this target, we adopted a text mining approach. Indeed, there are already tools that are well recognized for their mastery in Natural Language Processing (NLP) namely: Open NLP [25], UIMA (Unstructured Information Management Architecture [26] and GATE (General Architecture for Text Engineering) [27]. Thus, we do not need to reinvent the wheel by rebuilding one from the scratch. This is why we choose to reuse, among these latter, GATE since it is open source and very well documented as well as used in research and industry. In this regard it should be noted that the terminological unification phase is performed using the information extraction open source component of GATE which is ANNIE acronym for A Nearly-New Information Extraction system [28].

We should notice that in addition to GATE configuration, we have made a considerable extension to ANNIE's system by adding Gazetteer lists such as "problem.lst", "context.lst", "relationship.lst and so on in order to help ANNIE's system recognize key terms and concepts used in patterns' descriptions. However, the use of these lists is necessary but insufficient to detect patterns segments, for this reason we added Jape rules such as "*identification.jape*" to capture the pattern's identification facet in a pattern description, "*guidance.jape*" to identify the pattern's guidelines, etc.

As it is illustrated by Fig. 6, we used in a first step, the Sentence Splitter and the Tokeniser to perform a morphological analysis of unstructured patterns allowing the extraction of sentences and basic entities. Then in a next step, the POS Tagger is performed to associate grammatical category to tokens allowing thus recognition of various entities. The last step consists in patterns tagging by extracting pattern concepts.

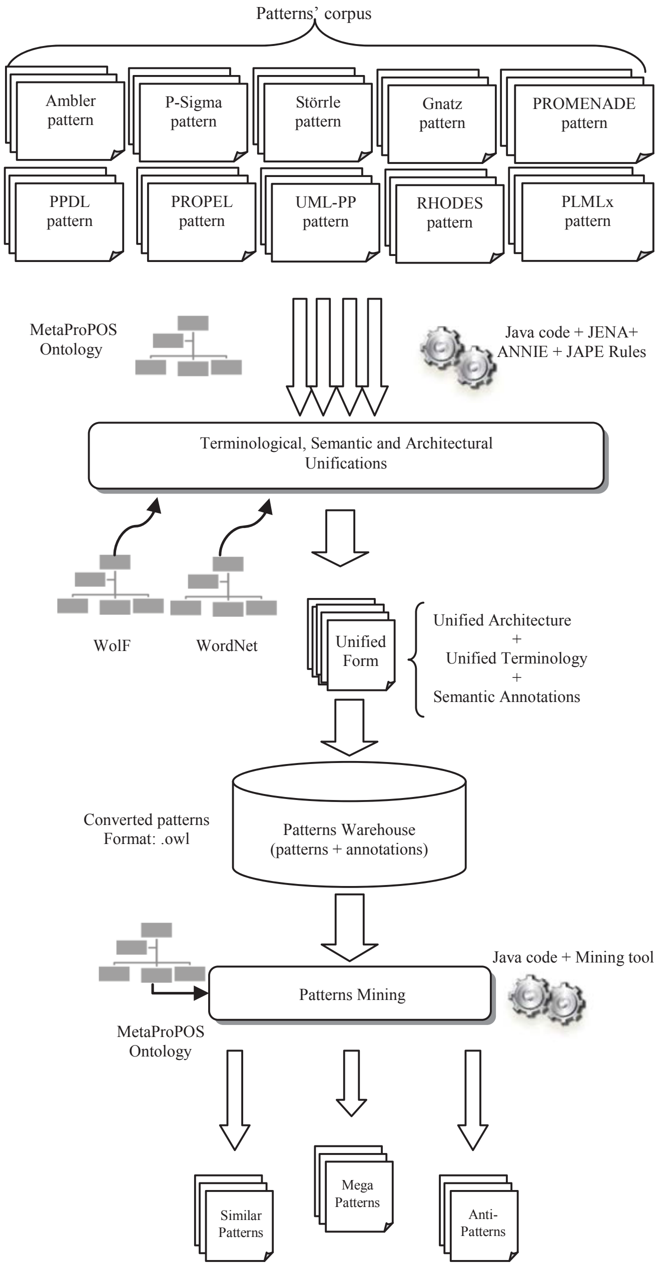


Figure 5. Functional architecture of SCATTER.

This step has recourse to previously generated results and uses the NE Transducer to extract named pattern's entities (problem, context, solution, relationship, etc.) through new JAPE rules and Gazetteer lists. The components implied to achieve these steps are described as follows:

a) Tokeniser: This component identifies various symbols in text documents (punctuation, numbers, symbols and different types). It applies basic rules to input text to identify textual objects.

b) Gazetteer: This component creates annotation to offer information about entities (persons, organizations, etc.) using lookup lists.

c) POS tagger: This component produces tags to words or symbols.

d) Sentence splitter: This component identifies and annotates the beginning and the end of each sentence.

e) NE transducer: This component applies JAPE (Java Annotations Pattern Engine) rules [29] to input text in order to generate new annotations.

f) Semantic tagger: This component contains rules which act on annotations assigned earlier, in order to produce outputs of annotated entities.

2) *Semantic unification:* This unification level concerns the content of patterns' labels. It consists in a text mining process, extracting terms and / or concepts that are most representative for the different patterns fields' content. These latter (terms, concepts) are weighted according to their occurrence number in the content and then, sorted according to their weight representing thus, a semantic annotation for the concerned features. The patterns' fields might also be enriched by synonyms extracted from the WordNet [30] ontology for the English as well as the WolF [31] ontology for the French language, contributing thus, to the enrichment of these semantic annotations. The semantic unification phase is performed using the GATE framework on structured patterns produced by the first phase. It adds semantic annotations to patterns content to form what we have called semantic process patterns (c.f. Fig. 7).

3) *Architectural unification:* This phase aims to normalize the description of any given process pattern by converting it from XML to OWL (Ontology Web Language). In other words, this unification level should allow the autmatic population of the proposed OWL ontology, named MetaProPOS with individuals based on a given patterns corpus. In order to achieve this, we use the Jena framework [32] to implement a Java component for ontology's population based on the produced semantic process patterns of the two previous unification phases as well as some extraction rules established for this purpose (c.f. Fig. 8). For information purpose, Jena is a Java framework for building Semantic Web applications. It provides a collection of tools and Java libraries as well as an ontology's API for handling OWL ontologies in order to help develop semantic web applications [32].

B. Patterns Mining Process

Since the overall goal of our research work is to build up a semantic and intelligent framework enhancing software process patterns capitalization and reuse, we adopted a process patterns warehousing and mining technique. As mentioned earlier, the pattern warehousing process is ensured by the three unification levels detailed above. As regards the pattern mining process, it consists on a reasoning process performed on the unified patterns in the warehouse based on MetaProPOS and the key ontolgy's concept Relationship. Indeed, this main concept would allow a better search of related patterns.

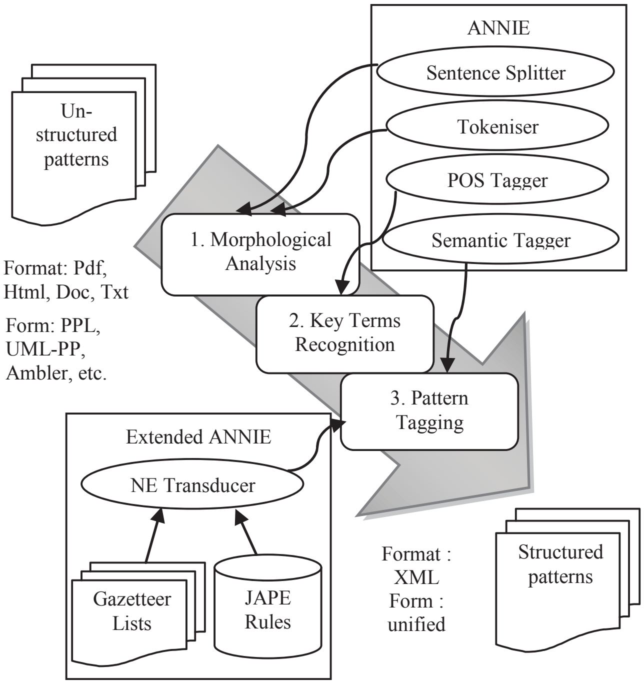


Figure 6. Terminological unification.

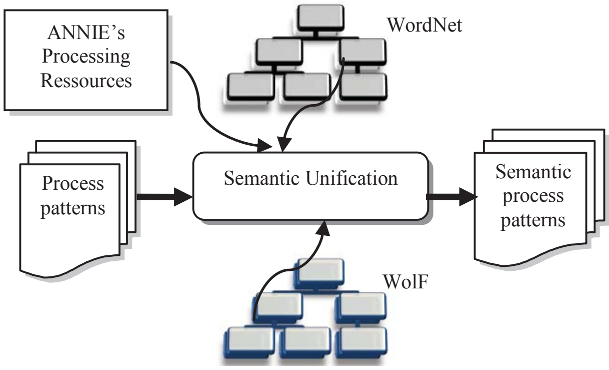


Figure 7. Semantic unification.

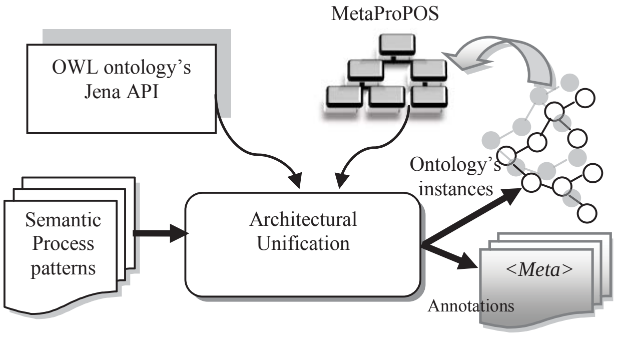


Figure 8. Architectural unification.

These latter would be clustered according to the relationship kind (c.f. Fig. 4). Consequently, a patterns' algebra is created allowing rigorous and automatic patterns processing providing better search of similar patterns through the Similar and Alternative relations as well as efficient guidance for patterns composition and aggregation through the Refinement, Consequent and Anti-pattern relations. Furthermore, process patterns could be clustered according to the software development phase or activity. So, a mega process pattern could be built as an aggregation and / or a composition of different process patterns for a given context or problem.

In order to reach this objective, we propose to combine the use of MetaProPOS with a mining tool such as Weka [33] which consists in a collection of machine learning algorithms for data mining tasks containing tools for data pre-processing, classification, regression, clustering, association rules, and visualization.

V. Conclusion and work in progress

The most valuable contribution of this paper is the general overview of targeted approach SCATTER which aims to provide a semantic framework for process patterns warehousing and mining given different patterns collections. This research work mediation implies terminological, semantic as well as architectural mediation efforts ensured by the three proposed unification levels. The process patterns' mining process would improve process pattern's capitalization and reuse quality.

However, SCATTER has not been completely finalized for implementation and is subject to refinement and validation, which remains our work in progress. Indeed, we should expand the proposed approach to annotate figures described through Petri nets to cover workflow process patterns [34] as well as UML activity diagrams. In this direction, we plan to combine the use of the UIMA plug-in and ANNIE to deal with image analysis and structuring in addition to text since GATE does not support this kind of information.

References

[1] Buschmann, F., Henney, K., and Schmidt, D.C., "Pattern-oriented Software Architecture: On Patterns and Pattern Languages", Wiley & Sons, 2007.

[2] Henninger, S., Corrêa, V., "Software pattern communities: current practices and challenges", 14th International Conference on Pattern Languages of Programming, pp. 1--19. ACM Proceedings, New York, 2007.

[3] Conte, A., Fredj, M., Giraudin J.P., and Rieu, D., "P-Sigma: a formalism for A unified representation of patterns (in French), 19ème Congrès Informatique des Organisations et Systèmes d'Information et de Décision, pp. 67--86. Martigny, 2001.

[4] Hagen, M., "Support for the definition and usage of process patterns", 7th European Conference on Pattern Languages of Programs, Dortmund, 2002.

[5] Hagen, M., and Gruhn, V., "Process patterns - a means to describe processes in a flexible way", 5th International Workshop on Software Process Simulation and Modeling, ICSE Workshops, pp. 32--39. Scotland, 2004.

[6] Tran, H.N., Coulette, B., and Dong, B.T., "Modeling process patterns and their application", 2nd International Conference on Software Engineering Advances, pp. 15--20, IEEE Proceedings, Cap Esterel, 2007.

[7] Tasharofi, S., and Raman, R., "Process patterns for agile methodologies", Situational Method Engineering: Fundamentals and Experiences, Proceedings of the IFIP WG 8.1 Working Conference, pp. 222--237,Springer, Switzeland, 2007.

[8] Ambler, S.W.: Process Patterns: Building Large-Scale Systems Using Object Technology. Cambridge University Press/SIGS Books, Cambridge, 1998.

[9] Kouroshfar, E., Yaghoubi Shahir, H., and Ramsin, R., "Process patterns for component-based software development", CBSE 2009, LNCS 5582, pp. 54--68, 2009.

[10] Fahmideh, M., Sharifi, M., Jamshidi, P., Feridoon, S., and Haghighi, H., "Process patterns for service-oriented software development", proceedings of the 5th IEEE International Conference on Research Challenges in Information Science (RCIS'2011), pp. 1--9, 2011.

[11] Khaari, M., and Ramsin, R., "Process patterns for aspect-oriented software development", ECBS, pp. 241--250, England, 2010.

[12] Jlaiel, N., and Ben Ahmed, M., "MetaProPOS: a meta-process patterns ontology for software development communities", KES Proceedings, Part I. LNCS 6881, pp. 516--527 Springer, Germany, 2011.

[13] Jlaiel, N., and Ben Ahmed, M., "Ontology and agent based model for software development best practices' integration in a knowledge management system", OTM Workshops, OntoContent 2006, LNCS 4278, pp. 1028 -- 1037, Springer, France, 2006.

[14] Gzara, Yesilbas, L., Rieu, D., and Tollenaere, M., "Patterns approach to product information systems engineering", Requirement. Engineering, vol. 5(3), pp. 157--179, 2000.

[15] Bouassida, N., and Ben-Abdallah, H., "A new approach for pattern problem detection", CAiSE, LNCS, Tunisia, pp.150--164, 2010.

[16] Störrle, H.,"Describing process patterns with UML", 8th EWSPT, LNCS, vol. 2077, pp. 173--18, Springer, Witten, 2001.

[17] Dittmann, T., Gruhn, V., Hagen, M.: Improved Support for the Description and Usage of Process Patterns. In: 1st Workshop on Process Patterns, 17th ACM Conference on Object-Oriented Programming, Systems, Languages and Applications, pp. 37--48. Seattle, 2002.

[18] Gnatz, M., Marschall, M., Popp, G., Rausch, A., and Schwerin, W.: "Towards a tool support for a living software development process", 8th EWSPT 2001. LNCS, vol. 2077, pp. 182--202. Springer, Witten, 2001.

[19] Coulette, B., Crégut, X., Dong, T.B., Tran, D.T., "RHODES, a process component centered software engineering environment", 2nd International Conference on Enterprise Information Systems, pp. 253--260, Stafford, 2000.

[20] Ribó, J.M., and Franch X., "Supporting Process Reuse in PROMENADE", Research report, Politechnical University of Catalonia, 2002.

[21] Hagen, M., and Gruhn, V., "Towards flexible software processes by using process patterns", 3rd IASTED Conference on Software Engineering and Applications, pp. 436--441, Cambridge, 2004.

[22] http://www.cs.kent.ac.uk/people/staff/saf/patterns/diethelm/plmlx_doc

[23] Meng, X.X., Wang, Y.S., Shi, L., and Wang, F.J., "A process pattern language for agile methods", 14th Asia-Pacific Software Engineering Conference, pp. 374--381, Nagoya, 2007.

[24] Jlaiel, N., and Ben Ahmed, M.,"Reflections on how to improve software process patterns capitalization and reuse", IKE, pp 30--35, USA, 2010.

[25] http://opennlp.apache.org/

[26] http://uima.apache.org/

[27] http://gate.ac.uk/

[28] http://gate.ac.uk/sale/tao/splitch6.html#chap:annie

[29] http://gate.ac.uk/sale/tao/splitch8.html#chap:jape

[30] http://wordnet.princeton.edu/

[31] http://alpage.inria.fr/~sagot/wolf-en.html

[32] http://incubator.apache.org/jena/

[33] http://www.cs.waikato.ac.nz/ml/weka/

[34] Van der Aalst, W.M.P., Ter Hofstede, A.H.M., Kiepuszewski, B., and Barros, A.P., "Worflow patterns", Distributed and parallel Databases, 14(3), pp. 5--51, 2003.

DC2AP: A Dublin Core Application Profile to Analysis Patterns

Lucas Francisco da Matta Vegi, Jugurta Lisboa-Filho, Glauber Luis da Silva Costa,
Alcione de Paiva Oliveira and José Luís Braga
Departamento de Informática
Universidade Federal de Viçosa
Viçosa-MG, Brazil, 36570-000
lucasvegi@gmail.com, jugurta@ufv.br, glaubercosta@gmail.com, alcione@dpi.ufv.br, zeluisbraga@ufv.br

Abstract—**Analysis patterns are reusable computational artifacts, aimed at the analysis stage of the process of software development. Although the analysis patterns can facilitate the work of analysts and programmers adding value through reuse of proven useful and tested ideas, the access to them is still very poor because of the way they are usually described and made available. In order to reduce these deficiencies, supporting cataloging and encouraging the reuse of analysis patterns, it was proposed the Analysis Patterns Reuse Infrastructure (APRI). This infrastructure comprises a repository of analysis patterns documented through a specific metadata profile and accessed via web services. Based on the proposal of APRI, this article presents the specific metadata profile to the documentation of analysis patterns called Dublin Core Application Profile to Analysis Patterns (DC2AP).**

Keywords- Analysis Patterns; Reuse; Metadata Standards; Dublin Core.

I. INTRODUCTION

Correction of errors made during the encoding of a software is usually more costly than the correction performed during the stages of analysis and design. The costs for correction of errors increase on each stage and in advanced stages they can be up to 100 times higher than in the early stages [1].

Due to the needs of companies, programmers and analysts are constantly pressured to deliver encoded projects of software as soon as possible, and the analysis stage is often left in the background [2]. This common situation in companies ends up generating software errors identified too late, thus burdening the costs of the final product.

According to Fernandez and Yuan [2], analysis patterns can make the analysis stage faster and more accurate for developers, thus preventing that this important stage of development is ignored. The analysis patterns are reusable computational artifacts, aimed at the analysis stage of the process of software development. According to Fowler [3], the analysis patterns are ideas proved useful in a given practical context and that may be useful in other contexts.

Although the analysis patterns can facilitate the work of analysts and programmers adding value through reuse of proven useful and tested ideas, the access to them is still very poor [4]. So far there is no template to specify the analysis patterns that is widely accepted making each set of analysis patterns is specified according to the preferences of its authors. In addition to not having a pattern specification, the analysis patterns are normally provided in scientific books and papers which are restricted access means and do not allow the efficient retrieval of patterns performed, for example, through a search tool [4].

In order to minimize these problems of specification and retrieval of analysis patterns, it was proposed the Analysis Patterns Reuse Infrastructure (APRI) [5]. This infrastructure, which was inspired by the Spatial Data Infrastructure (SDI) [6], consists of a repository of analysis patterns, documented in a specific metadata profile and accessed via web services.

This article proposes a specific metadata profile to the documentation of analysis patterns compatible with the APRI [5]. The metadata profile is based on the Dublin Core metadata standard [7] and on the template proposed in [8] and [9] to specify analysis patterns.

The remainder of this paper is organized as follows: Section 2 describes related works to documentation of analysis patterns and to the Dublin Core metadata standard. Section 3 describes the proposed metadata profile. Section 4 presents an example of analysis pattern specified with the proposed metadata profile and Section 5 presents some conclusions and possible future works.

II. RELATED WORK

A. Documentation and Organization of Analysis Patterns

Documentation of analysis patterns is an important way for contextualizing the reuse scope of a pattern and for enabling the sharing of knowledge among designers. However, this documentation is performed in a heterogeneous manner among the authors, since there is no standardized way to specify analysis patterns [4]. There are many approaches to specify analysis patterns, ranging from non-formalized textual descriptions to formalized descriptions based on templates.

Some analysis patterns specified in an on-formalized textual manner can be found in [3] and [10]. This little formal way of describing a analysis pattern affects reuse, because it makes harder for designers to quickly understand the contextual scope of patterns, and, mostly, it limits the retrieval of analysis patterns through computerized search engines. Thus important detailed information for designers may not be described or even retrieved, thus limiting the spread of these patterns and thereby their potential for reuse.

Analysis patterns have also been described through the use of templates, which are structures with predetermined topics similar to those used to describe design patterns [11]. Usually a template is composed of essential topics such as context, problem, motivation and solution [3], combined with other specific topics defined by their authors.

Some analysis patterns documented through templates can be found in [12] and [13]. Meszaros and Doble [14] present in their work a template composed of topics: name, problem, context, motivation, solution, participants and related patterns. This template was used in [12]. Pantoquilho, Raminhos and Araujo [8] and Raminhos et al. [9] present in their work a proposal of detailed template specifically for documentation of analysis patterns. This template combines common topics used previously by several authors, with new topics aimed at describing the analysis patterns more broadly.

Besides adequate documentation, another important factor to increase the potential for reuse of analysis patterns is the way they are organized and therefore available because before a pattern is applied to a project, the designer needs to know of its existence and then select it [4]. Usually analysis patterns are organized together forming collections of patterns, and in one collection, usually the patterns are documented homogeneously by the same author, although this is not a rule.

The collection of analysis patterns can have different formats, such as books, articles and websites, and they can still be classified as pattern languages and pattern catalogues [4]. The pattern languages are basically collections of analysis patterns aimed at solving a specific problem. In a pattern language, the patterns are related to each other and must follow application rules, for example, the order in which they must be applied to solve the problem in question [4]. The pattern catalogues are collections of analysis patterns not necessarily related, but organized based on criteria in common and searchable. Fowler's book [3] is an example of an analysis pattern catalog, because the organization of the patterns described in that book was obtained from groups of patterns with application domains in common and they may be found by potential users through a table of contents [4].

B. Dublin Core Metadata Initiative

Metadata are data about data, i.e., information that makes it possible to know the resources of the data. They can be used to standardize data representation based on the description of their authors, quality levels, application domains and other elements, thereby encouraging the appropriate reuse of data [6].

The metadata standards are metadata structures used to describe the data. According to the domain of the data to be described, these description structures may have variations compared to other domains due to peculiarities of each of them. Thus, several metadata standards have already been proposed in order to meet the specific needs of some domains. Examples of metadata standards for specific domains can be found in [15] and [16].

The Dublin Core metadata standard [7] appeared in 1995 from a workshop held in Dublin city in the U.S. state of Ohio. This event brought together professionals from several fields of knowledge in order to establish a generic metadata standard composed of a small set of recurring elements in all areas [17]. This metadata standard has two levels, Simple Dublin Core and Qualified Dublin Core.

The Simple Dublin Core consists of fifteen elements, as the Qualified Dublin Core has seven additional elements that allow more detailed descriptions of data. Beyond the twenty-two elements that compose the levels of the Dublin Core standard, it has many element refinements that can be used to specialize the semantics of an element in certain situations and thus facilitate the discovery of the data [18].

A major advantage of the Dublin Core standard is its versatility. Although it is very simple and does not provide enough resources to describe data of complex domains [17], it can be specialized from the creation of application profiles for specific domains [6].

Coyle and Baker [19] describe in their work the basic steps for creating a Dublin Core application profile. An example of a Dublin Core application profile for specific domain can be found in [20].

III. A Dublin Core Application Profile to Analysis Patterns

The Dublin Core Application Profile to Analysis Patterns (DC2AP) was developed based on the template proposed in [8] to specify analysis patterns. The main objectives of DC2AP are to improve retrieval and reuse of analysis patterns by means of a description that allows a more precise treatment performed by a computer, providing detailed information about the analysis patterns that were not retrieved by search engines.

A. Mapping between Dublin Core Metadata Elements and Pantoquilho's Analysis Pattern Template

In contrast to the Dublin Core metadata standard, which is generic and therefore aimed to document resources of several domains, the template proposed in [8] by Pantoquilho et al. is designed specifically for the documentation of analysis patterns, so it is rich in specific details of this domain. Due to such level of detail, this template was chosen to be used as a basis for the creation of DC2AP.

An important task for the creation of DC2AP was the realization of a mapping between the elements proposed by the Dublin Core and elements of the template proposed in [8]. In this mapping process, elements of both structures were compared and classified based on their semantic correspondences and some conceptual conflicts were identified. These conflicts are characterized by similar concepts that are expressed differently by each of the mapped structures. After identifying and resolving the existing conceptual conflicts between structures it became possible to combine the elements of Dublin Core and Pantoquilho's template, thus creating a single structure free of redundancies and semantic inconsistencies. Table 1 shows the result of the mapping between the Dublin Core standard and Pantoquilho's template. Although Table 1 shows only the mapping between the elements of Simple Dublin Core and the Pantoquilho's template, the elements contained in Qualified Dublin Core were also considered in this comparative process, but none of them had equivalents in the template used.

Several mappings between elements of the Dublin Core and elements of other structures have already been performed and made available in the literature. An example of such mapping is presented in [21].

B. Addition of New Metadata Elements and Creation of Application Rules

From the mapping described above, all equivalent elements have been identified and combined, thereby allowing that the Pantoquilho's template to be merged with Dublin Core, giving rise to the basic structure of the DC2AP.

Most elements of the Pantoquilho's template that had direct equivalent mapping became element refinements of others from the Dublin Core. This happened because Dublin Core elements are generic and therefore require specializations to compose an application profile for a specific domain. These necessary specializations were made by the elements of the chosen template [8].

During the merge process of the structures in question, two elements from Pantoquilho's template were discarded. The element "Applicability" was discarded because it has semantics very similar to the element "Problem" and therefore was considered redundant. The element "Structural adjustments" was discarded for not fitting in the context of an application profile of metadata, where application rules are well defined, not being necessary to document structural adjustments performed during use of the profile.

Following the fusion of structures, some elements have undergone semantic adjustments and new ones were proposed to complete the set of elements that composes DC2AP. Table 2 presents all the elements that make up the profile proposed by this work.

TABLE I. MAPPING DUBLIN CORE TO PANTOQUILO'S ANALYSIS PATTERN TEMPLATE

Simple Dublin Core element	Pantoquilho's Template element
Title	1. Name 2. Also Known As
Creator	3. History *
Subject	7. Context
Description	5. Problem 6. Motivation 7. Context 8. Applicability 14. Examples * 18. Known Uses *
Publisher	No equivalent
Contributor	3. History *
Date	3. History *
Type	No equivalent
Format	No equivalent
Indentifier	1. Name *
Source	15. Related Patterns *
Language	No equivalent
Relation	13. Anti-Patterns Trap * 15. Related Patterns * 16. Design Patterns *
Coverage	No equivalent
Rights	No equivalent
No equivalent	4. Structural adjustments 9. Requirements 9.1. Functional requirements 9.2. Non-functional requirements 9.3. Dependencies and contributions 9.4. Conflict identification & guidance to resolution 9.5. Priorities 9.6. Participants 10. Modelling 10.1. Structure 10.1.1. Class diagram 10.1.2. Class description 10.2. Behaviour 10.2.1. Collaboration or sequence diagrams 10.2.2. Activity diagrams 10.2.3. State diagrams 10.3. Solution Variants 11. Resulting context 12. Consequences 17. Design guidelines

* Partly equivalent.

As shown in Table 2, DC2AP has some elements for version control of documented patterns and others for the sharing of experiences of use. These features were incorporated into this profile to allow the creation of dynamic collections of analysis patterns, where new improved versions of the patterns can be proposed from the collaboration of experience of usage of them. Moreover, all the versions of the analysis patterns may be related to each other, thereby providing the creation of a repository of analysis patterns rich in details. These resources allow potential users to retrieve the version that best meets their needs more efficiently. All these characteristics are consistent with the proposal of the APRI [5].

The analysis patterns usually have rules controlling their application. After defining the elements that compose DC2AP, it was proposed rules on the obligation, occurrence and type of value of each of the proposed elements. These rules are presented in Table 2 by acronyms, described at the table end.

Due to limited space, the semantic description of each of the elements that compose DC2AP, as well as some details of the rules for applying them are not presented in this paper. However a detailed technical description of this application profile can be obtained at [22].

TABLE II. DC2AP ELEMENTS AND APPLICATION RULES

DC2AP Element and their Application Rules		New
1. Identifier [M] [S] [UNS]		
2. Title [M] [S] [St]	2.1. Alternative Title[O] [Mu] [St]	
3. Creator [M] [Mu] [St]		

DC2AP Element and their Application Rules			New
4. Subject [M] [Mu] [St]			
5. Description [M] [S] [N]	5.1. Problem [M] [S] [St]		
	5.2. Motivation [M] [Mu] [St]	5.2.1. Example [M] [Mu] [St]	
		5.2.2. Known Uses** [O] [Mu] [St]	
	5.3. Context [M] [S] [St]		
6. Publisher [O] [Mu] [St]			
7. Contributor [Cd] [Mu] [St]			
8. Date [M] [S] [N]	8.1. Created [M] [S] [D]		
	8.2. Modified [Cd] [S] [D]		
9. Type [M] [S] [US]	9.1. Notation [M] [S] [St]		YES
10. Format [M] [Mu] [US]			
11. Source [Cd] [S] [UNS]			
12. Language [M] [S] [US]			
13. Relation [Cd] [S] [N]	13.1. Is Version of [Cd] [S] [UNS]		
	13.2. Is Replaced by* [Cd] [Mu] [UNS]		
	13.3. Replaces* [Cd] [Mu] [UNS]		
	13.4. Is Part of [O] [Mu] [UNS]		
	13.5. Has Part [O] [Mu] [UNS]		
	13.6. Is Designed with** [O] [Mu] [UNS]		YES
	13.7. Should Avoid** [O] [Mu] [UNS]		YES
	13.8. Complemented by** [O] [Mu] [UNS]		YES
	13.9. About[Cd] [S] [St]		
14. Coverage [O] [Mu] [St]			
15. Rights [Cd] [Mu] [US]			
16. History* [M] [Mu] [N]	16.1. Event Date [M] [S] [D]		YES
	16.2. Author [M] [Mu] [St]		YES
	16.3. Reason [M] [S] [St]		YES
	16.4. Changes [Cd] [S] [St]		YES
17. Requirements [M] [S] [N]	17.1. Functional Requirements [M] [Mu] [St]		
	17.2. Non-functional Requirements [O] [Mu] [St]		
	17.3. Dependencies and Contributions [M] [S] [St]	17.3.1. Dependency Graph [M] [S] [U]	YES
		17.3.2. Contribution Graph [Cd] [S] [U]	YES
	17.4. Conflict identification & Guidance to Resolution [Cd] [Mu] [St]		
	17.5. Priorities Diagram [M] [S] [U]		
	17.6. Participants [M] [Mu] [St]		
18. Modelling [M] [S] [N]	18.1. Behaviour [M] [S] [N]	18.1.1. Use Case Diagram [M] [S] [U]	YES
		18.1.2. Collaboration/ Sequence Diagrams [M] [Mu] [U]	
		18.1.3. Activity/State Diagrams [O] [Mu] [U]	YES
	18.2. Structure [M] [S] [N]	18.2.1. Class Diagram [M] [S] [U]	
		18.2.2. Class Descriptions [M] [S] U]	
		18.2.3. Relationship Descriptions [M] [Mu] [St]	YES
	18.3. Solution Variants** [O] [Mu] [U]		
19. Resulting Context** [O] [Mu] [St]			
20. Design Guidelines** [O] Mu] [St]			
21. Consequences [M] [S] N]	21.1. Positive [M] [Mu] [St]		YES
	21.2. Negative [M] [Mu] [St]		YES

Rules' Acronyms		
Obligatoriness	Occurrence	Value Type
[M] Mandatory [O] Optional [Cd] Conditional	[S] Single [Mu] Multiple	[St] String [D] Date [U] URI [N] Null [UNS] URI, number or string [US] URI and string

* Version Control element.

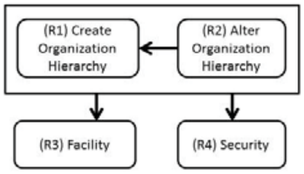
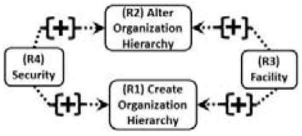
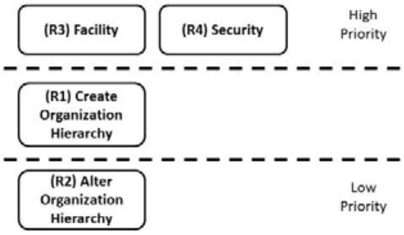
** Experiences Collaboration element.

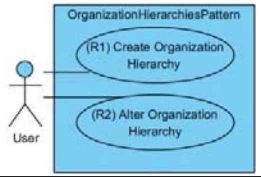
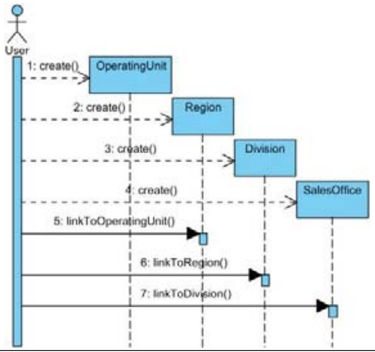
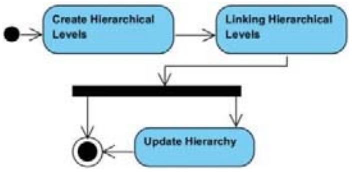
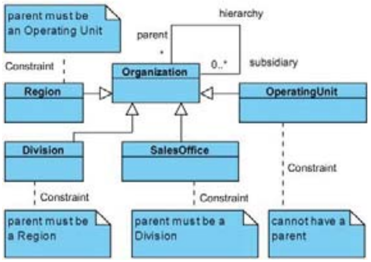
IV. Example

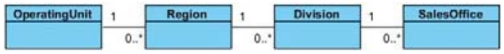
In order to demonstrate the application of metadata profile proposed in this work, Table 3 presents an example that uses DC2AP to specify the well-known Fowler's analysis pattern called Organization Hierarchies, proposed in [3]. Not all DC2AP elements are presented in this specification, since some necessary information for these elements are absent in the original specification made by Fowler.

TABLE III. Organization Hierarchies Pattern Specification

Example of use DC2AP		
1. Identifier: OrganizationHierarchies-v1		
2. Title: Organization Hierarchies	2.1. Alternative Title: Hierarquias de organização	
3. Creator: Martin Fowler		
4. Subject: Companies, Hierarchy, Organizational Structure, Sudsidiaries		
5. Description	5.1. Problem: There are many systems where we need to manage the hierarchy of an organization, registering its subsidiaries and linking them in accordance with the rules of hierarchy. How can we represent this process in a general and abstract way?	
	5.2. Motivation: - An organizational hierarchy has subdivisions like Operating Units, Regions, Divisions and Sales Offices. - Operating Units are divided into Regions. - Regions are divided into Divisions. - Divisions are divided into Sales Offices. - We need to provide a solution easy to be changed because organizations undergo changes in its hierarchy over the course of time.	5.2.1. Example: - A management system of a multinational company, for example, Microsoft. - A management system of a national company that has several branches scattered throughout the territory of a country.
	5.3. Context: This pattern is valuable to institutions or companies that have any subsidiaries. In some cases institutions may have more than one hierarchical organizational structure, but this is not a rule.	
6. Publisher: Lucas F. M. Vegi		
7. Contributor: Lucas F. M. Vegi		
8. Date	8.1. Created: 1997	
	8.2. Modified: 2012-01-15	
9. Type: Analysis Pattern	9.1. Notation: UML	
10. Format: JPEG and XMI		
11. Source: Party Pattern [3]		
12. Language: English		

Example of use DC2AP		
13. Relation	13.2. Is Replaced by: Organization Structure Pattern [3]	
	13.4. Is Part of: Organization Structure Pattern [3]	
	13.8. Complemented by: Party Pattern [3]	
	13.9. About: This analysis pattern can be replaced by Structure Organization pattern, because it contains the pattern Hierarchies Organization specialized with a higher level of details, and thus to more complex organizational hierarchies the analysis patter Organization Structure may be more suitable. The Organization Hierarchies pattern can be complemented by Party pattern because an Organization can be a specialization of Party, as well as the user of the system responsible for registering and changing the hierarchy can also be like that. Thus the Party pattern can be used as a complement of Hierarchies Organization pattern.	
15. Rights:This analysis pattern was originally published in [3].		
16. History	16.1. Event Date: 1997	
	16.2. Author: Martin Fowler	
	16.3. Reason: Criation of this analysis pattern.	
16. History	16.1. Event Date: 2012-01-15	
	16.2. Author: Lucas F. M. Vegi	
	16.3. Reason: Specification of this analysis pattern with the DC2AP metadata profile.	
	16.4. Changes: Structuring the analysis pattern proposed by Fowler in a metadata profile. Within this process new diagrams referring to the solution presented by this pattern were proposed and the initial ideas of Fowler were reorganized into a structure that promotes its retrieval and subsequent reuse.	
17. Requirements	17.1. Functional Requirements: (R1) Create Organizational Hierarchy - The user should be able to register the hierarchical levels of an organization for all its units. (R2) Alter Organizational Hierarchy - The user should be able to change the hierarchical levels of an organization whenever necessary, because as time goes on, with the expansion or contraction of it, such changes will certainly happen.	
	17.2. Non-functional Requirements: (R3) Facility - User must make changes in the hierarchy of an organization in a fast and simple way . (R4) Security – Only users should be allowed to register and make changes in the hierarchy.	
	17.3. Dependencies and Contributions: R2 depends on R1 because only registered hierarchies can be changed. R1 and R2 depends om R3 and R4 because it is desired that all these processes are being made only by authorized users, and in a simple way.	17.3.1. Dependency Graph  (R1) Create Organization Hierarchy (R2) Alter Organization Hierarchy (R3) Facility (R4) Security 17.3.2. Contribution Graph  [+] (R2) Alter Organization Hierarchy [+] (R4) Security (R3) Facility [+] (R1) Create Organization Hierarchy [+]
	17.5. Priorities Diagram  (R3) Facility (R4) Security High Priority (R1) Create Organization Hierarchy (R2) Alter Organization Hierarchy Low Priority	
	17.6. Participants: User	

Example of use DC2AP		
18. Modelling	18.1. Behaviour	18.1.1. Use Case Diagram  OrganizationHierarchiesPattern (R1) Create Organization Hierarchy (R2) Alter Organization Hierarchy User
		18.1.2. Collaboration/Sequence Diagrams  User 1: create() OperatingUnit 2: create() Region 3: create() Division 4: create() SalesOffice 5: linkToOperatingUnit() 6: linkToRegion() 7: linkToDivision()
		18.1.3. Activity/State Diagrams  Create Hierarchical Levels Linking Hierarchical Levels Update Hierarchy
	18.2. Structure	18.2.1. Class Diagram  parent must be an Operating Unit hierarchy parent * 0..* subsidiary Constraint Organization Region OperatingUnit Division SalesOffice Constraint Constraint Constraint parent must be a Region parent must be a Division cannot have a parent
		18.2.2. Class Descriptions: - Organization: This class holds all the attributes common to all possible types of subsidiaries of an organization. - Operating Unit: This class represents the highest hierarchical level of an organization. - Region: This class represents a hierarchical level of an organization. This is the level immediately below an Operating Unit in a hierarchy. - Division: This class represents a hierarchical level of an organization. This is just below a Region in a hierarchy. - SalesOffice: This class represents a hierarchical level of an organization. This is just below a Division in a hierarchy.
		18.2.3. Relationship Descriptions: The self-relationship between the Organization class, super-class of all hierarchical levels, represents that the hierarchical levels communicate with each other. To correctly obey the hierarchical structure, there are restrictions establishing what hierarchical level each of them can communicate directly with.

Example of use DC2AP	
	18.3. Solution Variants:  OperatingUnit Region Division SalesOffice 1 0..* 1 0..* 1 0..*
21. Consequences	21.1. Positive: - This analysis pattern represents an organizational hierarchy easy to change, thus making it useful in different contexts. - To change the hierarchical structure of an organization of this pattern, it is not necessary to change the model structure, but the subtypes and restrictions of the pattern. This makes the pattern flexible and reusable.
	21.2. Negative: - This pattern supports only a single organizational hierarchy, thereby limiting some contexts of use. - If restrictions are not well established for the context of use of this pattern, the self-relationship in it can be dangerous, allowing some hierarchical levels relate directly improperly.

V. Conclusions and Future Work

DC2AP allows a detailed specification of the analysis patterns, since it was developed specifically for this domain. This profile was developed to be integrated into the proposal of Analysis Patterns Reuse Infrastructure (APRI) [5]. Thus it aims to solve the problem of documentation, organization, search and access to analysis patterns.

The use of DC2AP in an APRI allows the creation of digital collections of analysis patterns in the form of pattern catalogues and pattern languages. Through web services proposed by APRI, the analysis patterns specified with DC2AP can be retrieved more quickly and efficiently, offering to potential users an easier access to well-documented analysis patterns, and consequently, with greater potential for reuse.

Because it is a generic metadata standard, Dublin Core allows interoperability between data of different domains, so DC2AP, being an application profile of Dublin Core, can be combined with future works aimed at creating new Dublin Core application profiles to document other types of reusable computational artifacts.

As future works, it is intended to align current description of DC2AP to the Singapore Framework. This framework is a set of descriptive components recommended to document an application profile [23]. With this alignment, DC2AP will fit in the concept of machine-processable application profile and thus can serve as basis for the definition and implementation of web services proposed in APRI for search, visualization, application and contribution of use experience of analysis patterns.

Acknowledgment

This work is partially financed by Brazilian funding agencies: FAPEMIG, CNPq and CAPES. The authors also acknowledge the financial support of the company Sydle.

References

[1] B. Boehm, and V. Basili, "Software defect reduction top 10 list," IEEE Computer, vol. 34, n. 1, January 2001.

[2] E. B. Fernandez, and X. Yuan, "Semantic Analysis Patterns," Proc. of the 19th Int. Conf. on Conceptual Modeling (ER 2000), LNCC vol. 1920. Springer, pp. 183-195, 2000.

[3] M. Fowler, Analysis Patterns: reusable object models. Addison-Wesley Publishing, 1997.

[4] N. Blaimer, A. Bortfeldt, and G. Pankratz, "Patterns in object-oriented analysis," Working Paper No. 451, Faculty of Business Administration and Economics, University of Hagen (Germany), 2010.

[5] L. F. M. Vegi, D. A. Peixoto, L. S. Soares, J. Lisboa-Filho, and A. P. Oliveira, "An infrastructure oriented for cataloging services and reuse of Analysis Patterns," Proc. of BPM 2011 Workshops (rBPM 2011), LNBIP vol. 100, Part 4. Springer, pp. 338 – 343, 2012.

[6] J. Nogueras-Iso, F. J. Zarazaga-Soria, and P. R. Muro-Medrano, Geographic information Metadata for Spatial Data Infrastructures: resources, interoperability and information retrieval. Springer, 2005.

[7] DCMI - Dublin Core Metadata Initiative. [Online]. Avaiable: http://www.dublincore.org

[8] M. Pantoquilho, R. Raminhos, and J. Araújo, "Analysis Patterns specifications - filling the gaps," Proc. of the 2nd Viking PLoP. pp. 169-180, 2003.

[9] R. Raminhos, M. Pantoquilho, J. Araújo, and A. Moreira, "A systematic Analysis Patterns specification," Proc. of the 8th International Conference on Enterprise Information Systems (ICEIS). pp. 453-456, 2006.

[10] D. C. Hay, Data Model Patterns: convention of thoughts. Dorset House Publishing: New York, USA, 1995.

[11] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design patterns: elements of reusable object-oriented software. Addison-Wesley Publishing, 1994.

[12] J. Lisboa-Filho, C. Iochpe, and K. A. Borges, "Analysis Patterns for GIS data schema reuse on urban management applications," CLEI Electronic Journal, vol. 5, n. 2. pp. 01-15, 2002.

[13] E. B. Fernandez, X. Yuan, "An Analysis Pattern for invoice processing," Proc. of the 16th Conference on Pattern Languages of Programs (PLoP). pp. 01-10, 2009.

[14] G. Meszaros, and J. Doble, "A pattern language for pattern writing," in Pattern languages of program design 3, R. C. Martin, D. Riehle, and F. Buschmann, Eds. Addison-Wesley: Boston, USA, 1997, pp. 529-574.

[15] U.S. Library Of Congress, "MARC standards," Network Development and MARC Standards Office. 2004. [Online]. Avaiable: http://www.loc.gov/marc/

[16] ISO, Geographic information - Metadata. ISO 19115:2003, International Organization for Standardization. 2003.

[17] NISO U.S. - National Information Standards Organization, The Dublin Core Metadata element set: an American national standard. NISO Press, 2001.

[18] DCMI - Dublin Core Metadata Initiative, "Using Dublin Core - Dublin Core Qualifiers". 2005. [Online]. Avaiable: http://dublincore.org/documents/usageguide/qualifiers.shtml

[19] K. Coyle, and T. Baker, "Guidelines for Dublin Core Application Profiles." 2009. [Online]. Avaiable: http://dublincore.org/documents/profile-guidelines/

[20] DCMI - Dublin Core Metadata Initiative, "Dublin Core Collections Application Profile." 2004. [Online]. Avaiable: http://dublincore.org/groups/collections/collection-application-profile/

[21] U.S. Library Of Congress, "Dublin Core to MARC Crosswalk,". Network Development and MARC Standards Office. 2008. [Online]. Avaiable: http://www.loc.gov/marc/dccross.html

[22] L. F. M. Vegi, "Technical description of Dublin Core application profile to Analysis Patterns (DC2AP)". 2012. [Online]. Available: http://purl.org/dc2ap/TechnicalDescription

[23] M. Nilsson, T. Baker, and P. Johnston, "The Singapore Framework for Dublin Core Application Profiles." 2008. [Online]. Avaiable: http://dublincore.org/documents/singapore-framework/

Bridging KDM and ASTM for Model-Driven Software Modernization

Gaëtan Deltombe
Netfective Technology Software
32, avenue Léonard de Vinci
33600 – Pessac, France
g.deltombe@netfective.com

Olivier Le Goaer, Franck Barbier
University of Pau
Avenue de l'université
64000 – Pau, France
{olivier.legoaer, franck.barbier}@univ-pau.fr

Abstract

Standardizing software modernization techniques has lead to the KDM (Knowledge Discovery Metamodel). This metamodel represents several application aspects (code, architecture, etc.), while transforming them into renewed versions. On the other hand, ASTM (the Abstract Syntax Tree Metamodel) has been recently released. It focuses on the parsing of text-based files written in a given language. In practice, KDM and ASTM are intended to be used jointly when modeling source code with formal links to other software features like components, user interfaces, etc. However, the link between ASTM and KDM is often fuzzy, or even unestablished since KDM is in charge of synthesizing all captured software artifacts. This has negative effects on the attainable level of automation and on the completeness of a software modernization project. To overcome this limitation, this paper introduces SMARTBRIDGE *as a means to reconcile both standards.*

1. Introduction

Modernization is at the heart of many software organizations that seek to migrate from obsolete or aging languages and platforms to more modern environments. Modernization projects have historically focused on transforming technical architectures; that is, moving from one platform to another and / or from one language to another [2]. This is achieved through code translation or various refactoring exercises such as restructuring, data definition rationalization, re-modularization or user interface replacement.

Meanwhile, model-driven development (MDD) is gaining increasing acceptance; mainly because it raises the level of abstraction and automation in software construction. MDD techniques, such as metamodeling and model transformation, not only apply to the creation of new software systems but also can be used to help existing systems evolve [7, 1]. These techniques can help reduce software evolution costs by automating many basic activities, including code manipulation.

There is currently great activity addressing model-driven modernization issues, for which the OMG's task force on modernization [16] plays a major role. It aims at making precise inventories of various kinds of legacy artifacts in order to propose more or less automatic model-driven migrations of applications by means of interoperable tools. This way, all discovered artifacts are considered as full-fledged models. This is an important shift compared to classical approaches, which do not consider abstract representations as perennial and reusable assets.

The context of this paper is rooted in the research work conducted by the European REMICS (www.remics.eu) project [11]. This project seeks an end-to-end model processing chain to transform legacy applications/information systems into services in the Cloud. Several modeling languages are operated in the project, within both reverse and forward engineering activities. To define and support this chain in tools, modeling language semantic gaps must be fulfilled. More specifically, the reverse engineering activity must be committed to go from the source code towards technologically-neutral UML models, which can then be consumed by a wide range of third-party MDD tools. The forward engineering activity starts from UML models and next uses SoaML (www.soaml.org) and CloudML (a forthcoming OMG standard that is currently specified within REMICS). So, between reverse and forward, a suite of models conforming to (i.e., instances of) OMG standard metamodels are involved, each aiming at representing the initial source code at different levels of abstraction, along with various refinements/quality levels [8]. The general idea is that a portfolio of metamodels and transformation chains are pre-implemented in a tool to support an intelligible seamless reverse and forward engineering process. Because of its industrial impact, REMICS complies to worldwide standards. Nonetheless, these standards lack large-scale experimentation, thus requiring adaptations in their core to really achieve a high degree of automation when targeting a

Legacy2Cloud logic. In this scope, this paper stresses reverse engineering activity, by showing and illustrating how modeling language can be rationally treated.

In reverse engineering, top-down approaches promote a recovery process that is conducted through architectural knowledge. On the other hand, bottom-up approaches [4] are more pragmatic, since reverse engineering activities lean on source code that is undoubtedly the most rich, self-contained and straightforwardly available material. REMICS focuses on the latter due to the dispersion, or even absence of knowledge. So, model transformations amount to inferring such knowledge from model details and from the expressiveness of their associated modeling languages. Risks occur when the knowledge in models has to move from one language formalism to another. For turning code written in a given programming language into a language-agnostic model, the OMG's task force on modernization puts forward two metamodels: the ASTM [13], which is a metamodel dedicated to syntax trees, and the KDM [12], which is a more global metamodel where the sub-parts deal with a wider spectrum of program-level elements (i.e., user interfaces, data, functions/services, running platform). Unfortunately, KDM and ASTM are not clearly linked to each other. They aim at being complementary but practice shows an unsound, ill-formalized dependency between them. To bridge this gap, this paper discusses, develops and illustrates an intermediary metamodel called SMARTBRIDGE, which allows a roundtrip relationship between KDM and ASTM and hence supports multiple iterations during the reverse engineering activity.

The remainder of the paper is organized as follows: in Section 2 we describe the motivations and the various specifications that gave birth to KDM and ASTM respectively. The SMARTBRIDGE is then detailed in Section 3, including an enumeration of the metaclasses and metarelationships involved in the junction of ASTM and KDM. To make these ideas more concrete, we provide in Section 4 a demonstration of SMARTBRIDGE on a tiny COBOL snippet. Section 5 gives an overview of the related works on model-driven modernization. We conclude this paper in Section 6.

2. Problem of Interest

Recently, MDD standardization has stressed modernization with the special objective of providing new concepts, tools and processes when moving legacy software to renewed applications/information systems running on top of the most up-to-date technologies. The idea behind that is to switch between different technical spaces [10] by underestimating the purely ”grammarware” approach in favor of the ”modelware” approach. Beyond traditional code-to-code approaches, model-centric approaches are those promoted by MDD in general: (a) conformance to well-defined metamodels, (b) powerful transformation techniques, (c) easier integration of various concerns. Standards in this area also emphasize wider interoperability. In this case, standard-compliant models should be exchangeable between tools involved in the migration chain. These tools process software elements having different shapes. These shapes include the lowest levels (code), as well as the highest: business rules process extraction and interpretation, dealing with the software architecture and components, exhibiting services (business functionalities), etc.

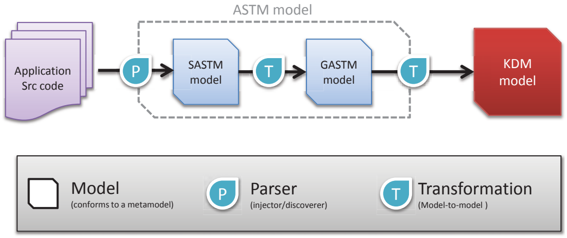


Figure 1. Reverse Engineering process according to the OMG's task force on modernization.

2.1. MDD-based Reverse Engineering

A modernization process encompasses two stages: reverse engineering and forward engineering. The forward engineering stage starts from a Platform-Independent Model (PIM) that serves as the basis for code generation. The reverse engineering stage extracts elements from legacy code and data description, rendering them into a Platform-Specific Model (PSM). KDM is the support candidate for representing PSMs by using ASTM as sub-support for the precise and comprehensive representation of a software system. The creation of PIMs (or technology-neutral models) is favored by the construction of formal mappings between KDM/ASTM on one side and UML (for PIMs) on the other side. More generally, metamodeling fosters the description of discrete steps to show how, at the very beginning, the rough code may be interpreted and analyzed in terms of architectural incidence: operating system adherences, business value discovery strategies, etc.

In this article, KDM is the pivot metamodeling language for representing entire enterprise software systems; including source code of course, but not exclusively. As a common intermediate representation for existing software systems, KDM is a good support for refactoring, the derivation of metrics and the definition of specific viewpoints. Figure 1 depicts the full reverse engineering process promoted by MDD modernization standards.

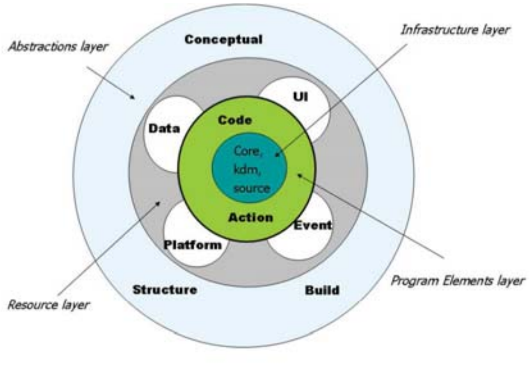


Figure 2. KDM consists of 12 packages arranged into 4 layers

2.2. MDD technologies for modernization

The concomitant use of KDM and ASTM requires a clear understanding of their current capabilities.

2.2.1 Knowledge Discovery Metamodel (KDM)

The architecture of KDM is arranged into four layers, namely the Infrastructure Layer, Program Elements Layer, Runtime Resources Layer and Abstractions Layer (Figure 2). Each layer is dedicated to a particular viewpoint of an application. In this paper we are only interested in the two main layers: the Runtime Resources Layer and the Program Element Layer. These layers allow one to represent the user interfaces, data and code of the legacy application.

The KDM Runtime Resources Layer The Runtime Resource Layer is composed of several packages (`Data`, `UI`, `Event` and `Platform`). However, we will concentrate on the `Data` and `UI` packages herewith. The first one is used to represent the organization of persistent data, especially to describe complex data repositories (e.g., record files, relational schemas, . . .). The second one is used to represent the structure of user interfaces and the dependencies between them in terms of interactions and sequences.

The KDM Program Elements Layer Special attention is paid to the Program Elements Layer, whichis concerned with program-level artifacts. The Program Elements Layer is composed of the `Code` and `Action` packages. The `Code` package represents programming elements as determined by programming languages (data types, procedures, classes, methods, variables, etc.), while the `Action` package describes the low-level behavior elements of applications, including detailed control and data flows resulting from statement sequences. In this scope, KDM is recognized as a way of representing the source code, if only at the execution flow level.

2.2.2 Abstract Syntax Tree Metamodel (ASTM)

As a complement, ASTM has been developed in accordance with the theory of languages to support the representation of source code. In fact, ASTM is composed of the GASTM (Generic Abstract Syntax Tree Metamodel), a standardized language-independent metamodel and the SASTM (Specific Abstract Syntax Tree Metamodel), a user-defined metamodel closely connected with a particular language (Java, COBOL and so on).

Generic Abstract Syntax Tree Metamodel (GASTM) GASTM enables the representation of the code without any language specificity. GASTM contains all of the common concepts of existing languages in the form of metatypes. Parsing some files means instantiating these metatypes along with creating links in order to model semantic dependencies between text pieces. The goal of GASTM is to provide a basis for SASTM in order to later help users to define the domain-specific features of the code to be parsed. The key achievement is to avoid an unintelligible separation (especially in terms of representations) between the generic and specific characteristics of the code. Besides, the (annotated) distinction in models between generic versus specific parts, is highly valuable at processing time (see below).

Specific Abstract Syntax Tree MetaModel (SASTM) SASTM is constructed through metatypes/metarelationships on the top of GASTM. This task is assigned to ASTM practitioners. It first leads to constructing a metamodel from scratch that is compatible with the legacy language/technology to be dealt with. The main goal of SASTM is to represent code peculiarities. Next, parsing the code amounts to distinguishing between generic and specific aspects, and thus instantiating GASTM or SASTM. The formal interrelation between the two metamodels ensures that models (or their respective instances which represent a given business case) are also consistently linked together based on (fully explicit) comprehensive links.

2.3. Realizing modernization

OMG provides a set of standard specifications for software modernization but fails to provide a guideline for the practitionners. As such, we experimented on the aforesaid technologies while endeavoring to adhere to the – somewhat idealized – process depicted in Figure 1.

2.3.1 Complementarity of MDD technologies

The complementarity of KDM and ASTM resides in the possible code level representations and the different operations that can be applied. On one hand, ASTM permits one to represent a given code source at procedure level, in the form of a syntax-tree whose production has involved a user-defined SASTM: code specificities are taken into account. On the other hand, KDM is used to represent the code at flow level (e.g., data inputs, data outputs, sequences). In fact, a flow level representation provides a direct support for flow analysis and the refactoring strategies thereof. In addition, any reverse engineering process relying on KDM models is reusable, whatever the source technology may be. So, ASTM deals with common parsing issues, while KDM deals with another viewpoint; thus creating the link with other facets like user interfaces, components and so on.

2.3.2 Discretization of the process

The modernization process proposed in this paper is divided into three codified steps:

1. The first step consists in the abstraction of data, code and user interfaces from the legacy material. This is transformed into several technology-specific models. Practically speaking, for each artifact we define its own parser. The parsing outputs are concrete syntax trees (CSTs). Next, these CSTs are transformed into ASTs, each conforming to predefined SASTM metamodels. Starting from chunks of text and ending up as models, this global process is often called ”Injection” in the MDD jargon.

2. This second step consists in the transformation of SASTM models into KDM models, with respect to user interfaces, data and code packages. The goal of this model transformation is to eliminate all technological specificities of the input code model. For that purpose, a reformulation is sometimes required.

3. The third step consists in transforming the code-related and data-related KDM models into UML models, thus generating models in a widely accepted format with their associated graphical notation. There are no technical difficulties except that of choosing between the (existing) concurrent UML-like Java representations that are tolerated by today’s modeling tools.

2.3.3 Current limitations

Our experience has led us to conclude that a modernization project is not a straightforward process, but instead a strongly iterative process, including the re-examination of the models’ parts and their mutual enrichment. We thereby advocate roundtrip capability as to enable information exchange and knowledge propagation within the process, at any level or step. At least two reasons explain this.

First, real modernization requires incorporating additional or derived knowledge into models, either automatically or manually. The most prominent examples are the following:

- Detection of code patterns. The purpose is to recognize sets of object codes that will facilitate refactorings, especially when targetting a completely new architecture.

- Determination of components fate (and the traceability thereof). In accordance with application experts, the purpose is to stamp components that will be migrated and those that will be instead replaced by off-the-shelf components.

- Extraction of business rules. The purpose is to gain a better comprehension of the business logic that underlies a huge amount of lines of code.

- Multi-view modeling. The purpose is to set up the right semantic relationships between the interrelated views of the system that is being reversed (such as user interface, business code and those data structures used for persistence).

Secondly, the strict discretization of the modernizing process envisioned by OMG is not realistic when aiming at providing a modernization CASE tool with a good user-experience level. Indeed, the way a user perceives the computer-aided modernization is an important question, from a tooling viewpoint. Typically, the new knowledge must be impacted to the code model and showed to the user. Thus, some code blocks within a code editor will be highlighted as patterns, while some others will be tagged as “to be replaced”, etc. In other words, the new knowledge derived from the KDM-level must be brought up to the ASTM-level.

3. Proposed research

The observations above stress the necessity to take advantage of the two worlds, while providing roundtrip capability. This means preserving the support for architecture, data, user interfaces (even metrics) that is provided by KDM; as well as achieving the level of details allowed by ASTM. Therefore, we suggest that ASTM and KDM could be interrelated thanks to a bridge that we have dubbed SmartBridge. This bridge will ensure inter-relationships as well as intra-relationships. The former deal with links between two distinct metamodels: KDM and ASTM. The latter deal with links between the layers that belong to the KDM metamodel itself.

Building SmartBridge has led us to focus on three important features:

1. interfacing KDM and ASTM via joint points
2. ensuring navigability between them
3. mapping the different KDM layers: code, data and user interface

3.1. Interfacing

The linkage between the low level code representation allowed by ASTM and the more abstract level allowed by KDM is not natively provided. To overcome this issue, SmartBridge interposes a number of meta-classes that are showed in table 1.

ASTM	SmartBridge	KDM
TypeDefinition	EDataType	DataType
DataDefinition	EDataElement	DataElement
TypeDeclaration	EDataType	DataType
AggregateTypeDefinition	EDataType	RecordType
AggregateTypeDeclaration	EDataType	RecordType
FunctionDeclaration	EControlElement	CallableUnit
FunctionDefinition	EControlElement	CallableUnit
VariableDeclaration	EDataElement	StorableUnit
VariableDefinition	EDataElement	StorableUnit
Statement	EActionElement	ActionElement

Table 1. Meta-classes for interfacing ASTM and KDM

Achieving this interfacing calls for an extension point. It turns out that such an extension point was provided in the KDM specification by way of the meta-class `CodeElement` which belongs to `Code` package. Figure 3 shows how SmartBridge exploits this extension point in order to introduce the required metaclasses to bridge toward ASTM.

Based on the KDM CodeElement meta-class, SmartBridge defines several meta-classes which are sub-classes of `EElement`. These meta-classes are `ECodeElement`, `EActionElement`, `EDataElement`, `EControlElement` and `EDatatype`. Figure 4 presents an interfacing example between an ASTM `Statement` and its KDM corresponding representation.

3.2. Navigability

We decided to provide a bidirectional navigation capability between KDM and ASTM in order to be able to obtain an ASTM code representation from an abstract code element representation, and conversely. The navigability implemented in SmartBridge is inspired by the relationship mechanism used in the KDM specification [12]. The KDM code package provides a natural extension point for this relationship mechanism through the `CodeRelationship` meta-class (Fig. 5). SmartBridge specializes this extension point in two metaclasses: `ERelationship` and `EAggregateRelationship`.

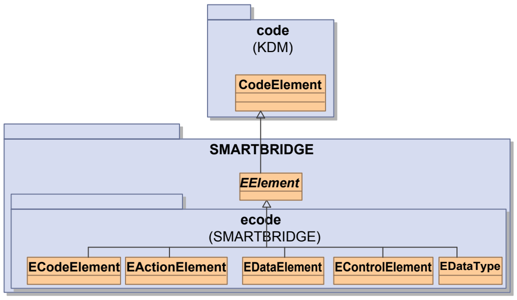


Figure 3. CodeElement extension point

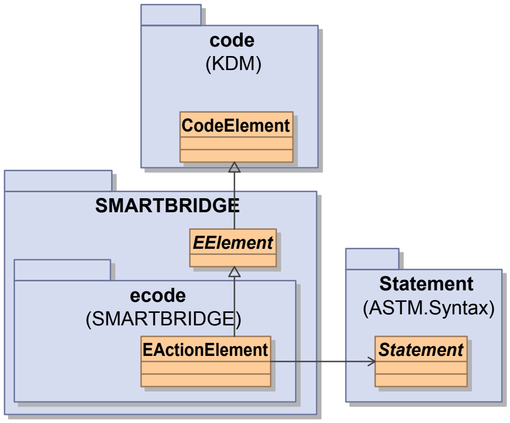


Figure 4. EActionElement meta-class

3.2.1 ERelationship

The `ERelationship` meta-class defines the navigability between KDM meta-classes and the SmartBridge meta-classes. An `ERelationship` instance is used in order to navigate from a KDM `CodeElement` to a SmartBridge `EElement` and vice versa. This relationship is exclusively used for a one-to-one relationship, like a function representation (ASTM `FunctionDefinition`), which corresponds exactly to one abstract representation (KDM `CallableUnit`).

3.2.2 EAggregateRelationship

The `EAggregateRelationship` meta-class defines the navigability one-to-many between a KDM element and many SMARTBRIDGE elements. This one-to-many relationship is useful in order to represent KDM abstract elements (like KDM `ActionElement`) using many concrete ASTM ones (like `Statement` metatype).

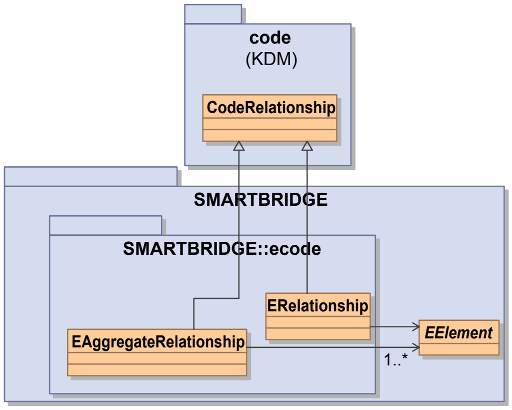


Figure 5. CodeRelationship extension point

3.3. Mapping

The need to provide a mapping between the type definition contained in the legacy source code and the data definition in a database or in the user interface is extremely strong. The KDM meta-model does not provide the concept of specific mapping between the different layers: code, data and user interface. For this reason SMARTBRIDGE introduces this lacking concept through a new package: the `Mapping` package. It defines the link between the source code and the UI from one side and the source code and the data from the other side. The package introduces a new KDM code model called `MappingModel` (c.f. Figure 6). This model contains the existing mapping set representing the data structures, the UI and the source code.

SMARTBRIDGE includes the mapping concept through the abstract meta-class `MappingElement`. Thereby two concrete meta-classes are defined to map the KDM code package element with the KDM data package element `DataMapping` and the KDM ui package element `UIMapping` (Fig. 7).

Thanks to these three important features, i.e. navigability, interfacing and mapping, SMARTBRIDGE fills the gap between the KDM and the ASTM metamodels. This is essential to reach a suitable level of abstraction which enables a better understanding of the legacy source code, and hence an easier modernization activity.

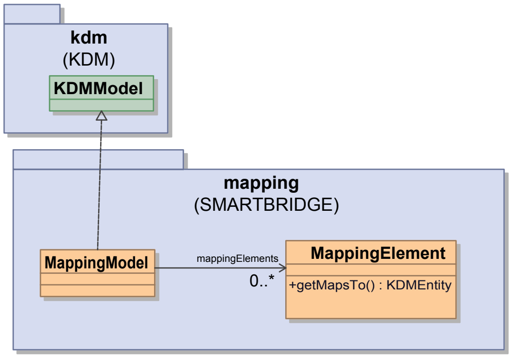


Figure 6. Mapping Model

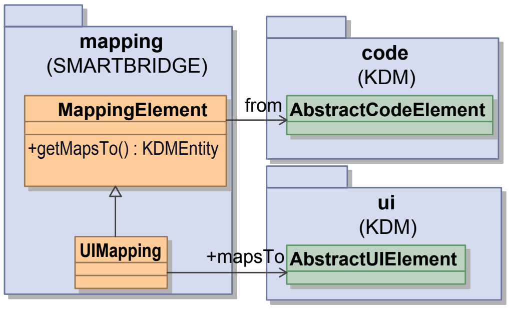


Figure 7. Focus on UI Mapping

4. Working example

This section provides an illustration of SMARTBRIDGE for the modernization of the COBOL legacy code. This illustration especialy focuses on the benefits of interfacing KDM and ASTM. For the sake of simplicity, the following illustrations are based on the tiny COBOL code snippet below:

```
MOVE "sample␣error" Error-Message
PERFORM FERROR
```

The modernization process is based on the three following steps:

1. Text to Model transformation (aka. Injection)
2. Interfacing
3. Abstraction refining

4.1. Injection

The first step in the modernization process as discussed in Section 2.3.2, is to obtain an abstract syntax tree conforming to the ASTM metamodel. This step is crucial in extracting relevant information contained in the code. Thus to build this ASTM model, a COBOL grammar must be used to parsing the legacy source code. The result of this parsing phase is then used to obtain the SASTM (Right part on Fig. 8).

The obtained SASTM model contains specificities closely related to the COBOL programming language like `MOVE`, `PERFORM`, etc. In order to obtain more abstract elements, the transformation of this model into a GASTM one is required (not showed here).

4.2. Interfacing

This second step aims at interfacing between ASTM and KDM by using our SMARTBRIDGE metamodel in order to progressively raise the abstraction level and also to iteratively enrich the target KDM model. This interfacing is illustrated in Figure 8.

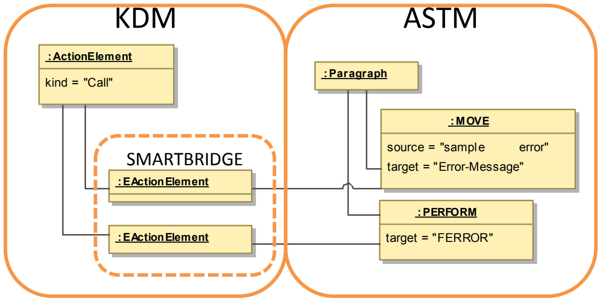


Figure 8. Interfacing example

4.3. Abstraction refining

The third and last step of our modernization process aims at reaching a first abstraction level and at enriching the initial model. In order to accomplish this step, a true source code comprehension is necessary. In fact the `MOVE` instruction (see COBOL code sample) initializes the value `Error-Message`. This field represents the parameter. The KDM representation of the `PERFORM` statement is a CallableUnit call with a parameter value of '*sample error*'. Figure 9 shows the KDM representation obtained using SMARTBRIDGE.

5. Related Works

In this section, we study other works that address modernization issues through model-driven technologies. Logically, we pay a special attention to those focusing on the code level of legacy applications. In contrast with KDM-compliant approaches, approaches relying on proprietary metamodels, tailored for particular usages do exist.

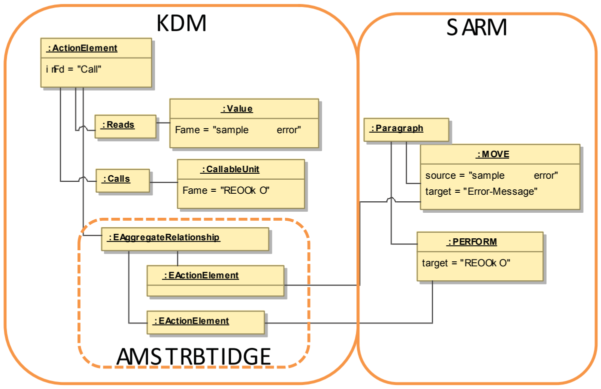


Figure 9. Abstraction refining example

5.1. KDM-uncompliant

Reus *et al.* in [15] propose a MDA process for software migration where they parse the text of the original system and build a model of the abstract syntax tree. This model is then transformed into an intermediate langage dubbed GenericAST that can be translated into UML.

In [6], the authors summarize the use and impact of the TGraph technology in Reverse Engineering. TGraphs [5] are directed graphs whose vertices and edges are typed, attributed, and ordered. In fact, representing source code as a typed graph can be rephrased as representing code as a model conforming to a metamodel. From this point of view, metamodels used by parsers are designed from scratch.

Izquierdo and Molina developed the Gra2MoL approach [3], where a model extraction process is considered as a grammar-to-model transformation, so mappings between grammar elements and metamodel elements are explicitly specified. Beyond the technical aspects of the proposed transformation language, one may notice that the target metamodels are user-defined.

Fleurey *et al.* describe in [9] a model-driven migration process in an industrial context. For that purpose, a tool suite for model manipulation is used as a basis for automating the migration. The reverse engineering step moves from a code model (output of the parsing) to a PIM, which is implemented by model transformations from a legacy language meta-model (e.g., COBOL) to a pivot metamodel. The pivot metamodel is called ANT and contains packages to represent data structures, actions, UIs and application navigation.

5.2. KDM-compliant

In [14], Perez-Castillo *et al.* propose a technique that recovers code-to-data links in legacy systems based on relational databases and enable one to represent and manage these linkages throughout the entire reengineering process. The proposal follows the ADM approach by leveraging KDM, especially the code package of the Program Elements Layer where SQL sentences have been modeled through a KDM extension. In this case, it is not an actual metamodel, it is a profile one instead.

MoDisco (Model Discovery) [1] is the model extraction framework part of the Eclipse GMT project (www.eclipse.org/gmt). This framework is currently under development and provides a model managing infrastructure dedicated to the implementation of dedicated parsers ("discoverers" in MoDisco terminology). A KDM-based metamodel, a metamodel extension mechanism and a methodology for designing such extensions are also planned.

6. Conclusion

ASTM and KDM complement each other in modeling software systems' syntax and semantics. In this article, we propose to fill the gap between the two by introducing SMARTBRIDGE in order to remain in the scope of ADM and hence ensure the interoperability of the outputs of MDD reverse engineering activities.

Gluing ASTM and KDM aims at overcoming the main flaw of a strict discretization of the modernization process, thus enabling roundtrip engineering. It also balances out the purely low-level representation of the legacy material supported by ASTM and the higher abstraction level supported by KDM. Hence, SMARTBRIDGE has been implemented within BLUAGE® (www.bluage.com) and has been proven to better represent code, while maintaining good architectural representation; rather than using KDM and ASTM separately. As such, we have demonstrated – in a tooling purpose – that SMARTBRIDGE supplies a practical answer to the traceability from end-to-end issue, along with knowledge propagation at every step.

We are currently improving SMARTBRIDGE with additional features. Indeed, similarly to the glue ASTM-KDM, we plan to fill the gap with further OMG ADM metamodels like SPAP (Software Patterns Analysis Package), SMM (Software Metrics Metamodel), etc. These new relationships will make it possible to handle the different aspects of a legacy system as a cohesive whole.

Acknowledgements

This work has been funded by the European Commission through the REMICS project (www.remics.eu), contract number 257793, within the 7th Framework Programme.

References

[1] H. Bruneliere, J. Cabot, F. Jouault, and F. Madiot. Modisco: a generic and extensible framework for model driven reverse engineering. In *Proceedings of the IEEE/ACM international conference on Automated software engineering*, ASE '10, pages 173–174, New York, NY, USA, 2010. ACM.

[2] E. J. Chikofsky and J. H. Cross II. Reverse engineering and design recovery: A taxonomy. *IEEE Softw.*, 7:13–17, January 1990.

[3] J. Cnovas Izquierdo and J. Molina. A domain specific language for extracting models in software modernization. In R. Paige, A. Hartman, and A. Rensink, editors, *Model Driven Architecture - Foundations and Applications*, volume 5562 of *Lecture Notes in Computer Science*, pages 82–97. Springer Berlin / Heidelberg, 2009.

[4] S. Ducasse and D. Pollet. Software architecture reconstruction: A process-oriented taxonomy. *IEEE Trans. Softw. Eng.*, 35:573–591, July 2009.

[5] J. Ebert and A. Franzke. A declarative approach to graph based modeling. In *Proceedings of the 20th International Workshop on Graph-Theoretic Concepts in Computer Science*, WG '94, pages 38–50, London, UK, 1995. Springer-Verlag.

[6] J. Ebert, V. Riediger, and A. Winter. Graph technology in reverse engineering: The tgraph approach. In *Workshop Software Reengineering*, pages 67–81, 2008.

[7] L. Favre. *Model Driven Architecture for Reverse Engineering Technologies: Strategic Directions and System Evolution*. Premier Reference Source. Igi Global, 2010.

[8] F.Barbier, G.Deltombe, O.Parisy, and K.Youbi. Model driven reverse engineering: Increasing legacy technology independence. In *The 4th India Software Engineering Conference*, Thiruvanantpuram, India, February 2011. CSI ed.

[9] F. Fleurey, E. Breton, B. Baudry, A. Nicolas, and J.-M. Jézéquel. Model-driven engineering for software migration in a large industrial context. In *MoDELS*, pages 482–497, 2007.

[10] I. Kurtev, J. Bézivin, and M. Aksit. Technological spaces: An initial appraisal. In *CoopIS, DOA 2002 Federated Conferences, Industrial track*, 2002.

[11] A. S. F. B. Mohagheghi Parastoo, Berre Arne Jrgen and G. Benguria. Reuse and migration of legacy systems to interoperable cloud services - the remics project. In *Proceedings of Mda4ServiceCloud'10 at the Sixth European Conference on Modelling Foundations and Applications*, ECMFA '10, June 2010.

[12] OMG). Knowledge discovery metamodel - version 1.3. http://www.omg.org/spec/KDM/1.3, 2011.

[13] OMG. Syntax tree metamodel - version 1.0. http://www.omg.org/spec/ASTM/1.0, 2011.

[14] R. Perez-Castillo, I. G.-R. de Guzman, O. Avila-Garcia, and M. Piattini. On the use of adm to contextualize data on legacy source code for software modernization. *Reverse Engineering, Working Conference on*, 0:128–132, 2009.

[15] T. Reus, H. Geers, and A. van Deursen. Harvesting software systems for mda-based reengineering. In *ECMDA-FA*, pages 213–225, 2006.

[16] W. Ulrich. A status on omg architecture-driven modernization task force. In *Proceedings EDOC Workshop on Model-Driven Evolution of Legacy Systems*, Monterey, California, USA, 2004. IEEE Computer Society.

Modal ZIA, Modal Refinement Relation and Logical Characterization

Zining Cao[1,2,3]

[1] College of Computer Science and Technology
Nanjing University of Aero. & Astro., Nanjing 210016, P. R. China
[2] Provincial Key Laboratory for Computer Information Processing Technology
Soochow University, Suzhou 215006, P. R. China
[3] National Key Laboratory of Science and Technology on Avionics System Integration
Shanghai 200233, P. R. China
Email: caozn@nuaa.edu.cn

***Abstract*—In this paper, we propose a specification approach combining modal transition systems, interface automata and Z language, named modal ZIA. This approach can be used to describe temporal properties and data properties of software components. We also study the modal refinement relation on modal ZIAs. Then we propose a logic *MZIAL* for modal ZIAs and give a logical characterization of modal refinement relation. Finally, we present a sublogic of *MZIAL*, named *muZIAL*, and give a model checking algorithm for finite modal ZIA.**

***Index Terms*—interface automata; Z notation; modal transition systems; refinement relation; modal logic; model checking**

I. Introduction

Modern software systems are comprised of numerous components, and are made larger through the use of software frameworks. Such software systems exhibit various behavioral aspects such as communication between components, and state transformation inside components. Formal specification techniques for such systems have to be able to describe all these aspects. Unfortunately, a single specification technique that is well suited for all these aspects is yet not available. Instead one needs various specialized techniques that are very good at describing individual aspects of system behavior. This observation has led to research into the combination and semantic integration of specification techniques.

Interface automata is a light-weight automata-based languages for component specification, which was proposed in [1]. An interface automaton (IA), introduced by de Alfaro and Henzinger, is an automata-based model suitable for specifying component-based systems. IA is part of a class of models called interface models, which are intended to specify concisely how systems can be used and to adhere to certain well-formedness criteria that make them appropriate for modelling component-based systems.

Z [15] is a typed formal specification notation based on first order predicate logic and set theory. The formal basis for Z is first order predicate logic extended with type set theory. Using mathematics for specification is all very well for small examples, but for more realistically sized problems, things start to get out of hand. To deal with this, Z includes the schema notation to aid the structuring and modularization of specifications. A boxed notation called schemas is used for structuring Z specifications. This has been found to be necessary to handle the information in a specification of any size. In particular, Z schemas and the schema calculus enable a structured way of presenting large state spaces and their transformation.

Modal transition systems [11], i.e., can be modelled as automata whose transitions are typed with may and must modalities. A modal transition system represents a set of models; informally, a must transition is available in every component that implements the modal transition system, while a may transition needs not be. In [14], a unification of interface automata and modal transition systems was presented.

In this paper, we present a new specification language which combines modal transition systems, interface automata and Z language. Interface automata are a kind of intuitive models for interface property of software components. We combine modal transition systems, interface and Z to describe modal property, temporal property and data property in a unifying model. We give the definition of modal ZIA. Roughly speaking, a modal ZIA is in a style of modal interface automata but its states and transitions are described by Z language. Furthermore, we define the modal refinement relation between modal ZIAs and give some propositions of such modal refinement relation. Then we present a logic for modal ZIA and give a logical characterization of modal refinement relation. Finally, we give a model checking algorithm for finite modal ZIA.

This paper is organized as follows: Section 2 gives a brief review of modal transition systems, interface automata and Z language. In Section 3, we propose a specification language-modal ZIA. Furthermore, the modal refinement relation for modal ZIA is presented and studied. In Section 4, we present a logic *MZIAL* for modal ZIA and give a logical characterization of modal refinement relation. In Section 5, we present a sublogic of *MZIAL*, named *muZIAL*. Then we give a model checking algorithm for finite modal ZIA. The paper is concluded in Section 6.

II. Overview of Modal Transition Systems, Interface Automata and Z Language

In this section, we give a brief review of modal transition systems, interface automata and Z language.

Modal transition systems have been proposed in [11]. A refinement relation and a logical characterization of refinement were also given in [11]. For a set of actions A, a modal transition system (MTS) is a triple $(P, \longrightarrow^{\Box}, \longrightarrow^{\Diamond})$, where P is a set of states and $\longrightarrow^{\Box}, \longrightarrow^{\Diamond} \subseteq P \times A \times P$ are transition relations such that $\longrightarrow^{\Box} \subseteq \longrightarrow^{\Diamond}$. The transitions in $\longrightarrow^{\Box}$ are called the must transitions and those in $\longrightarrow^{\Diamond}$ are the may transitions. In an MTS, each must transition is also a may transition, which intuitively means that any required transition is also allowed.

An interface automaton (IA) [1], introduced by de Alfaro and Henzinger, is an automata-based model suitable for specifying component-based systems. An IA consists of states, initial states, internal actions, input actions, output actions and a transition relation. The composition and refinement of two IAs are proposed in [1].

Z was introduced in the early 80's in Oxford by Abrial as a set-theoretic and predicate language for the specification of data structure, state spaces and state transformations. A boxed notation called schemas is used for structuring Z specifications. Z makes use of identifier decorations to encode intended interpretations. A state variable with no decoration represents the current (before) state and a state variable ending with a prime ($'$) represents the next (after) state. A variable ending with a question mark (?) represents an input and a variable ending with an exclamation mark (!) represents an output. In Z, there are many schema operators. For example, we write $S \wedge T$ to denote the conjunction of these two schemas: a new schema formed by merging the declaration parts of S and T and conjoining their predicate parts. $S \Rightarrow T$ ($S \Leftrightarrow T$) is similar to $S \wedge T$ except connecting their predicate parts by $\Rightarrow$ ($\Leftrightarrow$). The hiding operation $S \backslash (x_1, ..., x_n)$ removes from the schema S the components $x_1, ..., x_n$ explicitly listed, which must exist. The hiding operation $S \backslash (x_1, ..., x_n)$ removes from the schema S the components $x_1, ..., x_n$ explicitly listed, which must exist. Formally, $S \backslash (x_1, ..., x_n)$ is equivalent to $(\exists\, x_1 : t_1;\ ...;\ x_n : t_n \bullet S)$, where $x_1, ..., x_n$ have types $t_1, ..., t_n$ in S. The notation $\exists\, x : a \bullet S$ states that there is some object x in a for which S is true. The notation $\forall\, x : a \bullet S$ states that for each object x in a, S is true. For the sake of space, more details of Z can be refereed to some books on Z [15].

III. Modal Interface Automata with Z Notation

This paper is based on modal ZIA, a specification language which integrates modal transition systems, interface automata and Z. Modal ZIA is defined such that apart from enabling one to deal with the modal properties, behavioral properties and the data properties of a system independently. In this section, we combine modal transition systems, inference automata and Z language to give a specification approach for software components. We first give the definition of such model. Then we define the modal refinement of modal ZIA. Furthermore, we propose and prove some properties on the modal refinement of modal ZIA.

A. Model of Modal ZIA

Interface automata provide a specification approach for interface behavior properties. But this approach can not describe data structures specification of states. On the other hand, Z can specify the state of a system, but is not suitable to behavioral properties. This section describes modal ZIA as a conservative extension of both interface automata and Z in the sense that almost all syntactical and semantical aspects of interface automata and Z are preserved.

In the original interface automata, states and transitions are abstract atomic symbols. But in modal ZIA, states and transitions are described by Z schemas.

In the rest of this paper, we use the following terminology:

(1) A state schema is a schema which does not contain any variable with decoration $'$.

(2) An input operation schema is an operation schema which contains input variables.

(3) An output operation schema is an operation schema which contains output variables.

(4) An internal operation schema is an operation schema which contains variables with decoration $'$.

Intuitively, a state schema is assigned to a state which may contain many variables, this state schema describes the constraint of variables in the state. A variable with decoration $'$ denotes a variable at next state. So a state schema does not contain any variable with decoration $'$. An input (output) operation schema is assigned to an input (output) action which may contain many input (output) variables, this input (output) operation schema describes the constraint of variables in the input (output) action. So an input (output) operation schema is an operation schema which contains input (output) variables. An internal operation schema is assigned to an internal action, this internal operation schema describes the change of variables after performing the internal action. So an internal operation schema is an operation schema which contains variables with decoration $'$.

In the rest of paper, given an assignment ρ and a schema A, we write $\rho \models A$ if ρ assigns every variable x in the declaration part of A to an element of its type set, which satisfies the predicate part of A; we write $\models A$ if $\rho \models A$ for any assignment ρ.

Let S to be a Z schema, we use $V^I(S)$ ($V^O(S)$, $V^H(S)$) to denote the set of input variables (output variables, internal variables) in S.

Definition 1. A modal interface automaton with Z notation (modal ZIA) $P = \langle S_P, S_P^i, A_P^I, A_P^O, A_P^H, V_P^I, V_P^O, V_P^H, F_P^S, F_P^A, G_P^{IA}, G_P^{OA}, A_P^{\Box}, A_P^{\Diamond}, V_P^{\Box}, V_P^{\Diamond}, T_P \rangle$ consists of the following elements:

(1) S_P is a set of states.

(2) $S_P^i \subseteq S_P$ is a set of initial states. If $S_P^i = \emptyset$ then P is called empty.

(3) A_P^I, A_P^O and A_P^H are disjoint sets of input, output, and internal actions, respectively. We denote by $A_P = A_P^I \cup A_P^O \cup A_P^H$

the set of all actions.

(4) V_P^I, V_P^O and V_P^H are disjoint sets of input, output, and internal variables, respectively. We denote by $V_P = V_P^I \cup V_P^O \cup V_P^H$ the set of all variables.

(5) F_P^S is a map, which maps any state in S_P to a state schema in Z language. Intuitively, for any state s, $F_P^S(s)$ specifies the data structure properties of all the variables in the state s.

(6) F_P^A is a map, which maps any input action in A_P^I to an input operation schema in Z language, and maps any output action in A_P^O to an output operation schema in Z language, and maps any internal action in A_P^H to an internal operation schema in Z language. Intuitively, for any action a, $F_P^A(a)$ specifies the data structure properties of all the variables before and after performing action a.

(7) G_P^{IA} is a map, which maps any input action in A_P^I to a set of input variables. Intuitively, an input action a inputs all the input variables in $G_P^{IA}(a)$. For any input action a, $G_P^{IA}(a) \subseteq V^I(F_P^A(a))$.

(8) G_P^{OA} is a map, which maps any output action in A_P^I to a set of output variables. Intuitively, an output action a outputs all the output variables in $G_P^{OA}(a)$. For any output action a, $G_P^{OA}(a) \subseteq V^O(F_P^A(a))$.

(9) $A_P^{\Box}$, $A_P^{\Diamond} \subseteq A_P^I \cup A_P^O$, where $A_P^{\Box}$ is the set of must actions, and $A_P^{\Diamond}$ is the set of may actions.

(10) $V_P^{\Box}$, $V_P^{\Diamond} \subseteq V_P^I \cup V_P^O$, where $V_P^{\Box}$ is the set of must variables, and $V_P^{\Diamond}$ is the set of may variables.

(11) T_P is the set of transitions between states, $T_P \subseteq S_P \times A_P \times S_P$. If $(s, a, t) \in T_P$ then $((F_P^S(s) \wedge F_P^A(a)) \backslash (x_1, ..., x_m) \Leftrightarrow F_P^S(t)[y_1'/y_1, ..., y_n'/y_n])$ is a tautology, where $\{x_1, ..., x_m\}$ is the set of the variables in $F_P^S(s)$, $\{y_1, ..., y_n\}$ is the set of the variables in $F_P^S(t)$, the set of variables in $F_P^A(a)$ is the subset of $\{x_1, ..., x_m\} \cup \{y_1', ..., y_n'\}$.

An action $a \in A_P$ is enabled at a state $s \in V_P$ if there is a step $(s, a, s') \in T_P$ for some $s \in S_P$. We indicate by $A_P^I(s)$, $A_P^O(s)$, $A_P^H(s)$ the subsets of input, output and internal actions that are enabled at the state s and we let $A_P(s) = A_P^I(s) \cup A_P^O(s) \cup A_P^H(s)$.

In the following, we call a a must (may) action if $a \in A_P^{\Box}$ ($a \in A_P^{\Diamond}$), and call x a must (may) action if $a \in V_P^{\Box}$ ($a \in V_P^{\Diamond}$). A must action (variable) can be regarded as a necessary action (variable), i.e., an action (a variable) which must be included in the implementation, and a may action (variable) can be regarded as a possible action (variable), i.e., an action (a variable) which may be or may not be included in the implementation. Another usefulness of must actions (variables) and may actions (variables) is in the abstraction of systems. In general, abstraction in model checking falls into three types, depending on the approximation relations between concrete and abstract models and property preservation relations for temporal properties. One type is abstraction methods that support both verification and refutation of program properties in the same framework, which we refer to as exact-approximation. In the over-approximation abstraction framework [6], an abstract model contains more behaviors than the original program. The dual of this framework is under-approximation [13]. In this case, an abstract model contains less behaviors than the original one. In the following definition of modal refinement relation of modal ZIAs, must actions (variables) provide the over-approximation method of actions (variables), and may actions (variables) provide the under-approximation method of actions (variables).

B. Modal Refinement Relation

The modal refinement relation aims at formalizing the relation between abstract and concrete versions of the same component, for example, between a specification and its implementation.

Roughly, a modal ZIA P refines a modal ZIA Q if all the must actions of P can be simulated by Q and all the may actions of Q can be simulated by P. To define this concept, we need some preliminary notions.

In the following, we use $V^{\Diamond}(A)$ to denote the set of may variables in Z schema A, $V^{\Box}(A)$ to denote the set of must variables in Z schema A, and $V^{\otimes}(A)$ to denote the set of other variables in Z schema A.

In order to define the modal refinement relation between Z schemas, we need the following notation.

Definition 2. Consider two Z schemas A and B with $V^{\Diamond}(A) = V^{\Diamond}(B)$, $V^{\Box}(A) = V^{\Box}(B)$ and $V^{\otimes}(A) = V^{\otimes}(B) = \emptyset$. We use the notation $A \geq B$ if one of the following cases holds:

(1) If $V^{\Diamond}(A) \neq \emptyset$ and $V^{\Box}(A) \neq \emptyset$ then given an assignment ρ on $V^{\Diamond}(A)$, for any assignment σ on $V^{\Box}(A)$, $\rho \cup \sigma \models A$ implies $\rho \cup \sigma \models B$, and given an assignment σ on $V^{\Box}(A)$, for any assignment ρ on $V^{\Diamond}(A)$, $\rho \cup \sigma \models B$ implies $\rho \cup \sigma \models A$, where $\rho \models A$ means that A is true under assignment ρ, $\rho \cup \sigma$ is the union of ρ and σ.

(2) If $V^{\Diamond}(A) \neq \emptyset$ and $V^{\Box}(A) = \emptyset$ then for any assignment ρ on $V^{\Diamond}(A)$, $\rho \models B$ implies $\rho \models A$.

(3) If $V^{\Diamond}(A) = \emptyset$ and $V^{\Box}(A) \neq \emptyset$ then for any assignment ρ on $V^{\Box}(A)$, $\rho \models A$ implies $\rho \models B$.

(4) $V^{\Diamond}(A) = \emptyset$ and $V^{\Box}(A) = \emptyset$.

Intuitively, $A \geq B$ means that schemas A and B have the same may variables and the same must variables, and schema B has bigger domains of must variables but smaller ranges of may variables than schema A. This means that must variables can be regarded as the over-approximation of variables, and may variables can be regarded as the under-approximation of variables.

For example, $A \widehat{=} [x^{\Diamond} : R;\; y^{\Box} : N \mid y^{\Box} = 2\lfloor x^{\Diamond} \rfloor] \geq B \widehat{=} [x^{\Diamond} : N;\; y^{\Box} : R \mid y^{\Box} = 2x^{\Diamond}]$, where $x^{\Diamond}$ is a may variable, $y^{\Box}$ is a must variable, N is the set of natural numbers, R is the set of real numbers, and $\lfloor x^{\Diamond} \rfloor$ is the largest natural number that is not larger than $x^{\Diamond}$.

Now we give the modal refinement relation between Z schemas, which describe the modal refinement relation between data structures properties of states. Roughly speaking, for two Z schemas A and B, we say that B refines A if the may variables and the must variables in A are also in B, and schema B has bigger domains of these may variables but smaller ranges of these must variables than schema A.

Definition 3. Consider two Z schemas A and B, we use the notation $A \trianglerighteq B$ if

(1) $V^{\Box}(A) \subseteq V^{\Box}(B)$, $V^{\Diamond}(A) \subseteq V^{\Diamond}(B)$.

(2) $A \backslash (x_1, ..., x_m) \geq B \backslash (y_1, ..., y_n)$, where $\{x_1, ..., x_m\} = V(A) - V^{\Box}(A) - V^{\Diamond}(A)$, $\{y_1, ..., y_n\} = V(B) - V^{\Box}(A) - V^{\Diamond}(A)$.

For example, $A \widehat{=} [x^{\Diamond} : R;\ y^{\Box} : N \mid y^{\Box} = 2\lfloor x^{\Diamond} \rfloor] \trianglerighteq B \widehat{=} [x^{\Diamond} : N;\ u^{\Diamond} : R;\ y^{\Box} : R;\ v^{\Box} : R;\ z : N \mid y^{\Box} = 2x^{\Diamond};\ v^{\Box} = z * u^{\Diamond}]$, where $x^{\Diamond}$, $u^{\Diamond}$ are may variables, $y^{\Box}$, $v^{\Box}$ are must variables.

So intuitively, $A \geq B$ describes the modal refinement of data properties of schemas A and B if schemas A and B have the same may and must variables. $A \trianglerighteq B$ describes the modal refinement of data properties of schemas A and B in the general case, i.e., schemas A and B may have different may and must variables.

The precise definition of modal refinement must take into account the fact that the internal actions of P and Q are independent. For this, we need some following preliminary notions.

We now give the following definition which describes the set of states after performing a sequence of internal actions from a given state.

Definition 4. Given a modal ZIA P and a state $s \in S_P$, the set $\varepsilon - closure_P(s)$ is the smallest set $U \subseteq S_P$ such that (1) $s \in U$ and (2) if $t \in U$ and $(t, a, t^*) \in T_P^H$ then $t^* \in U$.

The environment of a modal ZIA P cannot see the internal actions of P. Consequently if P is at a state s then the environment cannot distinguish between s and any state in $\varepsilon - closure_P(s)$.

The following definition describes the set of states after performing several internal actions and an external action from a given state.

Definition 5. Consider a modal ZIA P and a state $s \in S_P$. For an action a, we let

$ExtDest_P(s, a) = \{s^* \mid \exists (t, a, t^*) \in T_P . t \in \varepsilon - closure_P(s)$ and $s^* \in \varepsilon - closure_P(t^*)\}$.

In the following, we give a modal refinement relation between modal ZIAs. For modal ZIAs, a state has not only behavioral properties but also data properties. Therefore this modal refinement relation involves both the modal refinement relation between behavioral properties and the modal refinement relation between data properties.

Definition 6. Consider two modal ZIAs P and Q. A binary relation $\succeq_m \subseteq S_P \times S_Q$ is a modal refinement from Q to P if for all states $s \in S_P$, there exists $t \in S_Q$ such that $s \succeq_m t$ the following conditions hold:

(1) $F_P^S(s) \trianglerighteq F_Q^S(t)$.

(2) For any action $a \in A_P^{\Box}$, if $s^* \in ExtDest_P(s, a)$, then there is a state $t^* \in ExtDest_Q(t, a)$ such that $F_P^S(s^*) \trianglerighteq F_Q^S(t^*)$ and $s^* \succeq_m t^*$.

(3) For any action $a \in A_Q^{\Diamond}$, if $t^* \in ExtDest_Q(t, a)$, then there is a state $s^* \in ExtDest_Q(s, a)$ such that $F_P^S(s^*) \trianglerighteq F_Q^S(t^*)$ and $s^* \succeq_m t^*$.

Intuitively, must actions (variables) represents the over-approximation of actions (variables), and may actions (variables) represents the under-approximation of actions (variables). The modal refinement relation describes the both over-approximation and under-approximation for transitions and variables for modal ZIAs.

We say that modal ZIA P is refined by modal ZIA Q if for some initial states s in P and t in Q, s is refined by t.

Definition 7. The modal ZIA Q refines the modal ZIA P written $P \succeq_m Q$ if:

there is a modal refinement $\succeq_m$ from Q to P, a state $s \in S_P^i$ and a state $t \in S_Q^i$ such that $s \succeq_m t$.

The above definitions of modal refinement relations can be extended to the definitions of bisimulation relations by adding the symmetric condition of relations.

The following lemma states that $\trianglerighteq$ is a partial order (i.e., reflexive and transitive).

Lemma 1. (1) $A \trianglerighteq A$.

(2) If $A \trianglerighteq B$ and $B \trianglerighteq C$, then $A \trianglerighteq C$.

The following proposition means that $\succeq_m$ is a partial order.

Proposition 1. (1) $P \succeq_m P$.

(2) If $P \succeq_m Q$ and $Q \succeq_m R$, then $P \succeq_m R$.

IV. A Logic for Modal ZIAs

Providing a logical characterization for various refinement/equivalence relations has been one of the major research topics in the development of automata theory and process theories. A logical characterization not only allows us to reason about behaviors of systems, but also helps to verify the properties of systems. For modal transition system, a logic for modal refinement relation was proposed and a logical characterization was given in [3]. For convariant-contravariant simulation of labelled transition systems, a logical characterization for convariant-contravariant simulation relation was given in [8]. In this section, we give the similar result for modal ZIAs. In the following, we give a logic for modal ZIAs named *MZIAL* and give a logical characterization of modal refinement relation $\succeq_m$.

A. Syntax of MZIAL

Throughout this paper, we let *MZIAL* be a language which is just the set of formulas of interest to us.

Definition 8. The set of formulas called *MZIAL*, is given by the following rules:

(1) $\top \in MZIAL$.

(2) $\bot \in MZIAL$.

(3) If φ is in the form of $p(x_1^{\odot}, ..., x_n^{\odot})$, then $\varphi \in MZIAL$, where $x_1^{\odot}, ..., x_n^{\odot}$ are may or must variables, i.e., $x_i^{\odot}$ is in the form of $x_i^{\Diamond}$ or $x_i^{\Box}$, p is a $n - ary$ prediction.

(4) If $\varphi_i \in MZIAL$ for any $i \in I$, then $\wedge_{i \in I} \varphi_i \in MZIAL$.

(5) If $\varphi_i \in MZIAL$ for any $i \in I$, then $\vee_{i \in I} \varphi_i \in MZIAL$.

(6) If $\varphi \in MZIAL$, then $(\forall x^{\Diamond})\varphi \in MZIAL$.

(7) If $\varphi \in MZIAL$, then $(\exists x^{\Box})\varphi \in MZIAL$.

(8) If $\varphi \in MZIAL$, then $[[a^{\Diamond}]]\varphi \in MZIAL$.

(9) If $\varphi \in MZIAL$, then $\langle\langle a^{\Box} \rangle\rangle\varphi \in MZIAL$.

B. Semantics of MZIAL

We will describe the semantics of *MZIAL*, that is, whether a given formula is true or false.

The satisfaction relation $\models$ is given recursively by the following definition, where $(P, s) \models \varphi$ means that state s of modal ZIA P satisfies formula φ.

Definition 9. Semantics of *MZIAL*

(1) $(P,s) \models \top$.

(2) $(P,s) \not\models \bot$.

(3) $(P,s) \models p(x_1^{\odot},...,x_n^{\odot})$ iff $p(x_1^{\odot},...,x_n^{\odot}) \trianglerighteq F_P^S(s)$.

(4) $(P,s) \models \wedge_{i\in I}\varphi_i$ iff $(P,s) \models \varphi_i$ for any $i \in I$.

(5) $(P,s) \models \vee_{i\in I}\varphi_i$ iff $(P,s) \models \varphi_i$ for some $i \in I$.

(6) $(P,s) \models (\forall x^{\Diamond})\varphi$ iff $(P,s) \models \varphi\{v/x^{\Diamond}\}$ for any $v \in T$, where $x^{\Diamond} \in V_P^{\Diamond}$, T is the type of $x^{\Diamond}$.

(7) $(P,s) \models (\exists x^{\Box})\varphi$ iff $(P,s) \models \varphi\{v/x^{\Box}\}$ for some $v \in T$, where $x^{\Box} \in V_P^{\Box}$, T is the type of $x^{\Box}$.

(8) $(P,s) \models [[a^{\Diamond}]]\varphi$ iff $a^{\Diamond} \in A_P^{\Diamond}$, and for any state $t \in$ $ExtDest_P(s,a^{\Diamond})$, $(P,t) \models \varphi$.

(9) $(P,s) \models \langle\langle a^{\Box}\rangle\rangle\varphi$ iff $a^{\Box} \in A_P^{\Box}$, and there exists a state t, such that $t \in ExtDest_P(s,a^{\Box})$ and $(P,t) \models \varphi$.

C. Logical Characterization of Modal Refinement Relation

Now we give the definition of logical refinement relation. Intuitively, for a state s in modal ZIA P, and a state t in modal ZIA Q, we say t refines s if the collection of *MZIAL* formulas satisfied by s is included in the collection of *MZIAL* formulas satisfied by t.

Definition 10. $(P,s) \succeq_l (Q,t)$ iff $(P,s) \models \varphi \Rightarrow (Q,t) \models \varphi$ for any φ in *MZIAL*.

In the following, we give the logical characterization of modal refinement relation which means that $\succeq_m$ is equivalent to $\succeq_l$.

Proposition 2. $(P,s) \succeq_m (Q,t)$ iff $(P,s) \succeq_l (Q,t)$.

V. Model Checking muZIAL for Finite Modal ZIAs

In this section, we will study the model checking problem for modal ZIA. But in *MZIAL*, there exist infinite length formulas since there are indexed disjunction $\vee_{i\in I}$ and indexed conjunction $\wedge_{i\in I}$. Hence *MZIAL* itself is not suitable for model checking. In this section we present a logic named *muZIAL*. It is well known that μ operator and ν operator can be rewritten into indexed disjunction $\vee_{i\in I}$ and indexed conjunction $\wedge_{i\in I}$, therefore *muZIAL* is a sublogic of *MZIAL*. Then we give an algorithm model checking *muZIAL* for finite modal ZIAs, i.e., modal ZIAs with finite state space and finite domains for all variables. The model checking problem for modal ZIAs asks, given a state s of a modal ZIA P and a *muZIAL* formula φ, whether $(P,s) \models \varphi$.

A. Syntax of muZIAL

Throughout this paper, we let *muZIAL* be a language which is just the set of formulas of interest to us.

Definition 11. The set of formulas called *muZIAL*, is given by the following rules:

(1) $\top \in muZIAL$.

(2) $\bot \in muZIAL$.

(3) If φ is in the form of $p(x_1^{\odot},...,x_n^{\odot})$, then $\varphi \in muZIAL$, where $x_1^{\odot},...,x_n^{\odot}$ are may or must variables, p is a $n-ary$ prediction.

(4) If $X \in$proposition variables set PV, then $X \in muZIAL$.

(5) If φ_1, $\varphi_2 \in muZIAL$, then $\varphi_1 \wedge \varphi_2 \in muZIAL$.

(6) If φ_1, $\varphi_2 \in muZIAL$, then $\varphi_1 \vee \varphi_2 \in muZIAL$.

(7) If $\varphi \in muZIAL$, then $(\forall x^{\Diamond})\varphi \in muZIAL$.

(8) If $\varphi \in muZIAL$, then $(\exists x^{\Box})\varphi \in muZIAL$.

(9) If $\varphi \in muZIAL$, then $[[a^{\Diamond}]]\varphi \in muZIAL$.

(10) If $\varphi \in muZIAL$, then $\langle\langle a^{\Box}\rangle\rangle\varphi \in muZIAL$.

(11) If $\varphi(X) \in muZIAL$, then $\nu X.\varphi(X) \in muZIAL$.

(12) If $\varphi(X) \in muZIAL$, then $\mu X.\varphi(X) \in muZIAL$.

It is well known that $\nu X.\varphi(X)$ $(\mu X.\varphi(X))$ can be rewritten into an "unfolded" form $\varphi[\nu X.\varphi(X)/X]$ $(\varphi[\mu X.\varphi(X)/X])$, and this "unfolding" proceeding can be continued. Intuitively, $\nu X.\varphi(X)$ $(\mu X.\varphi(X))$ can be "unfolded" to a formula in *MZIAL* by using indexed disjunction $\vee_{i\in I}$ (indexed conjunction $\wedge_{i\in I}$). Thus *muZIAL* can be regarded as a sublogic of *MZIAL*.

B. Semantics of muZIAL

We will describe the semantics of *muZIAL*, that is, whether a given formula is true or false.

Formally, a formula φ is interpreted as a set of states in which φ is true. We write such set of states as $[[\varphi]]_P^e$, where P is a modal ZIA and e: $V \rightarrow 2^S$ is an environment. We denote by $e[X \leftarrow W]$ a new environment that is the same as e except that $e[X \leftarrow W](X) = W$. The set $[[\varphi]]_P^e$ is defined recursively as follows:

Definition 12. Semantics of *muZIAL*

(1) $|| \top ||_P^e = S_P$, where S_P is the set of states in modal ZIA P.

(2) $|| \bot ||_P^e = \emptyset$.

(3) $|| p(x_1^{\odot},...,x_n^{\odot}) ||_P^e = \{s \mid p(x_1^{\odot},...,x_n^{\odot}) \trianglerighteq F_P^S(s)\}$.

(4) $|| X ||_P^e = e(X)$.

(5) $|| \varphi_1 \wedge \varphi_2 ||_P^e = || \varphi_1 ||_P^e \cap || \varphi_2 ||_P^e$.

(6) $|| \varphi_1 \vee \varphi_2 ||_P^e = || \varphi_1 ||_P^e \cup || \varphi_2 ||_P^e$.

(7) $|| (\forall x^{\Diamond})\varphi ||_P^e = \cap_{v\in T} || \varphi\{v/x^{\Diamond}\} ||_P^e$, where $x^{\Diamond} \in V_P^{\Diamond}$, T is the type of $x^{\Diamond}$.

(8) $|| (\exists x^{\Box})\varphi ||_P^e = \cup_{v\in T} || \varphi\{v/x^{\Box}\} ||_P^e$, where $x^{\Box} \in V_P^{\Box}$, T is the type of $x^{\Box}$.

(9) $|| \ [[a^{\Diamond}]]\varphi \ ||_P^e = \{s \mid$ for any state t, such that $t \in$ $ExtDest_P(s,a^{\Diamond})$ and $t \in || \varphi ||_P^e\}$.

(10) $|| \ \langle\langle a\rangle\rangle\varphi \ ||_P^e = \{s \mid$ there exists a state t, such that $t \in ExtDest_P(s,a^{\Box})$ and $t \in || \varphi ||_P^e\}$.

(11) $|| \nu X.\varphi(X) ||_P^e = \cup\{W \subseteq S_P \mid W \subseteq || \varphi(X) ||_P^{e[X\leftarrow W]}\}$.

(12) $|| \mu X.\varphi(X) ||_P^e = \cap\{W \subseteq S_P \mid || \varphi(X) ||_P^{e[X\leftarrow W]} \subseteq W\}$.

Given a modal ZIA P, we say that φ is valid in P, and write $P \models_{muZIAL} \varphi$, if $s \in [[\varphi]]_P^e$ for every state s in P, and we say that φ is satisfiable in P, and write $P, s \models_{muZIAL} \varphi$, if $s \in [[\varphi]]_P^e$ for some s in P. We say that φ is valid, and write $\models_{muZIAL} \varphi$, if φ is valid in all modal ZIAs, and that φ is satisfiable if it is satisfiable in some modal ZIA. We write $\Gamma \models_{muZIAL} \varphi$, if φ is valid in all modal ZIAs in which Γ is satisfiable.

C. Model Checking Algorithm

In this section, we give a model checking algorithm over modal ZIAs with finite domain. We denote the set $|| \varphi ||_P^e$ by $Eval(\varphi,e)$, where e is an environment which maps each prediction variable to a closed formula. The function *Sub*, when given a formula φ, returns a queue of syntactic subformulas of φ such that if φ_1 is a subformula of φ and φ_2 is a subformula of φ_1, then φ_2 precedes φ_1 in the queue $Sub(\varphi)$.

Suppose $P = \langle S_P, S_P^i, A_P^I, A_P^O, A_P^H, V_P^I, V_P^O, V_P^H, F_P^S, F_P^A, T_P\rangle$ is a ZIA with finite domain, s is a state of P, φ is a formula of $\mu ZIAL$, the algorithm to verify $s \models \varphi$ is given as follows:

```
For each $\varphi'$ in $Sub(\varphi)$ do
    case $\varphi' = \top : Eval(\varphi', e) := S_P$;
    case $\varphi' = \bot : Eval(\varphi', e) := \emptyset$;
    case $\varphi' = p(x_1^{\odot}, ..., x_n^{\odot}) : Eval(\varphi', e) := \{s \mid p(x_1^{\odot}, ..., x_n^{\odot}) \trianglerighteq F_P^S(s)\}$, where Z schema $F_P^S(s)$ is regarded as a first order logical formula;
    case $\varphi' = X : Eval(\varphi', e) := e(X)$;
    case $\varphi' = \theta_1 \wedge \theta_2 : Eval(\varphi', e) := Eval(\theta_1, e) \cap Eval(\theta_2, e)$;
    case $\varphi' = \theta_1 \vee \theta_2 : Eval(\varphi', e) := Eval(\theta_1, e) \cup Eval(\theta_2, e)$;
    case $\varphi' = (\forall x^{\Diamond})\theta : Eval(\varphi', e) := \cap_{v \in T} Eval(\theta\{v/x^{\Diamond}\}, e)$, where $T$ is the type of $x^{\Diamond}$;
    case $\varphi' = (\exists x^{\Box})\theta : Eval(\varphi', e) := \cup_{v \in T} Eval(\theta\{v/x^{\Box}\}, e)$, where $T$ is the type of $x^{\Box}$;
    case $\varphi' = [[a^{\Diamond}]]\theta : Eval(\varphi', e) := \{s \mid$ for any $t$, $t \in ExtDest_P(s, a^{\Diamond})$ implies $t \in Eval(\theta, e)\}$, where $ExtDest_P(s, a^{\Diamond})$ is computable since the state space $S_P$ is finite;
    case $\varphi' = \langle\langle a^{\Box}\rangle\rangle\theta : Eval(\varphi', e) := \{s \mid$ for some $t$, $t \in ExtDest_P(s, a^{\Box})$ and $t \in Eval(\theta, e)\}$, where $ExtDest_P(s, a^{\Box})$ is computable since the state space $S_P$ is finite;
    case $\varphi' = \nu X.\theta$ :
        $S_{val} := S_P$
        repeat
            $S_{old} := S_{val}$
            $S_{val} := Eval(\theta, e[X \leftarrow S_{val}])$
        until   $S_{val} = S_{old}$
        $Eval(\varphi', e) := S_{val}$
    end case
    case $\varphi' = \mu X.\theta$ :
        $S_{val} := \emptyset$
        repeat
            $S_{old} := S_{val}$
            $S_{val} := Eval(\theta, e[X \leftarrow S_{val}])$
        until   $S_{val} = S_{old}$
        $Eval(\varphi', e) := S_{val}$
    end case
return $Eval(\varphi, e)$
```

Partial correctness of the model checking algorithm can be proved induction on the structure of the input formula φ. Termination is guaranteed, because the state space S_P is finite. Therefore we have the following proposition:

Proposition 3. The model checking algorithm given in the above terminates and is correct, i.e., it returns the set of states in which the input formula is satisfied.

VI. Conclusions

This paper proposed a specification approach of software components which is suitable to specify the modal behavior and data structures properties of a system. There are several other works for such topic. Some examples are LOTOS and Z [7], CSP-Z [9], CSP-OZ [10], Circus [16], and modal interface [14]. In this paper, we define a combination of modal transition systems, interface automata and Z called modal ZIA, which can be applied to specify modal properties, behavioral properties and data structures properties of a system. We also define the modal refinement relation for modal ZIA. The properties of the modal refinement relation for modal ZIA are also studied. Modal ZIA is well suited for specification and development of software components. It provides powerful techniques to describe modal properties, data structures and control aspects in a common framework. Furthermore, we present a logic *MZIAL* for modal ZIA and give a logical characterization of modal refinement relation. Finally, we present a sublogic of *MZIAL*, named *muZIAL*, and give a model checking algorithm for finite modal ZIA.

Acknowledgment

This work was supported by the National Natural Science Foundation of China under Grant No. 60873025, the Foundation of Provincial Key Laboratory for Computer Information Processing Technology of Soochow University under Grant No. KJS0920, the Natural Science Foundation of Jiangsu Province of China under Grant No. BK2008389, and the Aviation Science Fund of China under Grant No. 20085552023.

References

[1] Luca de Alfaro, Thomas A Henzinger. Interface Automata. In the Proceedings of the 9th Annual ACM Symposium on Foundations of Software Engineering (FSE), 2001.

[2] T. Bolognesi and E. Brinksma. Introduction to the ISO specification language LOTOS. Computer Networks and ISDN Systems, 14(1): 25-59, 1987.

[3] G. Boudol and K. G. Larsen. Graphical versus logical specifications. Theoretical Computer Science, 106(1):3-20, 1992.

[4] Z. Cao. Refinement Checking for Interface Automata with Z Notation. In Proceeding of Software Engineering and Knowledge Engineering 2010. 399-404. 2010.

[5] Z. Cao and H. Wang: Extending Interface Automata with Z Notation. In FSEN 2011, 359-367.

[6] E. M. Clarke, O. Grumberg, and D. E. Long. Model Checking and Abstraction. ACM Transactions on Programming Languages and Systems, 16(5):1512-1542, 1994.

[7] J. Derrick, E. Boiten, H. Bowman and M. Steen. Viewpoint consistency in Z and LOTOS: A case study. In FME'97, 644-664. Springer-Verlag, 1997.

[8] I. Fabregas, D. de Frutos-Escrig, and M. Palomino. Logics for contravariant simulations. In Proceedings of FORTE 2010, lncs 6117, 224-231, 2010.

[9] C. Fischer. Combining CSP and Z. Technical report, TRCF-97-1, University of Oldenburg, 1996.

[10] C. Fischer. CSP-OZ: A combination of Object-Z and CSP. In FMODDS'97, Chapman Hall, 1997.

[11] K. G. Larsen. Modal specifications. In Automatic Verification Methods for Finite State Systems, 232-246. Springer, 1989.

[12] Z. Manna and A. Pnueli. The Temporal Logic of Reactive and Concurrent Systems. Springer-Verlag, 1991.

[13] C. Pasareanu, R. Pelanek, and W. Visser. Concrete Model Checking with Abstract Matching and Refinement. In Proceedings of CAV05, 52-66. Springer, 2005.

[14] J.-B. Raclet, E. Badouel, A. Benveniste, B. Caillaud, A. Legay, and R. Passerone. A Modal Interface Theory for Component-based Design. Fundamenta Informaticae, 2010 1-31.

[15] J. M. Spivey. The Z Notation: A Reference Manual. Sencond Edition. Prentice Hall International (UK) Ltd. 1998.

[16] J. C. P. Woodcock and A. L. C. Cavalcanti. The semantics of circus. In ZB 2002: Formal Specification and Development in Z and B, 184-203. Springer-Verlag, 2002.

Towards Autonomic Business Process Models

Karolyne Oliveira, Jaelson Castro
Centro de Informática
Universidade Federal de Pernambuco
Recife, Brazil
{kmao, jbc}@cin.ufpe.br

Sergio España, Oscar Pastor
Centro de Investigación en Métodos de Producción de Software
Universitat Politècnica de València
Valencia, Spain
{sergio.espana, opastor}@pros.upv.es

***Abstract* - In many organizations business process models (BPM) play a central role by capturing the way activities are performed. However, in the dynamic circumstances often encountered in today's business world, the processes are becoming increasingly complex and heterogeneous. Unfortunately, most of the current approaches to BPM are too inflexible and unresponsive to change. This calls for Autonomic Business Process (ABP) aimed at self-management and self-adaptation in dynamic environments. However, a key challenge is to keep the models understandable and scalable in increasingly complex scenarios. In our work we rely on the principle of separation of concerns and modularization in order to properly represent autonomic features in Business Process Models. We argue that Communication Analysis can help to indicate mission-critical activities that must be treated in autonomic manner. We outline a process that helps to define the granularity of the Business Process Models, indicating where the system needs to be instrumented. This novel approach provides four well-defined levels of abstraction: Communicational Level, Technological Level, Operational Level and Service Level. A real example is used to illustrate our proposal.**

Keywords - Business Process Modeling; Autonomic Computing; Self-Adaptation; Communication Analysis

I. INTRODUCTION

Analysts and researchers in Information Technology and Communication (ITC) management have forecasted an unavoidable collapse, imposed by the sheer scale and complexity of the new systems. Hence, complexity and size have driven ITC management to consider less human-dependent solutions, such as Autonomic Computing (AC), aimed at developing computer systems capable of self-management [1].

An autonomic system must know itself, be able to self-configure and reconfigure when faced with unforeseen circumstances. Moreover, it should look for optimizations, to be able to self-heal as well as to self-recover from critical failures and protect itself. Hence, it must understand the environment and the context around the activities.

In our work, we are concerned about the use of principles of autonomic computing in the management of business processes. Our goal is to help organizations to survive in dynamics environments. More specifically we want to face an open challenge which is to promote modularization and separation of concerns (SoC) in Autonomic Business Process Models [2].

The paper proposes a process that exploits the high variability in service-oriented system environment by the use of contexts and autonomic adaptations by operationalizations of Non-Functional Requirements (NFR). In order to improve modularity and promote the use of different levels of abstractions in Autonomic Business Process we investigate the use of Communication Analysis Principles. The benefits are manifold and include addressing scalability problems and improving the understandability of ABP in complex scenarios [3].

A MAPE cycle (Monitor-Analyze-Plan-Execute), was utilized considering both aspects: system (self) and the instrumented BPM (context).

In order to illustrate our proposal we describe a real example which presents mission-critical activities often managed through BPM.

In the next sections, we present concepts related to this work, our approach and conclusions respectively.

II. BACKGROUND AND RELATED WORKS

In this section, we introduce self-adaptation and the use of business process modeling as well some relevant work on using these models to provide autonomic computing characteristics to software.

A. Autonomic Computing / Self-Adaptation

Self-adaptive software is expected to fulfill its requirements at run-time in response to changes. This objective can be achieved through monitoring the software system (self) and its environment (context) to detect changes, make appropriate decisions, and act accordingly. Certain characteristics, known as self-* properties are expected: (i) Self-configuration; (ii) Self-healing; (iii) Self-optimizing; and, (iv) Self-Protecting [4], [5]. Figure 1 shows the hierarchy of the self-* properties according [6].

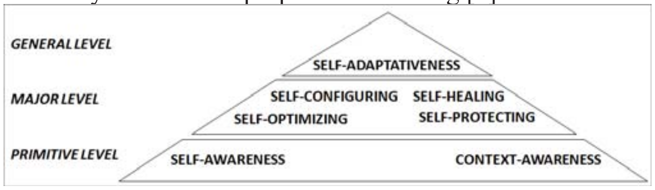


Figure 1. Hierarchy of the Self-* Properties, adapted from [6]

Adaptive software should basically determine **when** and **where** a change is required, **how** it can be changed, and **what** the impacts of a change are on system attributes. These dimensions (when, where, what, and how) are highlighted in the taxonomy of evolution [7]. *When* corresponds to temporal properties; *Where* refers to the object of change and describe which artifact needs to be changed; *What* relates to system properties (qualities attributes); *How* refers to change support (change models and the formality).

We advocate the use of feedback control (or closed loop control)[8], as the mechanism to support autonomic characteristics into a business process. We argue that self-adaptive software is a closed-loop system with feedback from the self and the context. The Autonomic Computer System's building block, named autonomic manager, constantly interacts with managed element in order to maintain its equilibrium in face of incoming perturbations. The MAPE cycle (Monitor-Analyze-Plan-Execute), represented in Figure 2, is an implementation of the classical feedback control technique [4].

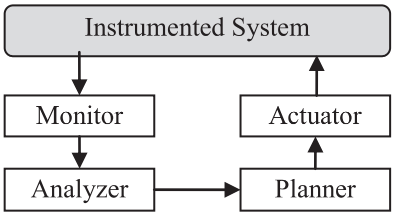


Figure 2. Closed Loop Control Mechanism

In a service-based mission-critical system, adaptation is an activity with the objective of delivering an acceptable/guaranteed service based on SLA (Service Level Agreement). Basically, SLA is a service contract between a customer and a service provider that specifies the forwarding service a customer should receive. One of the key components in SLA is SLO (Service Level Objective) which specifies QoS (Quality of Service) metrics governing service delivery. Each SLA may include several SLOs, each corresponding to a single QoS parameter related to quality factors. An example SLO is keeping the end-to-end response time in a specific range. For example, how a substituted method, replaced using an aspect weaving/unweaving operation, will violate or satisfy a condition on the system throughput [8].

In this paper, we claim that self-adaptive features must be aligned with Business Processes.

B. Business Process Modeling and Autonomic Computing Systems

Business Processes and Business Process Management (BPM) are essential in many modern enterprises. They constitute **organizational** and **operational knowledge** and often perpetuate independently of personnel and infrastructure change [9]. Autonomic Computing principles can be adapted to help organizations survive in dynamic business scenarios. Process that can be able to answer to AC principles are designated Autonomic Business Process (ABP) [10].

Autonomic Business Process (or autonomic workflow) must have the capability to adjust to **environment variations (**context). If one **component service node** (self) becomes unavailable, a mechanism is needed to ensure a business process execution is not interrupted [11]. ABP differs from traditional workflow as it relies on autonomic techniques to manage adjustments during its execution. Therefore, it enables dynamic and automatic configuration of its definition, activities and resources. It also allows self-optimization and self-healing. Furthermore, autonomic workflow must have intelligence to analyze situations and deduce adaptations at run-time.

There is a belief that self-* properties are related to software quality factors. For example, Salehie and Tahvildari discuss the potential links between these properties and quality factors [12].

We argue that in logical level of business process model the adaptations must result in quality attributes defined according NFR Framework [13]. In order to demonstrate such relationships, it is better to analyze how a well-known set of quality factors defined in the ISO 9126 -1 quality model [14] are linked to major and primitive self-* properties. Considering aspects discussed in [15], TABLE I presents a consolidation of NFR and Autonomic Computing Principles.

TABLE I. NFR AND AUTONOMIC COMPUTING PRINCIPLES

NFR	**Autonomic Computing Principles**			
	Self-Configuring	***Self-Healing***	***Self-Optimize***	***Self-Protect***
Maintainability	X	X		
Functionality	X		X	X
Portability	X			
Usability	X			
Realiability	X	X		X
Efficiency			X	

The benefits related to the use of NFR joined with BPM have been acknowledged in the last years. NFRs have been applied to help in the design of business process models through extensions of business process modeling languages, allowing for richer analysis and operationalizations of the process model. However, many works do not consider the variability in their solutions [16]. On the other hand, the **context** of a business process is the set of environmental properties that affect business process execution. Therefore, these properties should be taken into account when designing a business process. If context is analyzed when modeling a business process, then identification of all its variants (relevant states of the world in which the business process is executed) and definition of how the business process should be executed in them are facilitated [17].

Typically, self-adaptive systems have MAPE control loop and they conform to the principle of separation of concern (SoC) [18], i.e. adaptation implementation is encapsulated and separated from business logic. Considering this characteristic, we extend and refine Business Process obtained with Communication Analysis Principles and indicate abstraction levels to support AC features.

C. Communication Analysis

Communication Analysis proposes undertaking information system analysis from a communicational perspective [19]. It offers a requirements structure and several modeling techniques for BPM and requirements specification. Among these techniques, the Communicative Event Diagram and the Event Specification Templates are primary for conceptual model derivation. The *Communicative Event Diagram* is intended to describe business processes from a communicational perspective. A *communicative event* is a set of actions related to information (acquisition, storage, processing, retrieval, and/or distribution), that are carried out in a complete and uninterrupted way, on the occasion of an external stimulus. **Business process model modularity is guided by unity criteria**, method described in [20]. For each event, the actors involved are identified, as well as the corresponding communicative interactions and the precedence relations among events.

III. Our Approach

In the sequel we present our Autonomic Business Process Model approach, which consists of well-defined abstraction levels (i.e. Communicational Level; Technological Level; Operational Level and Service Level). It relies on a closed-loop mechanism to provide system adaptation (see Figure 3). Our framework considers both instrumented BPM (context) and Services (self).

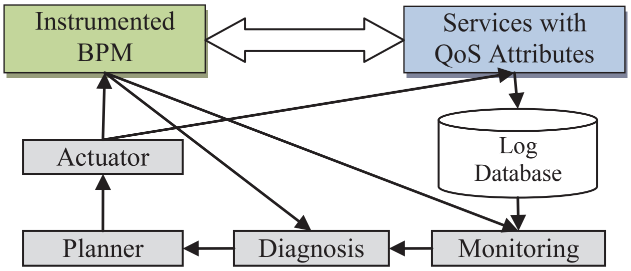


Figure 3. Framework overview

A running example will be used to illustrate our approach. We examine the CAGED (General Register of Employed and Unemployed), a project under the Ministry of Labour and Employment of Brazil (MTE) and governed by law 4923/65. It supports the submission of monthly declarations of change of company's employees due to dismissal or admittance (CAGED movements). The deadline for submission is the 7th day of every month. The data submitted is related to previous month (i.e. its competence). The declarations are processed to generate operational data, statistical data for the ministry of labor and employment.

In figure 4 we depict a process to help to define the appropriate BPM granularity and the required instrumentation of system. It consists of 7 stages: (i) Step S1 – Define Business Process with Communication Analysis Principles; (ii) Step S2 – Define Autonomic Processes; (iii) Step S3 – Link Services with Autonomic Processes; (iv) Step S4 – Define contextualization; (v) Step S5 – Define Services QoS; (vi) Step S6 – Monitor System at Run-Time; (vii) Step S7 – Choose Autonomic Processes.

We use Communicative Event Diagram and BPMN to model a simplified version of it. Note that due to lack of space we do not capture all context variations required for self-configuration, self-healing and self-protection. For the sake brevity, this paper focuses on the self-optimization feature of the "Declarant submits a declaration" event (detailed in Figure 6).

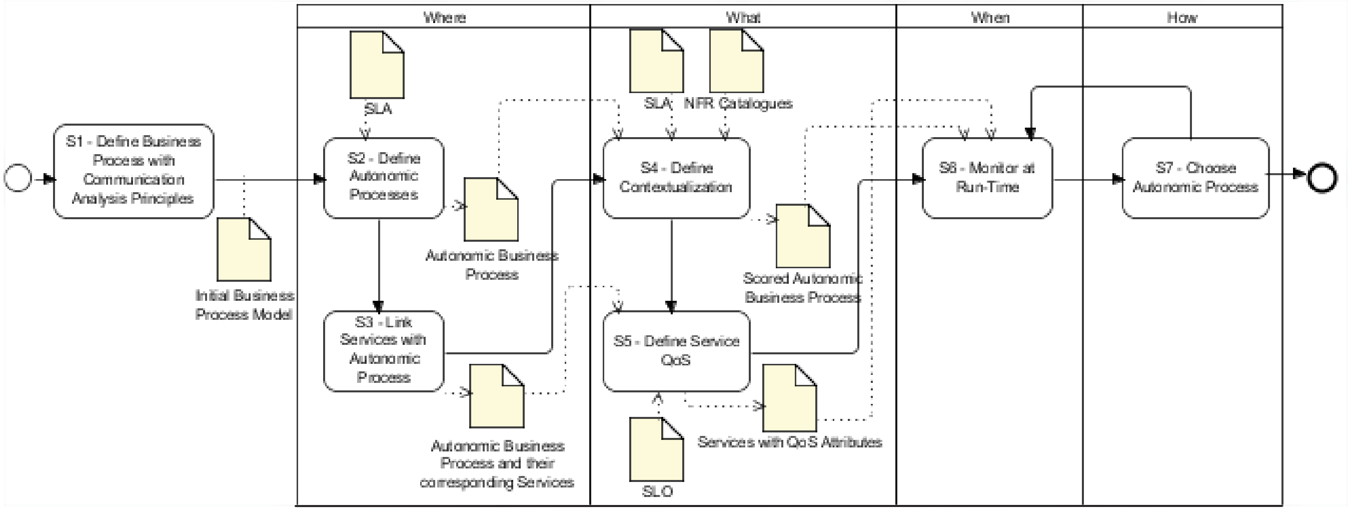

Where
What
When
How
SLA
SLA
NFR Catalogues
S1 - Define Business Process with Communication Analysis Principles
S2 - Define Autonomic Processes
S4 - Define Contextualization
S6 - Monitor at Run-Time
S7 - Choose Autonomic Process
Initial Business Process Model
Autonomic Business Process
Scored Autonomic Business Process
S3 - Link Services with Autonomic Process
S5 - Define Service QoS
Autonomic Business Process and their corresponding Services
Services with QoS Attributes
SLO

Figure 4. Process to define BPM granularity and to instrument the system[1]

A. Step S1 – Define the Business Process having in mind Communication Analysis Principles

According to the principles of Communication Analysis, a Communicative Event Diagram provides a notation to indicate the Communicational Level of a business process. At this level, it is assumed that business processes are immune to changing technologies and present the essentials of the business behavior. Event triggers an activity that receives an incoming message, processes it and provides an output.

Communicational Level is an important abstraction level in our approach which can present the critical events that must be monitored and refined. It supports the modularization of Autonomic Business Process by providing a level which presents critical events that must be refined. In this sense, we extended Communicative Event Diagram to incorporate an annotation of Critical Events. Figure 5 presents the Communicational Level of Business Process of CAGED, where we highlight the critical event "CAG2 - Declarant submits declaration".

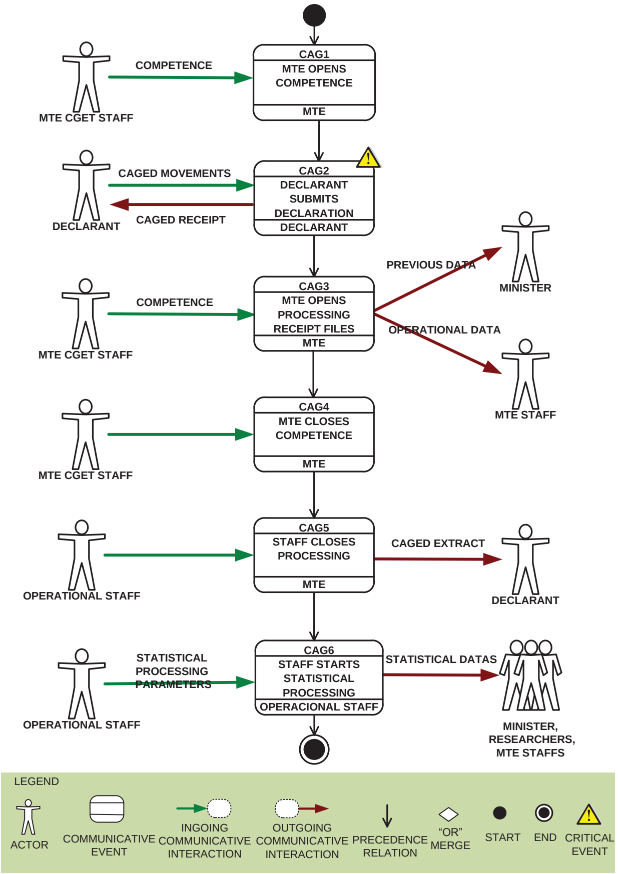


Figure 5. S1 - Communicational Level of Business Process of CAGED

B. Step S2 - Defining Autonomic Business Processes

In order to specify mission-critical events according to the logical level of business process model (Communicational Level), we defined a new level, named *Technological Level*. It expresses different activities that reflect the system behavior and the *Operational Level* to present the operationalizations related to NFR and Autonomic Computing Principles (see Figure 6, S2 box). The necessary steps to build the technological and operational level of critical events are explained as follows.

1) Define Technological Level: Technological Level represents the sub-division of an event processing to indicate important aspects that impact on software adaptation, such as:

- *Present different alternatives to perform an activity:*

It is n ecessary to in dicate possible reconfiguration, i.e. changes in business flow, as a result of deviation to certain metrics values. In our example, the operation of capture a CAGED file can be done in different manner: (i) Generate CAGED File; (ii) Generate Analyzed CAGED File; and (iii) Generate Short Analyzed CAGED File. If one of these activities became unavailable, another alternative can be e xecuted to guarantee the system operability until all processes return to an optimum state.

- *Indicate external dependences:*

External dependences are important to be expressed as they can lead to interoperability problems and impact on self-healing principle. Figure 6 presents that the task "Receive CAGED Receipt" is impacted if "Receive CAGED File" is unavailable.

- *Highlight monitorable activities*:

Some events are too complex and have to be processed by different kind of components. Hence, some are monitorable and others not. For example the "Generate analyzed CAGED File" task is executed out of MTE dominium by an offline desktop tool that is not expressed in SLA as m onitorable module. In our example the monitored tasks, such as "Receive CAGED File" are depicted in gray.

- *Define the autonomic characteristics and symptoms of monitorable activities*:

Indicate the autonomic principles that will be considered to monitor the activities. The SLA document is a good source of information. In the running example, we only consider the self-optimization (SO) principle to all monitorable activities.

2) Define Operational Level: Operational Level must express all t he tasks that must be performed if some deviation in the NFR defined in SLA occur:

- *Operationalizations according to NFR Characteristics*:

In our running example, we defined Self-Optimization (SO) as the desired autonomic principle. As previously noted, SO is related to the efficiency NFR, which in turn can be decomposed into others characteristics such as Response Time and Space Utilization. In this example, we deal only with Response Time attribute.

In order to treat Response Time deviations that may be related to the performance of the "Receive CAGED File" activity, we defined in Figure 6 (S2 box) three different activities (operationalizations): (i) Increase resource; (ii) Decrease Resource; and (iii) Analyze deviation.

C. Step S3 - Linking Autonomic Business Processes and Services

When providing software adaptation we define the link between Business Processes with their respective services, presented in the Service Level (Figure 6, box S3).

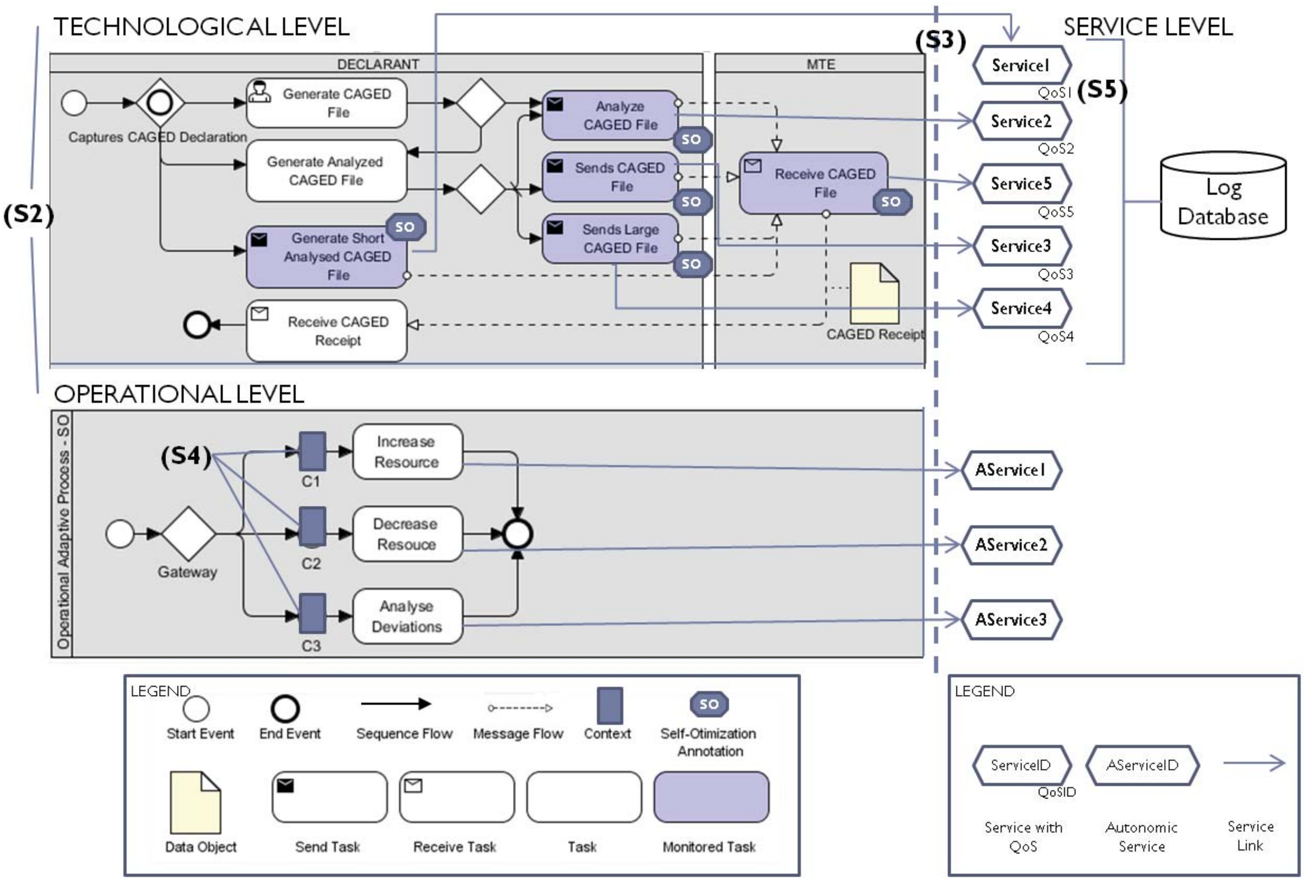


Figure 6. Steps S1, S2, S3, S4 and S5 of our approach considering the event "CAG2 - Declarant submits declaration"

Both monitorable and operationalizations tasks should be linked with system services. Operational services have the objective of return the system to an optimal state in an autonomic manner. For example, the "Receive CAGED File" activity is linked to "Service5"

D. *Step S4 – Define Contextualization*

Both contextualization and NFR are important factors to be considered as they can guide the execution of operational tasks that must provide autonomic adaptations in the system in order to return the equilibrium in face of incoming perturbations [16]. Hence in this step we define: (i) Variation Point; (ii) Variants; (iii) Context; and (iv) NFR attributes values. The 3 contexts (C1, C2 and C3) present in Figure 6 (box S4 are defined in TABLE II.

In this activity we state metrics to be monitored, i.e. we delimit the environment context that influences the choice of the system adaptation. In this sense, given that in our running CAGED there is a deadline for declaration submission, it is important to measure the daily reception. As observed in Figure 7, the reception rate peaks during the first seven days of the month. This trend has to be considered in case of a correct adaptation.

TABLE II. CONTEXTUALIZATION OF AUTONOMIC BUSINESS PROCESS

Task	**Contextualization**			
	Variation Point	***Variants***	***Context***	***NFR***
Receive CAGED File	Response Time Deviation	Increase Reception Rate	C1: ReceptionTrendIsOK= true and LastThreeCycleIncreasing= true	Response Time >= 220ms
		Decrease Reception Rate	C2:ReceptionTrendIsOK= true and LastThreeCycleDecreasing= true	Response Time <= 110ms
		Deviation Reception Rate	C3:ReceptionTrendIsOK= false and LastThreeCycleIncreasing= true	Response Time >= 220ms

E. *Step S5 – Define Service QoS*

All defined and linked services, represented as ServiceID in Figure 6, box S5, must be instrumented according to SLO. The definition of QoS is based on NFR attributes and represents the characteristics that must be guaranteed by each monitorable business activity. Considering the four send activities and one reception activity of our running example, the system must provide a *Response Time* less than 220ms to each one.

F. Step S6 – Monitor at Run-Time

The monitoring component collects indicators provided by the system from time to time (cycle) and saves them in a Log Database. In our approach, monitor must consider metrics obtained by the system (all monitored services based on monitored activities), selects the relevant data and passes it to the diagnostic component for analysis.

G. Step S7 – Choose Autonomic Process and Service

The main theme of software adaptation, considering autonomic features, is also mapped to this concept by adding an internal feedback loop to software. In our approach, this step has the objective of diagnose, plan and actuate. Considering the metrics obtained by the monitor, the diagnostic component checks the contexts expressed in the Business Process Model. In case of some deviation, the planner selects the appropriate autonomic activities and the actuator executes the Autonomic Services expressed in the Figure 6. According to the reception data (see Figure 7) for a given competence, the system must first execute AService1 (increase of resources during the first days of the month) then perform AService2 (decrease of resource after the reception has peaked during the first days of the month) and later AService3 (deviation detected from the expected maximal 220ms response time and out of peak period).

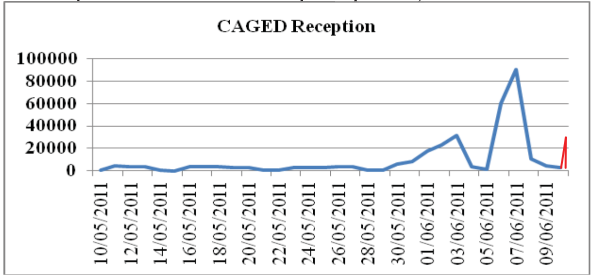


Figure 7. CAGED Reception Rate

IV. Conclusions

The expressiveness of Autonomic Computing features in Business Process Model is still a poorly explored issue. Our aim is to provide more expressivity, scalability and understandability in autonomic scenarios. For that matter we have presented an approach which relied on well-known principles such as separation of concern, modularity [3], contextualization [17], use of non-functional attributes to drive configuration [16] and Communication Analysis [20].

In particular we have argued that Communication Analysis provided a well-defined level of modularization of Business Process Model, helping to indicate mission-critical activities that must be treated in autonomic manner.

Our approach consists of four well-defined levels of abstraction: Communicational Level, Technological Level, Operational Level and Service Level. These levels provide aspects to properly express important autonomic features such as variability by the use of contexts and operationalizations of NFR.

We also outlined a process that helps to define granularity of the Business Process Models, indicating where the system need to be instrumented. A real example was used to illustrate the use of the approach.

The presented aspects were supported by a MAPE mechanism that considered both context and measurement of certain key performance indicators. System metrics were critical for the monitoring, diagnosing deviations, planning and executing adaptations.

As future work, we plan to develop the remaining MAPE components as well as to perform a formal evaluation.

References

[1] P. Horn, "Autonomic Computing: IBM's Perspective on the State of Information Technology," *IBM Corporation*, vol. 15. IBM, pp. 1-39, 2001.

[2] L. D. Terres, J. a. R. Nt, and J. M. D. Souza, "Selection of Business Process for Autonomic Automation," *2010 14th IEEE International Enterprise Distributed Object Computing Conference*, pp. 237-246, Oct. 2010.

[3] H. A. Reijers and J. Mendling, "Modularity in Process Models :," *Review Literature And Arts Of The Americas*, pp. 20-35, 2008.

[4] J. O. Kephart and D. M. Chess, *The vision of autonomic computing*, vol. 36, no. 1. IEEE, 2003, pp. 41-50.

[5] O. Babaoglu, "Self-Star Properties in Complex Information Systems: Conceptual and Practical Foundations," *Selfstar properties in complex information systems conceptual and practical foundations Ozalp Babaoglu ed*, vol. 3460. Springer, 2005.

[6] M. Salehie and L. Tahvildari, "Self-Adaptive software: Landsacape and research challenges," in *ACM Transactions on Autonomous and Adaptive Systems TAAS*, 2009, vol. 4, no. 2, pp. 1-40.

[7] J. Buckley, T. Mens, and M. Zenger, "Towards a taxonomy of software change," *Journal of Software*, pp. 1-7, 2005.

[8] M. Salehie, S. Li, R. Asadollahi, and L. Tahvildari, "Change Support in Adaptive Software: A Case Study for Fine-Grained Adaptation," in *2009 Sixth IEEE Conference and Workshops on Engineering of Autonomic and Autonomous Systems*, 2009, pp. 35-44.

[9] D. Greenwood and G. Rimassa, "Autonomic Goal-Oriented Business Process Management," *Management*, p. 43, 2007.

[10] J. A. Rodrigues Nt, P. C. L. Monteiro Jr., J. D. O. Sampaio, J. M. D. Souza, and G. Zimbrão, "Autonomic Business Processes Scalable Architecture," in *Proceedings of the BPM 2007 International Workshops BPI BPD CBP ProHealth RefMod semantics4ws*, 2007, pp. 1-20.

[11] T. Yu and K.-jay Lin, "Adaptive algorithms for finding replacement services in autonomic distributed business processes," in *Proceedings Autonomous Decentralized Systems 2005 ISADS 2005*, 2005, vol. 2005, pp. 427-434.

[12] M. Salehie and L. Tahvildari, "Autonomic computing: emerging trends and open problems," *Architecture*, vol. 30, no. 4, pp. 1-7, 2005.

[13] L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos, *Non-Functional Requirements in Software Engineering*, vol. 5600, no. 1999. Kluwer Academic Publishers, 2000, pp. 363-379.

[14] ISO/IEC 9126-1, "ISO/IEC 9126-1: Software engineering — Product quality — Part 1: Quality model," 2001.

[15] A. G. Ganek and T. A. Corbi, "The dawning of the autonomic computing era," *IBM Systems Journal*, vol. 42, no. 1, pp. 5-18, 2003.

[16] E. Santos, J. Pimentel, D. Dermeval, J. Castro, and O. Pastor, "Using NFR and context to deal with adaptability in business process models," *2011 2nd International Workshop on Requirements@Run.Time*, pp. 43- 50, Aug. 2011.

[17] J. Luis, D. Vara, R. Ali, F. Dalpiaz, J. Sánchez, and P. Giorgini, "Business Processes Contextualisation via Context Analysis," *Context*, pp. 1-6, 2010.

[18] P. K. McKinley, S. M. Sadjadi, E. P. Kasten, and B. H. C. Cheng, "Composing adaptive software," *Computer*, vol. 37, no. 7, pp. 56-64, 2004.

[19] S. Espana, A. Gonzales, and P. Oscar, "Communication Analysis : a Requirements Engineering," *Proceedings of the 21st International Conference on Advanced Information Systems Engineering CAiSE 2009*, pp. 1-15, 2007.

[20] González, A., S. España, and Ó. Pastor, Unity criteria for Business Process Modelling: A theoretical argumentation for a Software Engineering recurrent problem., in Third International Conference on Research Challenges in Information Science (RCIS 2009). 2009,. p. 173-182.

Interoperable EMR Message Generation:

A Model-Driven Software Product Line Approach

Deepa Raka and Shih-Hsi Liu
Department of Computer Science
California State University, Fresno
Fresno, CA, USA
deeparaka123@mail.fresnostate.edu, shliu@csufresno.edu

Marjan Mernik
Faculty of Electrical Engineering and Computer Science
University of Maribor
Maribor, Slovenia
marjan.mernik@uni-mb.si

Abstract—**This paper introduces a model-driven approach to extend Feature-Oriented Domain Analysis applied to heterogeneous Electronic Medical Records (EMRs) represented in XML format. Six new feature relationships are introduced to specify the structure differences among a set of XML messages conformed to different healthcare standards. Our EMR case study, applied to HL7 CDA and openEHR healthcare standards, shows that the proposed approach offers a flexible solution to model and generate an XML product line, which may be interoperated between HL7 CDA- and openEHR-based HISs.**

Keywords - Electronic Medical Records; HL7; OpenEHR; Feature Models; Model-Driven Engineering; Product Lines.

I. INTRODUCTION

Healthcare Information Systems (HISs) [7] are for healthcare institutes to store and manage medical, administrative and financial records, called Electronic Medical Records (EMRs) [7] usually in XML format. Although XML may offer advantages of comprehensibility and interoperability, extra care is still needed to map and transform between heterogeneous XML messages when such messages are requested to transfer between HISs whose EMRs conform to different healthcare standards.

Model-driven software product lines [3] are a combination of Model-Driven Software Development (MDSD) and Software Product Line Engineering (SPLE) [3] that advocates concurrently developing a set of software products sharing commonalities and variabilities. Feature-Oriented Domain Analysis (FODA) [5] is a classic approach for commonality and variability analysis. This approach consists of feature models that define features and associated dependencies, represented by feature diagrams in a hierarchical manner. In order to further enhance FODA, a number of extensions have been introduced. For example, feature and group cardinalities [2], consist-of and is-generation-of relationships between features [1], feature priorities [1], and feature modularization [2] that represents a special *leaf* node by a separate feature diagram to address the combinatory explosion problem. However, when FODA is applied to EMRs, represented in XML, in different standards, not only leaf nodes (i.e., atomic XML elements or attributes represented as <tag …>…</tag>, where no other XML element is embedded between starting and ending tags) but also *intermediate/composite* nodes (i.e., XML elements that embed other XML elements between starting and ending tags) should be analyzed. To our best knowledge, there has been no existing FODA extension that can handle this kind of structures and complexities.

This paper introduces a model-driven software product line solution to address how to utilize FODA and newly introduced extensions to analyze EMRs represented in XML and how to concurrently generate XML messages conformed to different standards. Our project starts with introducing a metamodel using the Generic Modeling Environment (GME)^{1}. The metamodel defines the syntax and static semantics of EMRs. Additionally, six new feature relationships (i.e., *Fold*, *Inverse*, *Add*, *Remove*, *Add_Remove*, and *Make_Tag*) that specify the structural differences between EMRs in different standards are introduced. Models are then introduced, acting as instances that conform to the metamodel. Most importantly, an interpreter is created to interpret models and then generate a set of interoperable EMRs conformed to different standards in parallel. In addition, the interpreter solves three challenges regarding XML tree structures: (1) For each XML element, after specifying its feature relationships and their dependencies using the classic FODA [5], the interpreter needs to correctly record each XML element's tree level and its structural relationship with other elements (e.g., parent-child and sibling) and then generate a set of XML product line members accordingly; (2) The six new feature relationships may change the structures and levels of existing XML elements, which should be addressed by the interpreter; and (3) Unlike an XML message represented graphically in a tree structure, each XML element represented in text form at has a closing tag. Inserting the closing tag of each XML element at the correct location and indent is also a responsibility of the interpreter.

This paper is organized as follows. Section 2 describes the project in more details with a case study followed by conclusion in Section 3.

II. OUR APPROACH

This section introduces the approach step by step.

A. STEP 1: Metamodel Level

Because EMRs are represented in XML, a metamodel needs to specify the structure of an XML message and the feature relationships of/between XML elements/attributes. In

[1] http://www.isis.vanderbilt.edu/Projects/gme/

Figure 1, a metamodel defines an XML message (*Project_Root*) that consists of atomic (*Child*) and/or composite (*Parent*) XML elements. Each element may contain 0..* attributes (*Attributes*) and 0..1 value (*Tag_Text* within *Parent* and *Child*). *operation* is a connection type in GME that are used to define the relationship between *Elements*. For example, the six new feature relationships are described as follows:

- *Fold* implies that the source of the connection becomes the parent of the destination. Suppose that if the first to-be-generated message (i.e., core asset) looks like:

```
<P>
    <A></A>
    <B></B>
</P>
```

Fold will also concurrently generate a second message shown as follows, if the connection is applied from A to B:

```
<P>
   <A>
      <B></B>
   </A>
</P>
```

Fold can be also applied to a chain of elements such as A - B-C-D. Then <A> will embed <B>, <C>, and then <D > in a hierarchical manner.

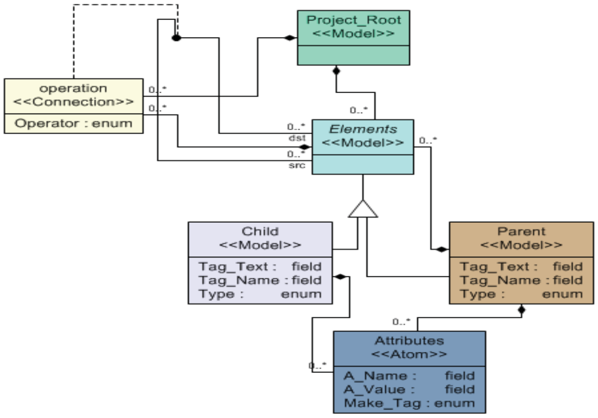


Figure 1. The metamodel.

- *Inverse* is the opposite of *Fold* – the destination of a connection becomes the parent of the source of the connection.

- *Add* (called *Add_Remove* when dealing with attributes) implies that the element that is missing in the first to-be-generated message will appear in the second to-be-generated message. For example, if the first to-be-generated message looks like:

```
<P>
   <A>
      <B></B>
   </A>
</P>
```

and if the second to-be-generated message looks like:

```
<P>
   <P1> </P1>
   <A>
       <B> </B>
       <C> </C>
   </A>
</P>
```

This can be specified by adding extra elements <P1> and <C> at the desired levels and connecting them by *Add*.

- *Remove* implies that an element appears in the first to-be-generated message should be removed in the second to-be-generated message.

- *Make_Tag* is similar to *Add* but applied to *Attributes.* If the first to-be-generated message is as follows:

```
<P A1="V1" A2="V2">
     <A> </A>
</P>
```

and the second to-be-generated message looks like:

```
<P A1="V1">
     <A2> V2 </A2>
     <A> </A>
</P>
```

Make_Tag can be applied to A2 with false value so that A2 becomes an element in the second to-be-generated message.

Additionally, these new feature relationships can also work together. For example, suppose the first to-be-generated message is the same as the first to-be-generated message described in *Fold*, and the second to-be-generated message is as follows:

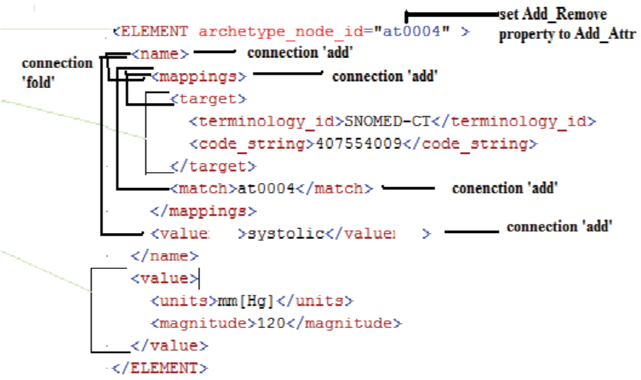
```
<P>
   <P1>
        <A> </A>
        <B> </B>
   </P1>
</P>
```

It means that <P1> will require two kinds of connections: <P1> self-connects *Add* to itself; and <P1> connects two *Folds* to <A> and to <B>.

In addition to the extensions, *Type* in Figure 1 defines FODA relationship among *Elements* in an XML tree structure. Namely, *Types* are used to specify FODA's mandatory, optional, one-of, and more-of feature relationships. If an XML element selects mandatory/optional, it means that the element is mandatory/optional to its parent/composing XML element. Similarly, if a set of XML elements with the same parent XML element selects one-of/more-of, one/more of the elements will be selected to construct a product line.

B. *STEP 2: Model Level*

This section introduces a model as a simple case study that specifies the structure, commonalities, and variabilities of two types of XML messages respectively conformed HL 7 CDA[2] and openEHR standards[3]. An XML product line then will be generated after the model is interpreted by the interpreter described in Section 3.C-F. Figure 2 shows the summary of an EMR product line modeled by FODA. For space consideration, please find model examples at [8].



```
<ELEMENT archetype_node_id="at0004" >
  <name>
    <mappings>
      <target>
        <terminology_id>SNOMED-CT</terminology_id>
        <code_string>407554009</code_string>
      </target>
      <match>at0004</match>
    </mappings>
    <value    >systolic</value    >
  </name>
  <value>
    <units>mm[Hg]</units>
    <magnitude>120</magnitude>
  </value>
</ELEMENT>
```

Figure 2. Summary of an HL7-CDA-and-openEHR product line.

C. *STEP 3:Interpreter Level -- Create Data Structures*

In order to keep track where each XML element is located and its relationships to other elements before and after applying feature relationships, two[4] two-dimensional vectors are introduced (For space consideration, Table 1 consists of two vectors which are distinguished by Columns 3 and 4). Column 2 stores a list of *Element* objects, each of which contains its feature relationship to its parent element, (i.e. mandatory, optional, one-of, or more-of). Columns 3 and 4 respectively store the parent indices of elements of two standards. Note that Column 3 has three integers. It is because Indices 3 to 8 should only be included in the openEHR message by using *Add*. When the six newly introduced feature relationships are identified in the model, Column 4 should be updated accordingly. For example, *Add*, *Fold*, *Make_Tag*, and *Add_Remove* with *Add_Attr* value may increase an element's parent level if the structure is changed; and *Inverse* and *Remove* may decrease an element's parent index. The table not only assists in updating elements' properties and relationships but also finding where to insert ending tags. Due to space consideration, please refer to [8] for the algorithm of creating data structures.

D. *STEP 4: Interpreter Level -- Create FDL Files*

Feature Description Language (FDL) [6] is a domain-specific language that describes feature models. Hence, STEP 4 is to generate FDL programs based on the results of STEP 3 (i.e., properties stored in Table 1) so that the FDL Rule Engine [6] in STEP 5 can execute the FDL programs. The interpreter starts from the root element and then retrieves those elements whose parent index indicates that they are the child elements of the current element being processed. Then depending on the parent-child feature relationship stored in the elements, the interpreter generates the textual representation for the element. The processes are repeated for each element.

TABLE I. VECTORS TO STORE XML ELEMENET PROPERTIES.

Index	Element	Parent Index	Parent Index
0	ObservationEvent	-1	-1
1	code	0	5
2	Value	0	0
3	Name	-	0
4	Mappings	-	3
5	Target	-	4
6	terminology	-	5
7	match	-	4
8	value	-	3

E. *STEP 5: Interpreter Level -- Execute FDL Rule Engine*

The FDL Rule Engine [6] is used to normalize and generate all the possible combinations of XML elements from a given FDL program. The output is a comma-separated list of XML elements for each unique possible combination.

F. *STEP 6: Interpreter Level -- Generate an XML Product Line*

The outputs from the FDL Rule Engine show all the possible combinations of XML elements based on feature relationships, both newly introduced and FODA's. However, there is no way to relate XML elements to their parent elements from the outputs, as they do not show parent elements at all. Such outputs do not offer any idea of the levels at which each element is present either. All this information is necessary in order to generate a valid XML product line that conforms to HL7 CDA and openEHR. In order to solve this challenge, the vectors generated in STEP 3 are used. The interpreter parses one combination of elements at a time. It traverses a vector and checks if an element is present in the current string of the combination of elements being considered. If yes, it adds all the child elements and attributes for the current element. Thus, the parent elements are added to the message. The interpreter then checks if any attribute is supposed to be converted into an element by checking *Make_Tag* property value and converts it into the element form and appends it to the message. It also checks if any attribute is to be added or removed using *Add_Remove*. The process is recursively repeated for every entry in the vector and an XML message is then generated. The interpreter continues to generate the rest of the XML messages based on the number of combinations from the FDL Rule Engine.

G. *A Case Study*

This section introduces a more complicated case study that consists of all the feature relationships mentioned in the paper.

[2] Health Level 7. http://www.hl7.org.

[3] openEHR. http://www.openehr.org.

[4] There could be more than two vectors. The reason "two" are created is because the product line conforms to two standards.

Figure 3 is a feature diagram that illustrates the case study. Note that the black ones in the case study conform to HL7 CDA and the black and red ones conform to openEHR standard. We use black ones (HL7 -CDA) as a core asset in a product line and generate a XML product line in parallel.

In Figure 3, "observation" is the root element. "Event", "BloodPressure", "Patient", and "type" are mandatory, and "Visitors" is optional. Child elements "systolic" and "diastolic" of "BloodPressure" are alternative, and child elements "Name" and "ID" of "Patient" are more-of. There is a Fold from "Event" to "Patient", which indicates that "Event" should be the parent element of "Patient" in openEHR product line members. Inverse between "examiner" and "type" implies that "examiner" should be a child element of "type". Additionally, "examiner" has an Add, which means "examiner" should be present as a child element of "type" in openEHR XML product line members. "SSN" is connected by Remove, so it should not be present in openEHR but should be a child element of "Patient" in HL7. "Fname" and "Lname" attributes should be converted to child elements of "name" in openEHR as they have the *Make_Tag* property set to 1. Note that "Lname"'s *Add_Remove* is set to *Add_Attr* (true), which implies that it should not appear as an element in any openEHR message. For space consideration, we again skip the model snapshot.

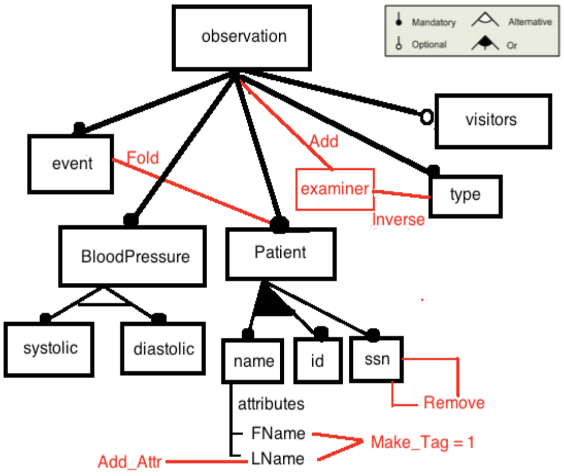
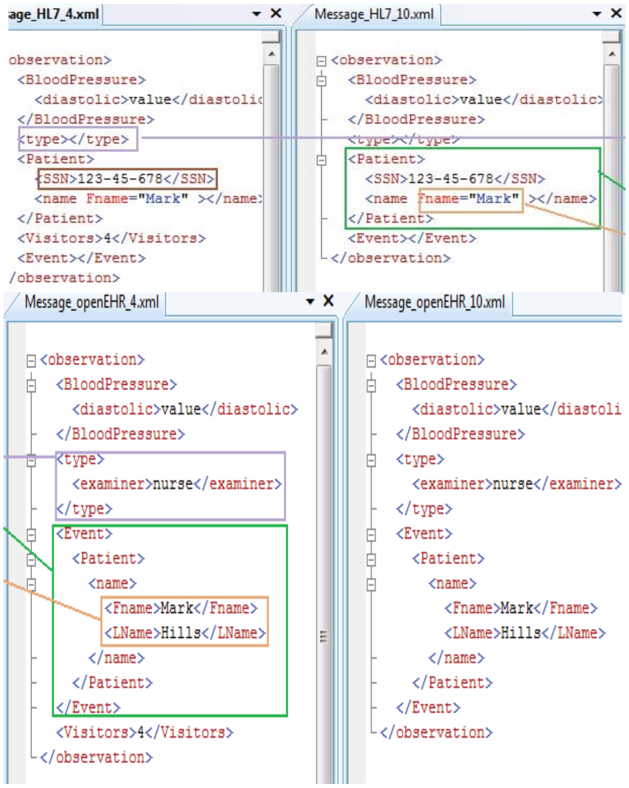


Figure 3. A case study.

The interpreter will then first create an internal table to store the positions of tags of each element. Then an FDL program will be automatically generated. The internal FDL Rule Engine then generates 12 messages for HL7 and 12 messages for openEHR are generated. Figure 4 shows two selected HL7 and two selected openEHR messages generated by the interpreter. The differences that are done by applying some of the feature relationships shown in Figure 3 are highlighted too.

III. Conclusion

As EMRs are getting more attention and ultimately will be pervasively required in healthcare domain, there is a need to solve the interoperability issue among EMRs conformed to different standards. This paper offers a flexible solution to address such an issue. Our work reproduces FODA's four main feature relationships and introduces six new feature relationships to tackle the structure changes among different XML messages. The results show that an interoperable EMR product line in XML format can be generated concurrently, which may reduce processing time, especially when patients are under emergent situations.



```
observation>
 <BloodPressure>
   <diastolic>value</diastolic
 </BloodPressure>
 <type></type>
 <Patient>
   <SSN>123-45-678</SSN>
   <name Fname="Mark" ></name>
 </Patient>
 <Visitors>4</Visitors>
 <Event></Event>
/observation>
```

```
<observation>
  <BloodPressure>
    <diastolic>value</diastolic>
  </BloodPressure>
  <type></type>
  <Patient>
    <SSN>123-45-678</SSN>
    <name Fname="Mark" ></name>
  </Patient>
  <Event></Event>
</observation>
```

```
<observation>
  <BloodPressure>
    <diastolic>value</diastolic>
  </BloodPressure>
  <type>
    <examiner>nurse</examiner>
  </type>
  <Event>
    <Patient>
      <name>
        <Fname>Mark</Fname>
        <LName>Hills</LName>
      </name>
    </Patient>
  </Event>
  <Visitors>4</Visitors>
</observation>
```

```
<observation>
  <BloodPressure>
    <diastolic>value</diastoli
  </BloodPressure>
  <type>
    <examiner>nurse</examiner>
  </type>
  <Event>
    <Patient>
      <name>
        <Fname>Mark</Fname>
        <LName>Hills</LName>
      </name>
    </Patient>
  </Event>
</observation>
```

Figure 4. Selected XML product line members.

References

[1] K. Czarnecki et al., Generative Programming and Active Libraries. *Generic Programming*. 25-39, 1998.

[2] K. Czarnecki, S. Helsen, and U. Eisenecker, "Staged Configuration through Specialization and Multilevel Configuration of Feature Models," Software Process: Improvement and Practice, vol. 10, no. 2, pp. 143 - 169, 2005.

[3] K. Czarnecki et al., "Model -driven Software Product Lines," 20th OOPSLA Companion, pp. 126-127, 2005.

[4] A. Ducrou, "Complete Interoperability in Healthcare: Technical, Semantic and Process Interoperability through Ontology Mapping and Distributed Enterprise Integration Techniques ," Doctoral Thesis, University of Wollongong, 2009.

[5] K. C. Kang, et al., "Feature-Oriented Domain Analysis (FODA) Feasibility Study," Tech. Report. Carnegie Mellon University, 1990.

[6] T. Kosar, et al., "A Preliminary Study on Various Implementation Approaches of Domain-Specific Language. Information & Software Technology," vol. 50, no. 5, pp. 390-405, 2005.

[7] J. Luo, "Electronic Medical Records. Primary Psychiatry," vol. 13, no. 2, pp. 20-23, 2006.

[8] D. Raka. "A Product Line and Model Driven Approach for Interoperable EMR Messages Generation," Master Project, California State University, Fresno, Dec. 2010.

A data collaboration model for collaborative design based on C-net

Xin Gao, Wenhui Hu, Wei Ye, ZHANG Shi-kun
School of Electronics Engineering and Computer Science,
National Engineering Research Center for Software
Engineering Key laboratory of High Confidence Software
Technologies (Ministry of Education)
Peking University
Beijing, China
gaoxin54@gmail.com

Xuan Sun
Key Lab of Universal Wireless Communications, MOE
Wireless Network Lab
Beijing University of Posts and Telecommunications
Beijing, China
E-mail: sunxuanbupt@gmail.com

Abstract—**Process modeling is the base of complex product collaborative design, and the requirements of data collaboration in collaborative design process challenge at the aspects of parameter flow, concurrency, multi-versioning, repeat design and so on. To specify the data collaboration between design processes based on parameters, this paper provides a dataflow-oriented process collaboration model based on C-net model. Through the analysis of the data collaboration between design processes, we conclude some process collaboration patterns, including level task interaction, concurrent task interaction and repeat task interaction. Then provide the structure descriptions for these patterns, and explain the application scenes of theses patterns in collaborative design. At last, define the collaborative unit for each pattern and describe the data collaboration mechanism of process collaboration patterns by their element and relationship.**

Keywords- collborative design; data collaboration; C-net

I. INTRODUCTION

In the process of complex product design, the way of collaborative product design and development is the advanced cooperation model, which organizes the distributed team into a group to develop the product collaboratively so as to improve the efficiency of development [1]. Comparing to traditional product design process, collaborative design process has the features of concurrency, distribution, dynamism and collaboration. In workflow, collaborative design process mainly depends on data collaboration based on parameter exchange. So how to implement the modeling for data collaboration of design process effectively becomes an important research topic, and at the same time it presents critical challenge to workflow modeling, which includes following aspects:

- Support iteration of design process
- Support the expression of data flow and control flow
- Support data collaboration in different scope

From these facts, we can see that collaborative design actually is the complex distributed parallel interaction process, which execute data collaboration mainly in the form of data exchange. So traditional control flow oriented business workflow can't describe complex data relationship and data exchange process, and need to take the idea and method of science workflow which mainly focus on data flow. Comparing to business workflow, science workflow can describe complex process and large-scale data exchange and transformation. At the same time, traditional business workflow can't deal with the alterations of process collaboration which are caused by the changeability of design requirements at runtime. Meanwhile science workflow support breakpoint setting, and iterative execution, which can improve the flexibility of data collaboration among processes.

Based on these features of collaborative design workflow, this paper takes the advantage of business workflow and science workflow, expresses dataflow oriented, constraint, parallel and iterative execution by C-net model. Then conclude a set of process collaboration model by analyzing the features of collaborative design workflow. At last, provide the construction mechanism of collaboration model based on C-net, so that the data collaboration model for collaborative design can ensure the workflow based on this model to effectively implement data collaboration process.

II. RELATED WORK

A. Workflow modeling

To the research of workflow modeling, currently there are still no uniform modeling standard. From current work, the modeling methods mainly include flow chart, state diagram, activity network, IDEF, ECAA, event driven process model, Petri net and so on. Among these modeling methods, Petri net is the most widely used modeling method [2].

To implement the parallel execution, collaboration and optimization between the workflows of related design teams, it needs to build the model which can reflect the relationships between upstream and downstream of design organizations, and process modeling is the important base of parallel execution. Traditional Petri net can't effectively support the expression of the combination of data flow and control flow, data collaboration, iterative execution. So C-net model which presents by doctor Yuan[3] is used in this paper to describe data collaboration among collaborative design workflows, because the system of C-net model is quantification structure which is similar with net system. “C” stands for computing and communication. C-net describes token and variable separately based on the basic principles of Petri net, so that C-net is suitable to describe collaborative design workflow.

B. Collaboration mechanism description

To the description of coarse collaboration mechanism, paper [4] integrates the branch of workflow and the object of software engineering into Petri net, and builds the invoking relationship, in which collaborate unit nets are collaborated around process net. Paper [5] presents a method of multi-workflow interaction and coordination based on bus. The research direction of these works is to build coarse collaboration mechanism to implement data translation and progress control among processes, but it can’t reflect the collaboration between tasks because its element is not task.

To the description of fine grit collaboration mechanism, paper [6] presents the solution of workflow collaboration by combing agent technology with ontology. Paper [7] describes workflow based on Petri net, and uses an annotation language to describe data exchange and process schedule between processes. Both of these two works present the description of fine grit collaboration mechanism, but they use different modeling methods to describe the workflow and the collaboration mechanism between workflows.

The inconsistency between workflow model and collaboration mechanism model brings complexity to analysis of the dynamic process data and execution state of workflow, and there are also some researches on the unified formalization of collaboration design process. Paper [8] uses Pi-Calculus technology to formally define the inter-organizational business process structure, and use the special mechanism of Pi-Calculus to describe parallel processes and their communication. But its problem is that the process model based on Pi-Calculus is too complex and abstract to let designer understand the process and conduct the implementation of specific design process. Paper [9] provides a set process modeling mechanism of pub/sub system based on color net, which adopts Petri net model to describe pub component so as to realize process collaboration and only focus on the description of control flow. So according to the feature of collaborative design, it needs a kind modeling method which can describe both design process and design collaboration of collaborative design by the data exchange of fine grit task level. The collaboration at task level asks for more requirements to process modeling, which mainly is reflected in the description of the process and collaboration between processes.

III. DATA COLLABORATION DESCRIPTION OF COLLBORATIVE DESIGN

Through C-net model, we can describe the basic process of dataflow oriented collaborative design, and we conclude three main process collaboration patterns in the scene of data collaboration between collaborative design processes. Then we use C-net to build model for process collaboration patterns.

A. C-net model

The formal definition of C-net is as follow:

Definition 1 Use $CN = \left(P, V, T; F, R, W_r\right)$ to stand for C_net，and $x \in V, \mathrm{t} \in T, p \in P$. *P*, *V* and *T* separately stand for the token set, variable set and transition set of *CN*； and *F*, *R* and *Wr* separately stand for flow relationship, read relationship and write relationship.

It is critical to realize the communication between processes by the collaboration between tasks and support the description of parameters exchange between tasks and constraints between parameters in collaborative design. C-net support parameters passing of collaborative design, because C-net introduces variable set which can make some corresponding extensions so as to provide the description of data flow and control flow between design tasks.

B. Data collaboration model of collaborative design

According the features of collaborative design, including inter-organization, data flow oriented, parallel and iterative execution, we conclude three major parameter interaction patterns to describe the complex data collaboration between design processes, including task interaction between superior process and subordinate process, task interaction between parallel processes and task interaction between iterative processes.

- Pattern 1 : task interaction between superior process and subordinate process

In this pattern, the relationship between superior process and subordinate process are reflected into parameter passing from one or more tasks of subordinate process to the task of superior process. This pattern often is used in the level structure to describe the interaction between the tasks which belong to the processes of different level. These interactions include the monitoring, collection, statistics and analysis of superior process to subordinate process, and the specific abstract process is as follow:

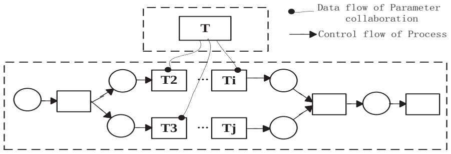


Figure 1. Task interaction pattern between superior and subordinate process

In figure 1, transition *T* strands for the task of superior process which implements the interaction with subordinate process. It shows that *T* collects parameters from *T2*, *T3* and *Ti*.

- Pattern 2 : task interaction between parallel processes

In this pattern, the tasks of different parallel processes interact with each other based on parameters, which is reflected into two scenes: one is that some tasks can pass parameters to multi tasks from other parallel processes; the other is that some tasks can receive parameters from multi-tasks from other parallel processes. In collaborative design process, this pattern is mainly used to describe data collaboration process in the form of parameter which is caused by the dependence of the design contents among different design teams. This pattern also describes the parameter interaction caused by multidisciplinary optimization which can improve multidisciplinary design. The specific abstract process of pattern 2 is as figure 2:

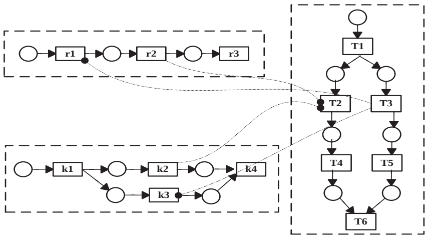


Figure 2. Task interaction pattern between parallel processes

In figure 2, transition *T3* separately reads parameters sets of *r1* and *k3*, and transition *r2* and *k2* also read parameters set produced by *T2*. They together produce complex parameter passing route, but these scattered data flows can't reflect the synchronous relationship between the transition which executes write operation and the transition which executes read operation. At the same time, the parameters sets of these parameter passing flows always cause the redundancy of data transition because of the common design contents.

- Pattern 3 : task interaction between iterative processes

This pattern is reflected into the scene that one or more transitions of iterative circuit pass parameters to the transitions of other parallel design processes. This pattern mainly describes the complex data exchange problem which is caused by multi-version process parameters produced by the iterative execution of Repeat. The multi-version property of Repeat will affect the parallel execution of other processes which exchange parameters with Repeat, and the specific abstract process of pattern 2 is as figure 3:

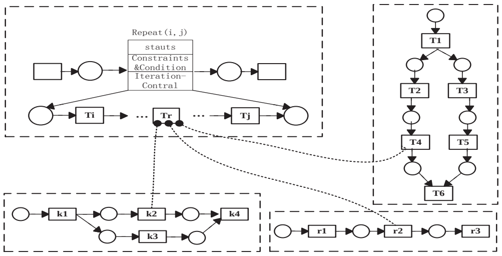


Figure 3. Task interaction pattern between iterative processes

Figure 3 describes a scene that a task *Tr* of iterative process *Repeat*(*i*,*j*) publishes parameters to the tasks of other parallel processes, including *a2*, *r2* and *t4*. The difference between pattern 2 and 3 is that the parameter passing will execute more than one times and produce the process data of different versions, which makes collaboration process more complex.

C. *Process collaboration based on C-net*

According to the features of these process collaboration patterns, we provide their model building based on C-net.

- Model building for pattern 1

In pattern 1, if we set *V* element for each parameter passing flow to stand for corresponding parameter set, it can't unified the interaction in the form of parameter collection between superior process and subordinate process and will cause the net system too complex to implement and analyze. So we use an unified middle data set variable to stand for the summarized parameters, and add the corresponding *P*-occurrence mechanism, which is called as *DX-U*, just the abbreviation of the collaboration unite of unidirectional summarization of data, just as figure 4.

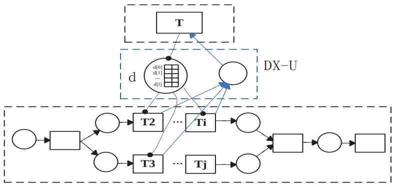


Figure 4. The C-net description of task interaction pattern between superior process and subordinate process

In *DX-U*, it uses the unified set d of parameter variables to collect the parameter set of the related tasks, just as follow:

$$d = T2.params \bigcup Ti.params \bigcup Tj.params$$

- Model building for pattern 2

The pub/sub model can be used to extend unidirectional data passing to describe complex parameter exchange between transitions effectively, and the ability of expression for parameter subscription directly affects flexibility of data collaboration between transitions. Now there are mainly two kinds of pub/sub model: the model based on theme and the model based on content. For parameter passing of the pub/sub model based on theme, the expression by C-net is as follow:

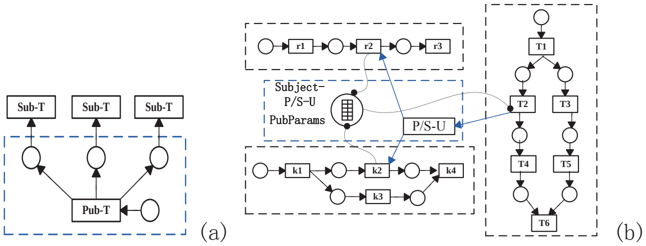


Figure 5. The C-net description of pattern 2 based on theme

Considering the feature of task interaction of parallel processes, we extend the process collaboration based on publish transition centered, and the operations include:

1) Set corresponding published parameter set called *PubParams* for each transition that executes read operation, so that let *PubParams* store the parameter set transited at runtime;

2) Add the data flow of write operation and read operation of *PubParams*;

3) Add *P* element to realize the occurrence right of parameter passing process from publish transition to subscription transition, which will build a *P*-occurrence unit of pub/sub model based on theme, as the content of dotted line box in figure 5(a), called as *P/S-U*. According the above steps, we take the publish transition *T4* for example, in which the token of *Pub-t* in *P/S-U* comes from *T4*, just as figure 5(b), and we can deal with *r1* and *k3* in figure 5 by the same way.

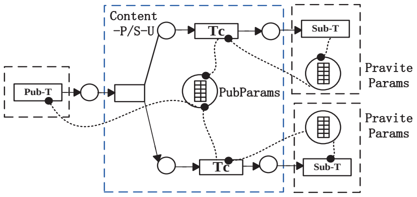


Figure 6. The C-net description of pattern 2 based on content

Though theme can mark the exchanged parameters, its expression to parameter is weak because theme can't reflect the constraint of subscription task to parameters, and its adaptation to the change of parameter constraints is also very weak. Meanwhile, the pub/sub model based on content allows subscription task to express their subscription by the value of related parameters, which has stronger ability of expression and more flexible ability of adaptation. So we extend the workflows by the C-net elements around publish transition, and the additional steps are as follow:

1) Add corresponding parameter set called as *PrivateParams* for each subscription transition to store all the parameters to be received, and add the data flow of read operation from each subscription transition to *PrivateParams*;

2) Add *P* element to realize the delivery of the right of occurrence from publish transition to subscription transition.

After these steps, it builds the collaboration unit of the pub/sub model based on content, just as the content of blue dotted line box in figure 6, called as *Content-P/S-U*. Comparing to *P/S-U*, *Content-P/S-U* has two more transitions *Tc* which are mainly used to check if the parameters of *PubParams* satisfy the corresponding requirements.

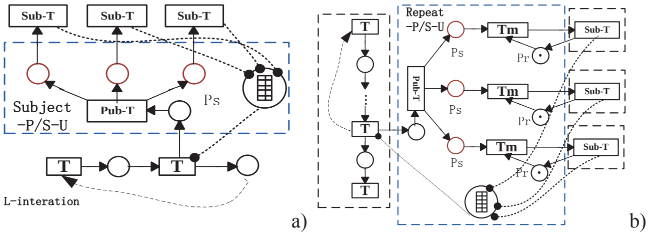


Figure 7. The Subject-P/S-U and Repeat-P/S-U solutions of pattern 3

- Model building for pattern 3

In figure 7(a), it is a scene that a task of iterative circuit publishes parameters which are passed to the tasks of other parallel processes. If we use *Subject-P/S-U* to deal with the data exchange as figure 7(a), there will be two problems: fist, the publish transition will execute more than one times publish operations. So *Sub-T* always can't consume the corresponding token in its preorder place called as *Ps* in time, which will be accumulated. And there will be repeated read operations to the same parameter set when *Sub-T* carries out read operation, which decreases the efficiency of the execution of net systems. Second, subscription transition will do more than one times read operations. As the parameters passing way of figure 7(a), there isn't any method to control the read operations of *Sub-T* to the following data.

For these two reasons, we extend *Subject-P/S-U* into the collaboration unit for iterative circuit pattern, called ad *Repeat-P/S-U*. The extension mainly is to add the consume mark place *Pr* to indicate if *Sub-T* has read the parameter set and the judge transition (called as *Tm*) between *Ps* and *Sub-T* to decide if *Sub-T* has consume the parameters that are read, just as figure 7(b). The initial value of *Pr* is one token, which means the first active read operation of *Sub-T* to the published data. After the first read, the second publication of *Pub-T* provides *Ps* another token, but the occurrence of transition Tm still need the right of occurrence of *Pr* which depends on the end of all the operations to the last set of parameters. So Tm can only inform *Sub-T* the advent of new parameters after all the operations of *Sub-T*, so that it prevents *Sub-T* to do loop detection and read operation.

IV. CONCLUTION

In this paper, we support the model building for the data collaboration of collaborative design based on C-net, which can satisfy the features of process collaboration, like parallel, iterative and dataflow oriented. We conclude collaboration pattern and provide a set collaboration unit model to describe these pattern, so that all complex collaboration mechanism can be hidden into these unite. Next step, we will prove the liveness and safety of collaborative unit, which can validate the rightness of data collaboration process.

ACKNOWLEDGMENT

This work was supported by the National Natural Science Foundation of China under Granted No. 60903001

REFERENCES

[1] Hao Bo Qiu, Y. Wang, Ping Jiang, Liang Gao. Research on Workflow Modeling Methods for Collaborative Product Development, Advanced Materials Research, vol 46, pp. 247-252, 2008.

[2] Guoyin Jiang, Lihong Dong, Overview of Workflow Modeling Theory, COMPUTER SYSTEMS & APPLICATIONS, vol 3, pp.90-93, 2006.

[3] Chongyi Yuan, Principles and Applications of Petri net. Beijing:Publishing house of Electronics Industry(PHEI), 2005.

[4] A N Yi-sheng , L I R en-hou, Analysis of cooperative design process using object-based extended Petri nets, Control and Decision, 2008, 29(3), pp.1004-1010.

[5] Yong Shi, Geert Dick Albada, Jack Dongarra, Peter M.Sloot, Towards a formal foundation for aggregating scientic workows, Proceedings of the 7th international conference on Computational Science, Springer-Verlag Berlin,2007, pp.216-219.

[6] Jiangbo Dang, Jiangbo Dang, Michael N. Huhns, Workflow Coordination for Service-Oriented Multiagent Systems, Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems, ACM New York, NY,2007, pp.1049-1052.

[7] Kamel Barkaoui, Awatef Hicheur, Towards Analysis of Flexible and Collaborative Workflow Using Recursive ECATNets, Proceedings of the 2007 international conference on Business process management, Springer-Verlag Berlin,2008, pp.232-244.

[8] PAN Xiao-hua, FENG Zhi-lin, YIN Jian-wei, ZHENG Zheng-ping, DONG Jin-xiang, Modeling Method for Inter-Organizational Workflow Based on Pi-Calculus, APPLICATION RESEARCH OF COMPUTERS, 2006, 23(1), pp.63-65.

[9] Hens, Pieter, Snoeck, Monique, Poels, Geert and De Backer, Manu, A Petri Net Formalization of a Publish-Subscribe Process System, http://ssrn.com/abstract=1886198

Working and Playing with SCRUM

Erick Passos - IFPI - Teresina - PI - Brazil
erickpassos@ifpi.edu.br

Danilo Medeiros - Infoway - Teresina - PI - Brazil
danilomedeiros@infoway-pi.com.br

Wandresson Araújo - Infoway - Teresina - PI - Brazil
wan@infoway-pi.com.br

Pedro Santos Neto - UFPI - Teresina - PI - Brazil
pasn@ufpi.edu.br

***Abstract*—Software development is sometimes considered a boring task. To avoid this fact we propose an approach based on the incorporation of game mechanics into SCRUM framework, in order to change its use to a more amusing task, by taking advantage of the gamification trend. Gamification is applied to non-game applications and processes, trying to encourage people to adopt them. This work shows a suggestion of SCRUM gamification together with an evaluation of the proposed approach in a preliminary evaluation of a software house. The use of this concept can help the software industry to increase the team productivity in natural way.**

I. Introduction

A software process is a set of activities with the goal of developing a software product. In general, software projects that are large, complex, poorly-specified, and involve unfamiliar aspects, are still particularly vulnerable to unanticipated problems. The software processes help to deal with these situations, anticipating alternatives and exposing all the tasks until the product completion. However, developers want more freedom. They do not desire to be guided by a bureaucratic process. They argue that this practice can limit their creativity. This fact was one motivation for the development of the Manifesto for Agile Software Development [2].

The Agile Manifesto motivates the creation of several software processes. Scrum, for instance, is one of the most used agile approach nowadays [3]. Scrum is a framework for project management commonly used as an agile software development model based on multiple small teams working in an intensive and interdependent way [4].

But many developers, even using Agile Methodologies, have dififculties to follow the required prescriptions. The problem related to this question is another: developing software is a challenging activity that is seldom regarded as fun. Just like many other types of activities, it can be organized as a set of hierarchical and partially ordered challenges that must be overcome, often requiring several different skills from developers, and lots of teamwork effort. Surprisingly, this is very similar to an abstract definition for games: activities in which a player must learn new skills, use and combine them to overcome challenges, getting rewards or punishments, depending on success or failure, respectively.

In this work we propose the inclusion of game design concepts into the Scrum framework, in order to try to change the software development guided by Scrum in a fun task. We show that software development using Scrum is closely related to a game when the governing rules of both activities are concerned.

We also present results from a preliminary evaluation from a real software development team, and a prototype of a tool to incorporate immediate feedback and game design features to Scrum-based project management tool. We understand that knowledge from the game academic community can contribute to other areas of applied research, and expect this work can inspire others to try this approach with different types of approaches.

A. Contributions of the Paper

The main contribution of the paper is to show that game design theory can be applied to other important real-world problems, such as Scrum framework. After a small survey about the important concepts related to this work, we present our contributions, which are summarized bellow:

- A gamification approach to Scrum framework;
- Results from a preliminary evaluation: we designed and evaluated individual and group achievements against historical data of a Scrum-based real-world development team;
- A proposal for a Scrum-based tool that incorporates game design elements, such as immediate feedback and achievements.

B. Related Work

According to Takahasi [5], gamification is a trend nowadays. The technique can encourage people to perform chores that they ordinarily consider boring, such as completing surveys, shopping, or reading web sites. In this paper, we propose going beyond with the gamification concept, applying it not only to interactive media, but to real-world activities, in this particular case, in the software development guided by Scrum.

An example of gamification in the software development context is the RedCritter Tracker[1] system. It is a software development task management tool that applies some mechanisms, like rewards and skill badges, after task accomplishments are achieved. Our proposal goes beyond the gamification of a tool, aiming to transforming the actual software development process into a game.

[1] http://www.redcrittertracker.com

Visual Studio Achievements, a Visual Studio[2] plug-in, enables developers to unlock badges and compete against anothers developers for a place on a leader board based on the code they write, its level of sophistication, and the Visual Studio capabilities they used. This is another example of gamification applied to software development close to our approach. However, we extend it by suggesting not only a tool gamification, but also a process gamification and a preliminary evaluation analysing a real team.

The idea of incorporating gamification concepts into the software development has started when we proposed the inclusion of game mechanics directly into a general software development process [8]. We have also specified a task management tool based on this proposal. Now we are specializing the approach to deal with a specific software process: Scrum.

It is important to highlight that there are personal pages, like blogs, considering Scrum as a gamification of software development. This work is completely different: we are interested in the proposal of a formal Scrum gamification, together with a specification of a tool to support this task, and associated with a preliminary evaluation of the proposal in a real software development scenario.

II. Scrum

Scrum is an agile framework for completing complex projects. Scrum originally was formalized for software development projects, but works well for any complex, innovative scope of work [4].

When using Scrum, requirements specified with the help of the *Product Owner*, generate a *Product Backlog*. This backlog is broken down into small features that can be delivered as working software during short development iterations, called *Sprints*. Work is pulled from the Product Backlog into a *Sprint Backlog* during a *Sprint Planning Meeting*. The *Team* must conclude the backlog during the sprint. Daily the team performs the *Daily Scrum* activities, including the updating of the *Burndown chart*. The sprint ends with a sprint *Review and Retrospective*, guided by the *Scrum Master*. The outcome of a sprint is a deliverable that, ideally, can be released immediately after acceptance by the product owner at the sprint review meeting. Part of these tasks are presented in Figure 1.

III. Game Design Concepts

The ultimate goal while transforming an activity into a game is to make it challenging and fun at the same time. Well designed game mechanics are the atoms that differentiate a fun game from other leisure activities such as reading. There are many definitions for game mechanics, but we'll use the one proposed by game designer Daniel Cook [9], which is in turn based on Raph Kosters Theory of Fun [10], for the scope of this work:

”Game mechanics are rule based systems / simulations that facilitate and encourage a user to explore and learn the properties of their possibility space through the use of feedback mechanisms.”

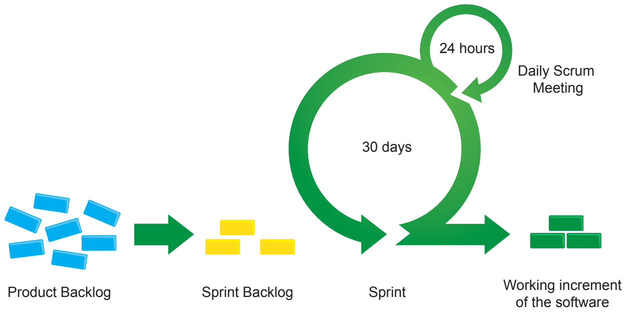


Fig. 1. Scrum framework.

In the process of designing gameplay rules, one has to consider that the challenges must be well balanced, since the brain stops enjoying challenges that are either too difficult or too easy to surpass. In this context, another element of game design that we wanted to bring to Scrum framework is the challenge-punishment-reward loop, which is closely related to the aforementioned definition of game mechanics.

A. Achievements

Achievements are a secondary scoring mechanism that measures the use of skills to solve the same type of challenge (normally a low level one). Often, achievements are computed using two approaches: repetition or rate.

Repetition achievements are defined as the number of times the player uses a certain skill to solve the same type of challenge such as killing a certain type of enemy. Eq 1 shows a formula for this type of achievement, where n denotes the total number of times challenge c was tried, and $f(c)$ is the value for its succeed function. Rate achievements are constrained to a scope, either the container challenge or a defined time slice, and normally measure the rate between succeeded against total attempts for a particular challenge. Eq 2 defines a formula for computing rate achievements.

$$rep = \sum_{i=1}^{n} f(c) \tag{1}$$

$$rate = \frac{\sum_{i=1}^{n} f(c)}{n} \tag{2}$$

Repetition achievements are often awarded based on an approximate logarithmic scale, rendering each subsequent level increasingly more difficult to earn. For rate achievements, there are multiple levels as well, normally following a linear scale of thresholds, often rewarding the player with a medal: either a bronze, silver or gold, depending on the threshold reached. They are also cumulative, meaning that the player can collect more medals for subsequent scopes.

B. Immediate Feedback

A key aspect of (specially electronic) games is the use of immediate feedback to keep the player aware of his progress (or failures) through the challenges. It is desirable that this

[2] http://www.microsoft.com/visualstudio/

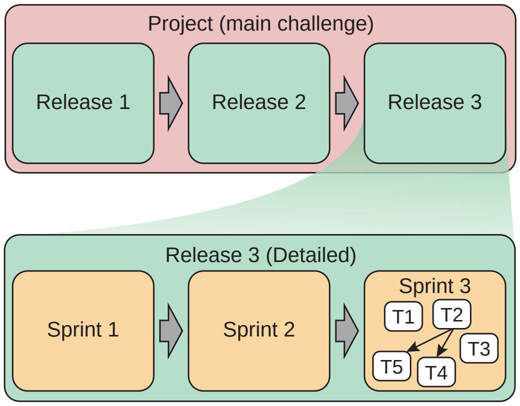


Fig. 2. Mapping of Scrum build blocks into a hierarchical challenge graph

feedback is given in real-time, increasing the feeling of immersion and also the perception that the player is the sole responsible for the outcome. Ideally, any activity that is suitable to using game design ideas should consider incorporating immediate feedback, as it is mandatory to maintain the sense of urge.

IV. Mapping Game Mechanics to Scrum

In order to create a game-like mechanics/rules to Scrum, the first concept one needs to map is the challenge-graph. The fundamental goals of Scrum must be adapted as challenges to compose the nodes. Figure 2 shows a mapping of Scrum to a challenge graph.

The leaf-nodes in this graph are the team tasks. Each team member must use their skills in requirements, design, programming and testing to succeed. The evaluation of the completion for these tasks is normally made by a PO, and logs indicating this are registered in the Burndown Chart. The major goal is to deliver a release, but this can be split in several inner goals, like to acomplish the sprints.

A. Metrics to Achievements

Besides the straightforward mapping of a software development project usign Scrum hierarchy into a challenge-graph, many companies also define and use a fairly large set of numeric metrics to either measure or even reward the best developers and teams. These metrics are commonly based on the developers performance in task planning and execution, programming, testing or other activities.

Each of these metrics have specific meanings in the Scrum context, but simply measuring and exposing them is not compelling because they lack reference goals and sense of competition. In order to convert metrics into desirable challenge, we propose the design of individual and team achievements. Individual achievements can be based in any individual metric, even if it includes tasks from more than one project. Team achievements are bounded to projects, and ideally should be based in team-grade metrics.

Any of these metrics can be converted in achievements, either repetition or rate based. However, it is important to carefully design and balance the levels (for repetition achievements) and thresholds (rate ones), in order to make them interesting. The first level and lowest grade medal (bronze) must be very easy to achieve, while the others must be rendered increasingly challenging. In Section V we show a preliminary evaluation of both solo and collective achievements.

V. Preliminary Evaluation

We have done a preliminary evaluation of our proposal involving real-world scrum teams. Infoway[3] is a medium-sized software house with 28 developers, split in four teams, which has been successfully using agile processes for six years, and using Scrum since 2008.

Infoway has a Scrum-based project management system that does not implement gamification in its behavior. But the system has information about some Scrum cerimonies, including planning, review and retrospective. There is also information about the backlog and estimated and actual time spent to accomplish tasks, grouped by sprints and projects. We used these information to evaluate our proposal in a real case.

In this section we describe the achievements designed to Scrum framework together with the evaluation of these achievements using the Infoway historical database. The preliminary evaluation comprises the design of ten game achievements and the evaluation of four game achievements based on measured metrics.

A. Achievements

As discussed before, achievements are representation of having accomplished something. Achievements can be easy, difficult, surprising, funny, accomplished alone or as a group. In this section we describe the achievements directly related to Scrum. The main idea related to the above described achievements is to keep all staff informed about the accomplishments.

It is important to emphasize that all the achievements are related to a project. A project is a temporary endeavor with a defined beginning and end. During a project, several sprints could be acomplished. This is the time box used to account the proposed achievements in a organization.

1) Role Achievements:

- **Super Scrum Master (SSM)**: A Super Scrum Master is an individual repetition achievement that accounts for the number of sprints that all the Scrum prescriptions were followed entirely. The achievement has 3 levels: Level 1 (>=1); Level 2 (>=3); and Level 3 (>=10).
- **PO Presence (POPr)**. PO Presence is a repetition achievement that accounts for the number of sprints which the PO participated of planning and review ceremonies. The achievement has 3 levels: Level 1 (>= 1); Level 2 (>= 3); and Level 3 (>= 10).
- **PO Partner (POPa)**. It is a rate achievement that accounts for the number of sprints which the PO participated of planning and review ceremonies compared with the total number of planning and reviews performed in

[3] http://www.infoway-pi.com.br

the project. This achievement has three thresholds/medals defined to the percentual of acceptance: 🥉 Bronze (50%); 🥈 Silver (75%); and 🥇 Gold (100%).

2) *Artifact Achievements:*

- **Sprint Backlog Completion (SBC)**. SBC is a repetition achievement that accounts for the number of sprints where all the item in the Sprint Backlog were developed. The achievement has 3 levels: Level 1 (>= 1); Level 2 (>= 3); and Level 3 (>= 10).
- **Sprint Review Acceptance (SRA)**. SRA is a team rate achievement that accounts for the number of accepted items by the PO in a review meeting compared to the total of items from the backlog of the current sprint. For this achievement three thresholds/medals were defined to the percentual of acceptance: 🥉 Bronze (50%); 🥈 Silver (75%); and 🥇 Gold (100%).

3) *Cerimony Achievements:*

- **Clockwork Developer (CD)**. CD is a rate individual achievement that accounts for the number of tasks a developer finished compared to the total number of tasks in an sprint. For this achievement three thresholds/medals were defined to the percentual of acceptance: 🥉 Bronze (50%); 🥈 Silver (75%); and 🥇 Gold (100%).
- **Clockwork Team (CT)**: CT is a repetition team achievement that accounts for the number of tasks finished by the team in a Sprint, on the planned time (with difference up to 20%). The achievement has 3 levels: Level 1 (>= 1); Level 2 (>= 3); and Level 3 (>= 10). The Clockwork Developer and Clockwork Team achivements are classified as cerimony achievements because the success to accomplish a task on the planned time is associated to a well done planning.
- **Daily Scrum Developer Presence (DSDP)**. DSDP is an indivual rate achievement that accounts for the number of participations of each developer at standups during a Sprint compared to the total of days of each Sprint. For this achievement three thresholds/medals were defined to the percentual of acceptance: 🥉 Bronze (50%); 🥈 Silver (75%); and 🥇 Gold (100%).
- **Daily Scrum Team Realization (DSTR)**. DSTR is a team rate achievement that accounts for the number of days that a daily scrum meeting was performed by the team compared to the total number of days of a Sprint. For this achievement three thresholds/medals were defined to the percentual of acceptance: 🥉 Bronze (50%); 🥈 Silver (75%); and 🥇 Gold (100%).
- **Sprint Latency (SL)**. SL is a repetition achievement that accounts for the number of sprints started after a sprint end within a maximum time gap of 24 hours. The achievement has 3 levels: Level 1 (>= 1); Level 2 (>= 3); and Level 3 (>=10).

B. Results and Analysis

The evaluation performed has considered four achievements: one rate achievement for individual developers and three repetition team achievements.

As mentioned before, we used historical data to compute the aforementioned achievements and award a selection of four project teams, comprising a total of 16 developers, during a time frame of seven months, which encompasses from nine to twelve Sprints from these four projects. The difference in the total amount of sprints for each project exists because each team can eventually work with different time boxes, also having time gaps among iterations.

The result of the clockwork developer achievement award is shown in Table I. There is the indication of the number of medals that would be earned by each developer, according with his project. The developers could have earned up to ten gold medals in the considered period. From the analysis, we can observe that an individual developer would win earn up to seven medals. At least one medal will be earned for each developer.

Clockwork Developer		
Project	**Developer**	**Medals**
Project I	Developer A	🥇🥇🥈🥈🥉🥉
Project I	Developer B	🥇🥇
Project I	Developer C	🥇🥈🥈🥈🥉🥉🥉
Project I	Developer D	🥇
Project I	Developer E	🥇🥇🥈🥈🥉🥉
Project II	Developer F	🥈
Project II	Developer G	🥇🥇🥈
Project III	Developer H	🥇
Project III	Developer I	🥇
Project III	Developer J	🥇🥈🥉🥉
Project III	Developer K	🥇🥉
Project IV	Developer L	🥇
Project IV	Developer M	🥈🥈🥉🥉
Project IV	Developer N	🥇🥈🥈🥉
Project IV	Developer O	🥇
Project IV	Developer P	🥇🥉🥉
Project IV	Developer Q	🥇🥇🥇🥉🥉🥉🥉

TABLE I
INDIVIDUAL RESULTS FOR THE CLOCKWORK DEVELOPER ACHIEVEMENT, GROUPED BY PROJECT: TOTAL MEDALS EARNED BY EACH DEVELOPER DURING THE SPRINTS CONSIDERED IN THE PRELIMINARY EVALUATION.

Table II shows the levels registered for each team considering the Clockwork Team achievement. All project teams reached at least the *Level 1*. This means that all teams completed at least one sprint on time. Projects II and III completed two sprints on the time among nine and twelve sprints respectively, but this fact did not assure a level update. Only Project I reached *Level 2*, since it has completed six sprints on time.

Table III shows the levels registered for Sprint Backlog Completion achievement. Only Project IV did not reach the *Level 2*. The Project II team completed all the sprint backlog six times, from nine possible times. It is the best team in this criteria, however it did not reach the *Level 3*.

Table IV shows the levels registered for Sprint Latency

Clockwork Team	
Project	**Level**
Project I	Level 2 (6/9 sprints)
Project II	Level 1 (2/9 sprints)
Project III	Level 1 (2/12 sprints)
Project IV	Level 1 (1/10 sprint)

TABLE II
TEAM RESULTS FOR THE CLOCKWORK TEAM ACHIEVEMENT: LEVELS ACHIEVED FOR EACH PROJECT AND TOTAL NUMBER OF SPRINTS FINISHED ON TIME.

Sprint Backlog Completion	
Project	**Level**
Project I	Level 2 (3/9 sprints)
Project II	Level 2 (6/9 sprints)
Project III	Level 2 (3/12 sprints)
Project IV	Level 1 (2/10 sprint)

TABLE III
TEAM RESULTS FOR THE SPRINT BACKLOG COMPLETION: LEVELS ACHIEVED FOR EACH PROJECT AND TOTAL NUMBER OF SPRINTS FINISHED ON TIME.

achievement. Only Project I did not reach the *Level 2*. The Project III and IV teams have started five sprints without time gaps. They are the best ones in this criteria. However they did not reach the *Level 3*.

Sprint Latency	
Project	**Level**
Project I	Level 1 (1/9 sprints)
Project II	Level 2 (3/9 sprints)
Project III	Level 2 (5/12 sprints)
Project IV	Level 2 (5/10 sprint)

TABLE IV
TEAM RESULTS FOR THE SPRINT LATENCY: LEVELS INDICATING THE SPRINTS WITHOUT TIME GAP AMONG THEM.

Figure 3 shows a comparison among projects, considering achievements evaluated. It is possible to notice that projects are in a similar stage. Project I is the best in two achievements, buts it is not so far from the others projects. Project IV has second bigger average number of Clockwork Developer, however it has the worse grades for two team achievements. This fact reveals an abnormal behavior.

Interestingly, due to this apparent inconsistency, the company decided to do further investigations to understand the reasons behind this behavior. This investigation lead to the conclusion that the use of achievements not only may help engage teams into doing their work, but can also help monitor, control and improve the development using Scrum process as a whole. Given the results obtained, all team members that we talked to became interested in further developments of this work. Gamification stimulates the team, and the achievements together with the immediate feedback help all the involved people to be aware with their work.

Of course, it is possible that the inclusion of achievements could stimulate some undesirable behavior, like the high competitiveness level of the individuals, affecting the team performance. However, we will apply a long term evaluation in order to analyze this effect together with another possibilities.

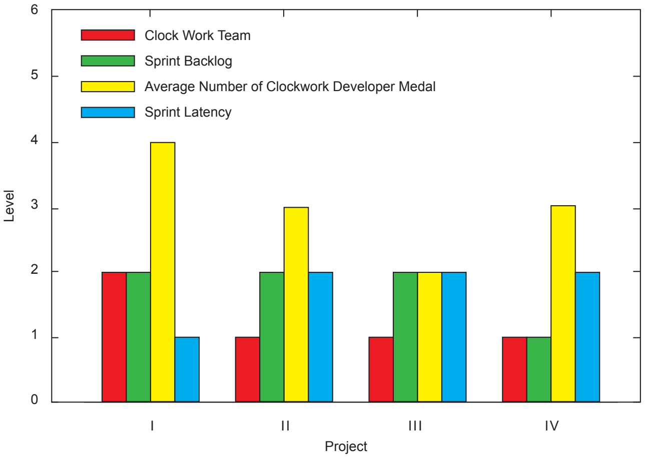


Fig. 3. Comparison of project teams using the proposed achievements.

VI. RUPGY: A SUPPORTING TOOL FOR SCRUM GAMMIFICATION

The mapping of the proposed achievements in Scrum framework makes clear for us that the idea related to Scrum gamification could be useful in order to turn the software development in a amusing task. Based on this belief we propose here a set of RPG-like features to be integrated into a scrum-task-management tool.

The goal is to include the game design features as an add-on, implementing as many feedback mechanisms as possible, making the developers always aware of these mechanics. Firstly, we will list the standard features required from a task-management software to make it eligible to incorporate the RUPGY elements. In Subsection VI-B, we suggest how the RPG concepts and features may be implemented and added to such software.

A. Standard ScrumFeatures Required

In this section we present only requirements related to a project management tool commonly used in Scrum-based software houses. The main goal of this tool is normally to register planning and to acomplish the sprints. Requirements are presented as a simple list of expressions, illustrating the concepts required, followed by an explanation that details the requirement and its use.

- **Project, backlog, release, sprints, and task containers**. A software house must keep in touch all the information about the projects in execution. A project has several requirements, represented by the backlog, to be delivered in several iterations, each one composed of several tasks. All these relationships among the project, backlog, release, sprint and tasks must be recorded, since the achievements proposed in this work requires the analysis of them in their own scope and branches to give immediate feedback.

- **Fine-grained logs for tasks (developer, time, requirement, discipline)**. It is fundamental to record the developer assigned to each task, the time spent for this execution, associated requirements, and the nature of the task (discipline). It must be easy to start, stop, pause and finish a task. Additionally, it is also important to create functions to help the developers remember doing this actions.
- **Scrum ceremonies**. The Scrum framework is intensively based on ceremonies. These events have to be registered, in order to allow the achievements evaluation. It is necessary to register when, what, why, who, how and how many time.

B. *RUPGY Game-based Proposed Features*

RUPGY is a proposal to incorporate RPG-like mechanics to the everyday use of Scrum. The goal is to make the developer more aware of his *programming character*, creating emotional bonds with this virtual persona, and thus better engaging in his daily duties. In this section we list the desired features for RUPGY.

- **Character Attributes Engine**. In RUPGY, the skills used to solve challenges are the software engineering disciplines (requirements, analysis, design, implementation, testing, project management) and since each task is related to one of these, the goal is to have the system compute the amount of experience the developer gains after finishing his assigned task.
- **Class Engine**. Given that attributes are not chosen, the **character class** must also be automatically inferred based on the most used disciplines for each developer (programmer, tester, Scrum master, Product Owner). Similar to repetition achievements, each class must have a threshold scale of minimum experience point to reach different levels.
- **Achievements Engine**. In our preliminary evaluation, we mined the achievements directly onto the task database. For the RUPGY tool, this feature must be automated, in order to give immediate feedback whenever an achievement is earned.
- **Immediate Feedback**. Developers update the state of their tasks all the time. Sometimes, an update generate a status change in the related task, or even project. The system should give immediate feedback, both visual and audible, of this changes in the character attributes, level and achievements.
- **Character Profile Screen**. The aforementioned character data must be available to the developer in a character profile screen, with historical information in graphical form.

VII. Conclusion

The Software Engineering has over 40 years of existence. There are advances in many directions but we are still researching and creating tools to solve the same problems registered 40 years ago, like projects running over-budget, projects running over-time and software often did not meet requirements.

As mentioned by Brooks [1], there is no silver bullet. But it is necessary to change this activity in something that can engage developers more effectively. In this paper we showed that game mechanics can be applied to Scrum framework. This approach has the main goal of turning software development with Scrum in a more amusing task, like a game. In order to allow this we presented several achievements that can be incorporated to a Scrum, together with the mapping of challenge-graph from games to Scrum framework.

We also presented a preliminary evaluation from a real software house. Some of the proposed achievements were evaluated against historical data. The results from this analysis showed that achievements can be an interesting metric to measure performance in this context, also helping to stimulate competition among developers. The feedback obtained from the team was encouraging (from informal inteviews), as everybody was impressed with the results and enthusiastic about the idea of incorporate even more mechanics into their activity.

The next step in this research is the development of a Scrum-based project management tool incorporating all the suggestions made here. After this will be possible to run a long term experiment with even more development teams.

References

[1] F. P. Brooks, Jr., “No silver bullet essence and accidents of software engineering,” *Computer*, vol. 20, no. 4, pp. 10–19, Apr. 1987. [Online]. Available: http://dx.doi.org/10.1109/MC.1987.1663532

[2] M. Fowler and J. Highsmith, “The Agile Manifesto,” In Software Development, Issue on Agile Methodologies, http://www.sdmagazine.com, last accessed on March 8th, 2006, Aug. 2001.

[3] “State of Agile Development,” http://www.versionone.com/state_of_agile_development_survey/10/default.asp [last accessed: 2011-04-04], VersionOne, 2010. [Online]. Available: http://www.versionone.com/state_of_agile_development_survey/10/default.asp

[4] K. Schwaber, *Agile Project Management With Scrum*. Redmond, WA, USA: Microsoft Press, 2004.

[5] D. Takahashi, “Gamification gets its own conference,” http://venturebeat.com/2010/09/30/gamification-gets-its-own-conference/, September 2010.

[6] A. Baker, E. O. Navarro, and A. V. D. Hoek, “Problems and programmers: An educational software engineering card game,” in *In ICSE 03: Proceedings of the 25th International Conference on Software Engineering*. IEEE Computer Society, 2003, pp. 614–619.

[7] A. Baker, E. O. Navarro, and A. van der Hoek, “An experimental card game for teaching software engineering processes,” *Journal of Systems and Software*, vol. 75, no. 1-2, pp. 3 – 16, 2005, software Engineering Education and Training. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121204000378

[8] E. Passos, D. Medeiros, P. S. Neto, and E. Clua, “Turning real-world software development into a game,” in *In SBGAMEs 2011: Proceedings of the X Simpósio Brasileiro de Games e Entretenimento DIgital*, Salvador, BA, Brazil, November 2011.

[9] D. Cook, “What are game mechanics?” http://www.lostgarden.com/2006/10/what-are-game-mechanics.html, October 2006.

[10] R. Koster and W. Wright, *A Theory of Fun for Game Design*. Paraglyph Press, 2004.

[11] D. Cook, “What activities can be turned into games?” http://www.lostgarden.com/2008/06/what-actitivies-that-can-be-turned-into.html , June 2008.

Follow-the-Sun Software Development: A Controlled Experiment to Evaluate the Benefits of Adaptive and Prescriptive Approaches

Josiane Kroll, Alan R. Santos, Rafael Prikladnicki, Estevão R. Hess, Rafael A. Glanzner, Afonso Sales,
Jorge L. N. Audy, Paulo H. L. Fernandes
Computer Science School
Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Porto Alegre, Brazil
{josiane.kroll, alan.ricardo, estevao.hess, rafael.audy}@acad. pucrs.br, {rafaelp, afonso.sales, paulo.fernandes, audy}@pucrs.br

Abstract— **Follow-the-Sun (FTS) has been used in the context of global software development projects in order to take the advantages of temporal differences between several productions sites located in different time zones. However, the lack of FTS experience in the software industry is observed as the main barrier for its adoption. Recent studies suggest that FTS is more suitable for adaptive approaches (e.g. agile methodologies). For this reason, we designed and executed a controlled experiment to investigate the benefits of adaptive and prescriptive approaches for FTS. We used fictional maps with teams distributed in two sites. Each site had two teams representing software designers and developers. Our results indicate that the use of adaptive approaches increases the speed, but not the accuracy and the quality of the work.**

Keywords- Global software development; follow-the-sun; adaptive approach; prescriptive approach; software engineering.

I. Introduction

Many software development companies are looking for costs reduction and the increase of productivity [1]. Follow-the-Sun (FTS) is a software development strategy where several productions sites are located in different time zones in a way that work can be done in a twenty-four hour basis per day, seven days per week. The main goal of FTS is to reduce the time-to-market, or increase the development speed [2].

FTS is not easy and many challenges have been reported in the literature, such as communication difficulties, process coordination, team management and cultural diversity [3]. Carmel et al. say that few FTS success cases are reported in the literature [4], and several authors discuss that the use of adaptive approaches for software development is a promising way to make FTS work [2] [5] [6].

Adaptive approaches or adaptive methods (e.g. agile methods and iterative development) are aimed to rapidly adapt to the changing reality. An adaptive approach emphasizes communication and collaboration in an iterative process [7, 16]. On the other hand, prescriptive approaches or methods (e.g. waterfall lifecycle) aimed at future plans in detail. The team works in linear-sequential planned activities, and usually struggle to deal with change [2]. For this reason, we conducted a research in order to investigate adaptive and prescriptive approaches for FTS. We performed a controlled experiment with students from a postgraduate Computer Science program PUCRS, a private University in the south of Brazil. We used fictional maps and simulated distributed teams in two sites. Each site had two teams representing software designers and developers. We collected and compared data about speed, accuracy and quality of the work developed by the teams in each approach.

This paper is structured as follows: in Section 2 we present some background about Follow-the-Sun (FTS). In Section 3, our research questions and hypothesis are presented. In Section 4, we present the experimental design and explain the experiment execution. In Section 5, we present the results obtained in this experiment. In Section 6, we discuss the results and Section 7 concludes the paper.

II. Follow-the-Sun Software Development

One of the main purposes of FTS is the reduction of software development cycle duration [2]. FTS software development teams are globally distributed across different time zones and production locations [8]. When a team finishes its own regular hours of work, another team located in another site and another time zone will receive the tasks to start its workday. Daily production handoffs are performed by teams following to the next production site that will be in a different time zone and will continue the work of the previous teams [2] [9].

In FTS, tasks are handed off from one team to another in the end of a working day in a specific site [4] [8]. A handoff is then the transition between teams and can be defined as a check-in of a work unit to the next production site [4]. In the context of FTS, handoffs are conducted on a daily basis, at the end of each site shift. However, daily handoffs coordination is very difficult and requires different practices to reduce coordination costs [2].

FTS is an important research area, but so far, it is relatively understudied within Software Engineering. The success cases of FTS usage in industry are still small. Carmel, Espinosa and Dubinsky (2009) claim that there are few documented success cases in industry and many difficulties in applying FTS practices [4]. There are several challenges related to communications, process coordination, team's management, cultural differences and geographic differences [10].

III. The Experiment

Our goal in this study is to investigate the use of the adaptive and prescriptive approaches in the context of FTS. We have defined the following research questions (RQ):

RQ1: *Are teams using adaptive approaches faster than teams using prescriptive approaches for software development in the context of FTS?*

RQ2: *Do teams using adaptive approaches deliver more accurate work than teams using prescriptive approaches for software development in the context of FTS?*

RQ3: *Do teams using adaptive approaches deliver more quality than teams using prescriptive approaches for software development in the context of FTS?*

A. The Study Design

Our experimental design was inspired in the laboratory study conducted by Espinosa, Nan and Carmel [12]. The authors executed an experiment to check the impact of time zone overlap on speed and accuracy. The teams used simple fictional map that consist of a background image, with various objects and colored arrows styles. The maps production was done in using PowerPoint© slides. Participants worked in pairs and the objective was to complete the maps according to the defined requirements. We borrowed from the authors the tasks performed by the teams during the development of our experiment. The maps design task was also used in another study [8]. In this study, the goal was to check the impact of increasing the number of sites in the FTS cycle on the quality of the work that was delivered at the end.

To conduct the experiment with teams working on FTS, we chose the context of a university with students from a computer science postgraduate program at PUCRS. In this environment, the experiment was simulated with the definition of distributed teams. The experiment dimensions are defined as:

- Process: in vitro, with the participants located in a controlled environment. This experiment was executed by the development of experiment activities as part of a post graduation class in two isolated labs.
- Participants: the experiment was executed with postgraduate students.
- Scenario: the studied problem was an academic task, and it was represented by the execution of tasks in a fictional task map [12].
- Generality: the experiment was specific and it is valid only in the scope of this study.

Selection of participants: We invited by convenience 12 students from Computer Science postgraduate program at PUCRS.

Type of experiment and experimental units: The experiment was performed using one factor and two treatments. We used the following notation:

- *vAdp*: Maps developed using the adaptive approach.
- *vPre*: Maps developed using the prescriptive approach.

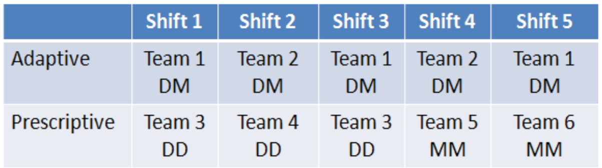
In this experiment, the factor was represented by developing maps and the treatments were prescriptive and adaptive approaches (Table 1).

TABLE I. Experimental Units Distribution

#Participant	*vAdp*	*vPre*
S1	X	
S2	X	
S3	X	
S4	X	
S5		X
S6		X
S7		X
S8		X
S9		X
S10		X
S11		X
S12		X

The participants' distribution to setup the teams was randomly defined between the factors adaptive and prescriptive. It was defined two designers and two makers for adaptive group and four designers and four makers for prescriptive group because the prescriptive group required two designers or two makers *per* shift.

The experiment was planned to have five shifts of 15 minutes each (representing one workday). For each approach (adaptive and prescriptive) the participants were organized in two sites where each site had two teams. We used FTS with no time-zone overlap between the distributed teams.



	Shift 1	Shift 2	Shift 3	Shift 4	Shift 5
Adaptive	Team 1 DM	Team 2 DM	Team 1 DM	Team 2 DM	Team 1 DM
Prescriptive	Team 3 DD	Team 4 DD	Team 3 DD	Team 5 MM	Team 6 MM

Figure 1. Distribution of each team

The members of each team could play the following roles: Map Designer (D) or/and Map Maker (M). As proposed by

Espinosa, Nan, and Carmel [12] these roles are similar to that of a software designer who needs to communicate the design specifications to a programmer. Each map designer (D) had a set of 13 maps while each map maker had a set of 13 blank PowerPoint© slides.

B. Instrumentation, Training & Execution

The experiment was executed in two isolated labs to avoid outside interference. Table II presents the instruments used to perform this experiment.

TABLE II. DETAILED INSTRUMENTATION

Type	Description	When	How	When
Experimental Unit	Prescriptive	Experiment	Maps Development	Execution
	Adaptive	Experiment	Maps Development	Execution
Document	Experiment guide	Experiment	Experiment preparation	Preparation
	Handoff form	Experiment	Collecting data from maps development	Execution
Guide	Training	Pre-experiment	To present the experiment context and motivation	Preparation
Metric	Report	Post-experiment	Data was gathered using a manual report.	Conclusion

To prepare the experiment execution, the following aspects were observed [11]:

- **Experiment consensus**: Participants were provided the required knowledge about the experiment, and clarification about the experiment goals;
- **Sensitive results**: Participant's names are anonymous throughout the experiment description.

All variables and resources were carefully defined before the experiment execution. We applied one treatment for each group (*Prescriptive* and *Adaptive*), contextualizing the objectives, motivation and technical procedure for the experiment.

Fig. 2 illustrates how the participants were distributed in the isolated labs.. Map designers (D) were in Lab1 and map makers (M) were in Lab2. A map designer's task was to provide instructions to the map makers about how to replicate the maps. Each map was composed by one background picture, five arrows and two extra icons. The joint of the arrows composing the path and icons in the background picture was executed by the map makers.

Teams that were working within the same shift (adaptive approach) had the possibility to use a communication tool. For this experiment we used GTalk [18]. Participants could "chat" with their teammates whenever they wanted. For the prescriptive approach the participants could only add instructions and comments in text log files from one shift to the next. Those text files were named as *handoff files.* GTalk logs helped us to rule out potential confounding effects of media richness [12]. The experiment execution had five shifts of 15 minutes each. Each team had 15 minutes to perform their tasks.

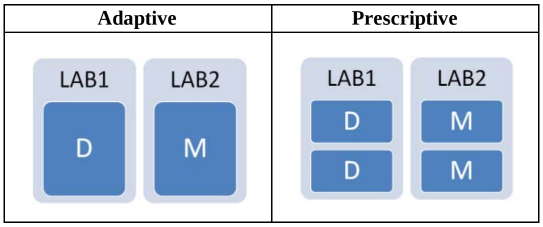


Figure 2. Labs division.

Fig. 3 p resents the map designer perspective. In the designer perspective, the idea of the task execution is to describe to makers a *path* composed by *five joint arrows* and the addition of *two extra icons* in a background picture. The designer must define the color (*blue*, *green* or *red*) of the arrows, the *start* and the *end* position of each arrow, as well as if these arrows are *solid* or *dashed*. Regarding the extra icons, the designer must precisely describe where these icons must be placed in the background picture. In the maker perspective, the successof a task depends on the accurate description done by the designer, and the correct interpretation done by the maker. The maker receives a set of steps that describe where the extra icons must be placed and the type and color of arrows, as well as where each arrow must be set in the figure in order to compose the path of a rrows. At the end, it was possible to create an experiment structure that could simulate adaptive and prescriptive software development using activities with fictional maps. During the experiment execution, researchers were available for clarification of questions from participants..

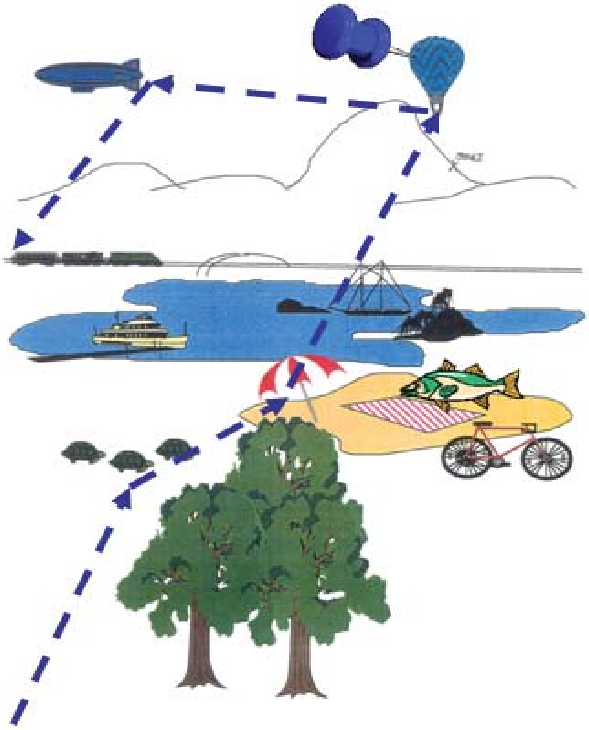


Figure 3. Map Designer perspective [11].

C. Dependent Variables and Measures

RQ1 checks the speed obtained by one team using an adaptive approach and another team using a prescriptive approach. The metric associated to RQ1 is calculated for each team dividing the amount of maps delivered (md) by the total number of maps (tm).

$$\text{Speed} = md / tm$$

RQ2 checks the accuracy of the maps delivered in each approach. The metric is calculated by dividing the amount of points from correct elements in each map (ce) by the number of total points (tP).

$$Accuracy = ce / tp$$

RQ3 verifies the quality the project delivered in each approach. The metric for quality is calculated by the sum of total accuracy points (ap) divided by the total number of maps delivered (md) by each team.

$$Quality = ap / md$$

To conduct the experiment we have formulated some hypotheses. We consider null and alternative hypothesis as following:

1. **The speed obtained in the adaptive and prescriptive approaches are the same**.

Null Hypothesis, H_0: Speed obtained by the team using the adaptive approach is the same as the speed obtained by a team using the prescriptive approach.

- **Metrics**: We calculated the speed as following:

SP – Represents the speed obtained by the team using a prescriptive approach.

SA – Represents the speed obtained by the team using an adaptive approach.

$$H_0\text{: SP = SA}$$

Alternative hypothesis, H_1: Teams using the adaptive approach are faster than teams using the prescriptive approach.

$$H_1\text{: SP < SA}$$

Alternative hypothesis, H_2: Teams using the prescriptive approach are faster than teams using the adaptive approach.

$$H_2\text{: SP > SA}$$

We have also collected information and tested similar hypotheses for accuracy and quality of the work delivered by the teams in each one of the approaches.

IV. Results

After five shifts, adaptive teams produced six out of thirteen maps and prescriptive teams produced four out of thirteen maps. Adaptive teams obtained 16% more speed than prescriptive teams. This result was obtained using the metric associated to RQ1:

Adaptive: 6/13 = 0.46

Prescriptive: 4/13 = 0.30

The following conditions were applied in order to check the accuracy obtained: Each map was composed by one background image plus two icons plus five arrows. It was the same composition used by Solingen and Valkema [8]. We measured the accuracy using the metric associated to RQ2. In addition, we also considered if map elements (arrows and icons) were in the right position. The maximum points per map are 12. Each accuracy criteria is described in details next:

- Background image = 1 point;
- Arrow color = 1 point;
- Arrow style = 1 point;
- First icon= 1 point;
- Second icon= 1 point;
- Position of each arrow = 1 point if the position match and 0.5 point when within the limit of 1 inch;
- Position of each icon= 1 point if the position match and 0.5 point when within the limit of 1 inch.

The results are described in Table 3. Seven maps were not produced (N/P) by adaptive teams and nine maps were not produced by prescriptive teams.

TABLE III. MAP ACCURACY IN DETAIL

	Adaptive	**Prescriptive**
Map 1	0.75	0.58
Map 2	0.63	**0.92**
Map 3	**0.96**	**0.96**
Map 4	**0.71**	N/E
Map 5	**0.54**	N/E
Map 6	**0.58**	N/E
Map 7	N/E	N/E
Map 8	N/E	**0.92**
Map 9	N/E	N/E
Map 10	N/E	N/E
Map 11	N/E	N/E
Map 12	N/E	N/E
Map 13	N/E	N/E
Total	**4.17**	**3.38**

N/E = Task not executed.

In Fig. 4, we present a comparison of the results presented in Table III regarding map accuracy.

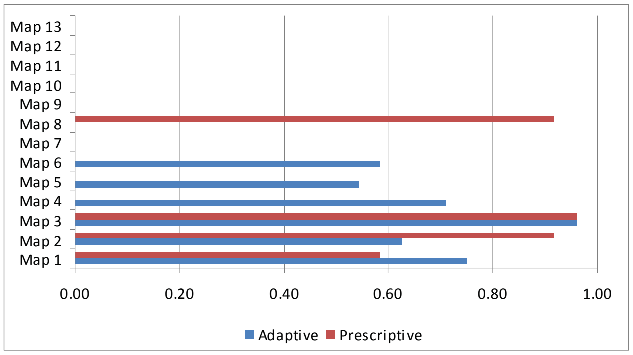


Figure 4. Map accuracy analysis.

The results indicate that teams using an adaptive approach produced more maps than teams using a prescriptive approach.

However, when we observed the total points of correct elements for each map, prescriptive teams have more accuracy. We also computed the quality metric, associated to RQ3:

Adaptive: 4.17/6 = 70%

Prescriptive: 3.38/4 = 85%

Our findings indicate that teams using adaptive approach had 15% less quality; in contrast, these teams had 16% more speed. This means that there is a trend showing that teams using adaptive approaches might have less quality but more speed. However, we did not have enough data points to support statistical analysis.

V. Discussion

FTS is a r esearch area with many important aspects to investigate [3] [4] [13]. In theory, the use of FTS can significantly reduce the duration of the software development lifecycle [14]. In this study, we investigated the difference and benefits of u sing prescriptive and adaptive approaches in the context of FTS.

The speed obtained by teams using an adaptive approach is higher than the speed obtained by teams using a p rescriptive approach. The percentage obtained by adaptive teams was 30% higher than the speed obtained by prescriptive teams. Phalnikar, Deshpande and Joshi [15] argue that agile teams are able to produce software faster, and Smite, Moe and Agerfalk [16] state that adaptive approaches such as agile methodologies aim to increase productivity of software teams [16]. Thus, adaptive approaches could benefit FTS by reducing the project cycle time. Moreover, FTS aims to reduce time-to-market [4], which it seems that can be obtained using adaptive approaches.

The accuracy obtained by teams in the delivered tasks was better in two tasks performed by prescriptive teams when we compare the first three tasks. On the other hand, when comparing the average accuracy of delivered tasks, adaptive teams are better than prescriptive teams. Teams using adaptive approaches are able to implement more requirements than teams using prescriptive approaches. This result shows that adaptive approaches may accelerate communication between teams and thus increasing the productivity, but we observed a lack of formal protocols to check the accuracy of the work.

The number of sites interacting in a development cycle may cause a small impact on the average accuracy [8]. We observed that the perception in relation to a task delivered is also a factor that affects the accuracy of the work. Our findings showed that students had less perception on task accuracy using adaptive approaches.

Quality is also an important factor for project success [5]. We observed this factor verifying the total accuracy points and the total number of maps delivered by each team. Results obtained showed that adaptive teams had 15% less quality than prescriptive teams. Carmel, Espinosa and Dubinsky [4] obtained similar result in their quasi-experiment. The decreasing of quality in maps delivered by teams using adaptive approaches could be caused by the lack of participant's perception in relation to the quality criteria required for each task.

Based on the results found, our experiment suggests that adaptive approaches could perform better than prescriptive approaches in the context of FTS. This is also claimed by Carmel, Espinosa, and Dubinsky [4] and Gupta [17], and should be deeply investigated in the future, with experiments also executed in real software development settings.

VI. Threats to Validity

One of the key issues in experimentation is evaluating the validity of the results [11]. In this section we discuss the potential threats that are relevant for our study and how they are addressed.

A. Construct Validity

Construct validity concerns the degree of accuracy to which the variables defined in the study measure the constructs of interests. In this study, we have followed the constructs defined in the two original FTS experiments [8] [12].

B. Internal Validity

Threats to internal validity influence the conclusions about a possible causal relationship between the treatment and the outcome of a study. We identified a couple of s uch kind of threats.

- *Skill to perform the task:* selected students had different skills and could potentially influence tasks performance. We randomly select students for each approach to minimize this threat.
- *Experiment unit:* we used an experimental condition different from a real scenario of s oftware development. Students' iteration included additional tasks, which are not usually done in a typical "real" iteration.
- *Measures:* our measures can be imperfect, since we simulate a full day work with only few minutes. We acknowledge that, but observe that any experiment would need to drastically reduce effort to represent a full day in an experimental setting.
- *The type of task:* participants manipulate maps and do not carry out real programming tasks. In this case, we refer to the original experiments where the authors indicate that they "decided in favor of a fi tional map task instead to eliminate possible confounds due to differences in the software programming abilities of the participants" [8] [12].

C. External Validity

External validity describes the study representativeness and the ability to generalize the results outside the scope of the study. For any academic laboratory experiment the ability to generalize results to industry practice is restricted by the usage of students as study participants. Although the students may not be representative of the entire population of s oftware professionals, it has been shown that the differences between students and real developers may not be as large as assumed by previous research [19].

D. Conclusion Validity

Conclusion validity is concerned with the relationship between the treatment and the outcome. We acknowledge that the small number of data points is not ideal from the statistical point of view. Small sample sizes, especially when the key experimental unit is at the team level, are a known problem difficult to overcome.

VII. Conclusion And Future Work

In this paper, we executed an experiment in order to investigate both the adaptive and prescriptive approaches in the context of FTS software development. We found that the usage of adaptive approaches increases the speed, but they do not always enhance accuracy and quality of the work done by distributed sites. We believe that this experiment has important findings that contribute to the literature on global software engineering and follow the sun software development.

For future work, we suggest new studies in order to replicate this experiment with more participants and different shifts scenarios (e.g. some time-zone overlap). We also suggest the replication of this experiment with industry participants.

Acknowledgment

The authors are funded by the PDTI program, funded by Dell Computers of Brazil Ltd. (Law 8.248/91). The third author is also funded by Ci&T and CNPq (projects 483125/2010-5 and 550130/2011-0. The authors thank all the students who participated in the experiment and Alberto Espinosa, Ning Nan and Erran Carmel for providing us the material used in their experiment.

References

[1] P. Sooraj, P. K. J. Mohapatra, "Modeling the 24-h software development process". Strategic Outsourcing: An International Journal, 122-141, 2008.

[2] E. Carmel, J. Espinosa, Y. Dubinsky, "Follow the Sun Workflow in Global Software Development," Journal of Management Information Systems Vol. 27 No. 1, 2010, 17 – 38.

[3] J. Kroll, E. Hess, J. L. N. Audy, and R. Prikladnicki, “Researching into Follow-the-Sun Software Development: Challenges and Opportunities,” In: 6th International conference on Global Software Engineering (ICGSE), 2011, Helsinki, Finland.

[4] E. Carmel, A. Espinosa, Y. Dubinsky, “Follow The Sun Software Development: New Perspectives, Conceptual Foundation, and Exploratory Field Study,” 42nd Hawaii International Conference on System Sciences, Proceedings, 2009.

[5] R. Jabangwe, I. Nurdiani, “Global Software Development Challenges and Mitigation Strategies: A Systematic Review and Survey Results”. Master´s program in Software Engineering, Blekinge Institute of Technology, OM/School of Computing, 2010.

[6] H. Holmstrom, E. O. Conchuir, P. J. Agerfalk, B. Fitzgerald, “Global Software Development Challenges: A Case Study on Temporal, Geographical and Socio- Cultural Distance”. Proceedings of the IEEE international conference on Global Software Engineering (ICGSE '06). IEEE Computer Society, Washington, DC, USA, 2006, 3-11.

[7] M. Yap, "Follow the sun: distributed extreme programming development," Agile Conference Proceedings, 2005, 218- 224.

[8] V. R. Solingen, M. Valkema, “The Impact of Number of Sites in a Follow the Sun Setting on the Actual and Perceived Working Speed and Accuracy: A Controlled Experiment”. Global Software Engineering (ICGSE), 5th IEEE International Conference,165- 174, 2010.

[9] C. Visser, R. V. Solingen, “Selecting Locations for Follow-the-Sun Software Development: Towards A Routing Model”. Fourth IEEE International Conference on Global Software Engineering, 2009.

[10] E. Hess, and J. L. N. Audy, “FTSProc: a Process to Alleviate the Challenges of Projects that Use the Follow-the-Sun Strategy,” In: 7th International conference on Global Software Engineering (ICGSE), 2012, Porto Alegre, Brazil, in press.

[11] C. Wohlin, “Experimentation in Software Engineering: An Introduction”, International Series in Software Engineering, Kluwer Print, 2000.

[12] J. A. Espinosa, N. Nan, E. Carmel, "Do Gradations of Time Zone Separation Make a Difference in Performance? A First Laboratory Study," Global Software Engineering, 2007. ICGSE 2007. Second IEEE International Conference on , pp.12-22, 27-30 Aug. 2007.

[13] J. Kroll and J. L. N. Audy "Mapping Global Software Development Practices to Follow-the-Sun Process," In: 7th International conference on Global Software Engineering (ICGSE), 2012, Porto Alegre, Brazil., in press

[14] R. M. Czekster, P. Fernandes, R. Prikladnicki, A. Sales, A. R. Santos, and T. Webber "Follow-The-Sun Methodology in a Stochastic Modeling Perspective". In 6th IEEE International Conference on Global Software Engineering (ICGSE): Methods and Tools for Project/ Architecture/Risk Management in Globally Distributed Software Development Projects (PARIS), pages 54–59, Helsinki, Finland, August 2011.

[15] R. Phalnikar, V. S. Deshpande, S. D. Joshi, "Applying Agile Principles for Distributed Software Development," Advanced Computer Control, 2009. ICACC '09. International Conference on, pp.535-539, 22-24 Jan. 2009.

[16] D. Šmite, N. B. .Moe, P. J.Ågerfalk, Agility Across Time and Space: Implementing Agile Methods in Global Software Projects. 1st Edition., 2010, XXXVI, 341 p. 37 illus.

[17] A. Gupta, “Deriving mutual benefits from offshore outsourcing,” Communications of the ACM, v.52 n.6, 2009.

[18] GTalk. Google talk. Available at http://www.google.com/talk/.

[19] M. Höst, B. Regnell, B. and C. Wohlin. “Using Students as Subjects - A Comparative Study of Students and Professionals in Lead-Time Impact Assessment.” Empirical Software Engineering, Vol. 5, No. 3, 2000, pp. 201-214.

Software Process Monitoring using Statistical Process Control Integrated in Workflow Systems

Marília Aranha Freire, Daniel Alencar da Costa, Eduardo Aranha, Uirá Kulesza
Programa de Pós-Graduação em Sistemas e Computação
Departamento de Informática e Matemática Aplicada
Universidade Federal do Rio Grande do Norte
Campus Universitário, Lagoa Nova – 59.078-970 – Natal, RN – Brazil
{marilia.freire, danielcosta}@ppgsc.ufrn.br,{uira, eduardo}@dimap.ufrn.br

Abstract— **This paper presents an approach that integrates statistical process control techniques with workflow systems in order to achieve software process monitoring. Our approach allows: (i) software process monitoring through the automated metrics collection; and (ii) the statistical process control of software process aided transparently by statistical tools. The use of workflow systems to this integration adds the benefits of statistical process control without the additional effort to integrate and use statistical tools. Our proposal allows project managers to identify problems early during the process execution, enabling quickly reactions (process improvements, training, etc.) to reduce costs and ensure software quality.**

Keywords: Software Process Monitoring, Statistical Process Control, Workflow Systems

I. INTRODUCTION

The increasing complexity of modern software systems has required more well-defined software development processes utilization. Software Process Modeling Languages – SPML support the definition and modeling of software processes, providing functionalities to create and edit the activities flow of their various disciplines, addressing the elements that define a software process, such as tasks, steps, artifacts, and roles [1] [2] [3]. In addition to the benefits and advantages brought by process modeling languages, several recent studies have emphasized the importance of providing mechanisms and tools to support the execution of software processes in order to enable the tracking and monitoring of their activities. Monitoring software projects is important to assess productivity and detect problems that may be occurring and thus promote continuous process improvement.

One approach to support software processes execution is the use of workflow systems. These kinds of systems have been consolidated over the past few years on the business process management domain. The Business Process Execution Language (BPEL), for example, is one of the main industrial results developed by this community. Some recent studies have promoted the integration of approaches and languages for processes modeling and execution [4] [5] [6].

Process control and monitoring is a concept that has been explored and adopted in the industrial scenario. Some decades ago, emerged a statistical technique called Statistical Process Control (SPC) [7] that aims to monitor and quickly detect problems in process execution, allowing fast corrective responses, increasing the quality and productivity of the production processes. This technique has been widely used in industry in general, and its concepts are already being employed in the software industry over the past years [8] [9] [10]. When using this technique, upper and lower control bounds are established for relevant attributes of production processes (time, cost, output quality, etc.), usually based on historical data. Then, if attribute values collected during the process execution are out of the range, these values are called as outliers and they are highlighted for investigation, since they may be caused by problems occurred during the process execution.

This paper presents an approach for integrating statistical process control techniques and workflow systems for monitoring the execution of software processes. Our approach supports: (i) the monitoring of process execution through an automated support for metrics collection and (ii) the statistical process control deployed in workflow systems. As benefits, the approach promotes the monitoring of process stability – ability to be predictable, and process capability – ability to meet specifications, as well as quick responses to outliers, supporting the analysis and decision-making to continuous software process improvement.

The remainder of this paper is organized as follows. Section 2 presents the foundations of process monitoring and statistical process control. Section 3 presents an overview of our approach, which an implementation is presented in Section 4. Section 5 details the approach while illustrating its application and section 6 describes the related works. Finally, Section 7 concludes the paper and provides some directions for future work.

II. BACKGROUND

A. Software Process Monitoring

The automated support for the software development process definition is a concrete reality today. Several approaches have been proposed to facilitate not only the process definition, but also to provide better ways to specify software processes customizations [2] [1] [3]. They provide a set of tools, formalisms and mechanisms used for modeling processes together or even specialize them. Moreover, others research approaches have being proposed, such as DiNitto [11], PROMENADE [12], Chou [13] and UML4SPM [14].

While methodologies, tools and techniques for software processes definition are already consolidated, the environments supporting such software processes execution are still in the process of ripening. The integration of techniques for software processes definition, execution and monitoring has emerged as a way to support the automatic process monitoring, allowing the estimation of activities and evaluation of team productivity, quality control and process management, which eventually contribute to continuous software process improvement. Software process monitoring is a complex activity that requires the definition of metrics to be collected during execution. The metrics collected at runtime can help the manager during the analysis of the project progress, facilitating the decision-making.

Freire et al [15] presents an approach for software processes execution and monitoring. In that approach, software processes are specified using the Eclipse Process Framework (EPF), which can be automatically transformed into specifications written in the jPDL workflow language [16]. These specifications in jPDL can then be instantiated and executed in the jBPM workflow engine [17]. In addition to supporting the automatic mapping of EPF process elements in workflow elements, the approach also: (i) supports the automatic weaving of metrics collection actions within the process model elements, which are subsequently refined to actions and events in the workflow; and (ii) refines the workflow specification to generate customized Java Server Faces (JSF) web pages, which are u sed during workflow execution to collect important information about the current state of the software process execution.

Such an approach has been implemented using existing model-driven technologies. QVTO and Acceleo languages were used to support model-to-model and model-to-text transformations, respectively. The approach proposed in this paper is developed based on the work presented in [15].

B. Statistical Process Control

The Statistical Process Control (SPC) is a set of strategies to monitor processes through statistical analysis of the variability of attributes that can be observed during the process execution [18] [10] [19] [20]. In terms of SPC, the sources of variations in the process are encompassed in two types: (i) common source of variation and (ii) special source of variation.

The difference between the common and special source of variation is that the former always arises, as a part of the process, while the later is a cause that arises due to special circumstances that are not linked to the process. For example, a common variation on development productivity could be caused by differences in programming experience between developers. On the other hand, a special variation could be caused by a lack of training in a new technology. As special sources of variation are usually unknown, their detection and elimination are important to keep the quality and productivity of the process.

Control charts, also known as *Shewhart charts*, are the most common tools in SPC used to monitor the process and to detect variations (outliers) that may occur due to a special source of variation. The use of control charts can classify variations due to common or to special causes, allowing the manager to focus on variations from special causes. The control chart usually has thresholds at which a metric of the process is considered as an outlier. Those thresholds are called *Upper Control Limit* (UCL) and *Lower Control Limit* (LCL). One pair of UCL and LCL are defined based on the statistical analysis of historical data or based on expert opinion, being used to identify the outliers. However, other pairs can be defined to highlight, for instance, limits that should be respected due to client requirements, such as process productivity (function points implemented per week, etc.) or quality (number of escaped defects, etc.).

Despite the known benefits of using SPC to monitor software processes in order to detect problem during process execution, this task in practice is still very arduous. The software project manager needs to know not only the statistical foundation, but also understand and manipulate statistical tools for the generation of graphics and information necessary for monitoring the processes. To reduce these problems in using SPC, the approach suggested here minimizes the work of the project manager by transparently integrating the use of statistical tools for monitoring the process execution in a workflow system. Furthermore, this automatic control enables continuous recalibration of the control limits, according to the changes occurring in the process performance, and ensures the correct use of statistical techniques.

III. SOFTWARE PROCESSES MONITORING USING SPC INTEGRATED IN WORKFLOW SYSTEMS

A. Approach Overview

The approach proposed in this paper promotes the monitoring of software processes integrating workflow systems and S PC. This integration promotes the collection and analysis of metrics quickly and automatically, simplifies the use of the s tatistical process control t echnique and enables the automatic recalibration of the control limits used. This section is organized in six steps, as shown in Figure 1 and detailed next..

1) Process Modelling and Definition

The first approach step is directly related to the software process definition. At this stage one should use a process modeling language (SPL) to specify the process to be monitored. As described in Section IV, the current implementation of our approach provides support to the process definition using the EPF framework. EPF offers features and functionalities for the process definition and modeling through the us e of the UMA process modeling language (Unified Method Architecture) [21], which is a variant of the SPEM (Software Process Engineering Meta-Model) [22]. Existing process frameworks such as OpenUP [23] (available in the EPF repository) can be reused and customized to define new software process, reducing the costs of this activity.

2) Metrics Modelling and Definition

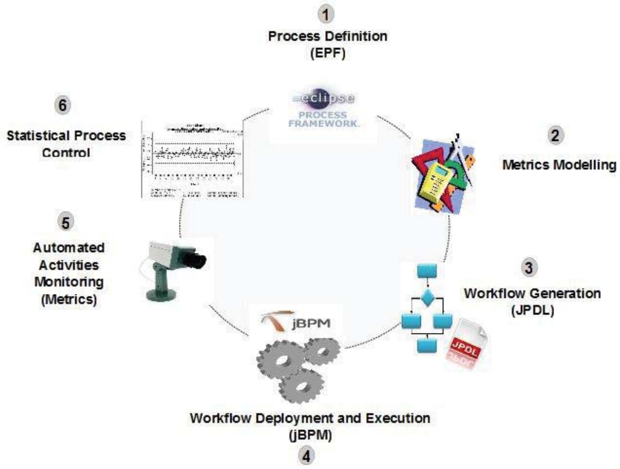


Figure 1 Approach Overview

After the process modeling and definition, it is necessary that process engineers specify the metrics to be collected and monitored during the process execution. Each metric must be defined and associated with one or more activities of the monitored process. The definition of this association is accomplished by specifying the activities that produces each metric, which are modeled using the following meta-model [15].

3) Workflow Generation

To enable the process execution in a workflow system, supporting the automatic collection of defined metrics, a model-to-model (M2M) transformation is performed to generate the JPDL workflow elements from EPF process elements. This transformation is responsible for the generation of actions that allows the automated collection of metrics during the execution of the process activities in the workflow system. In addition, the transformation also generates web pages that will be used for interaction with the process users.

4) Workflow.Deployment and Execution

After generating the workflow and other configuration files, the jBPM workflow engine is used to support the software process execution. It allows project managers to visualize the process execution in real-time and be aware of what is happening during the project development in order to take decisions. They can know, for instance, what activity each member of the project is performing, as well as the status of project activities (performance, quality, etc.) based on the collected metrics.

5) Automatic Activities Monitoring

Monitoring software processes in an automated manner allows greater control of the process by the project manager. This approach, as presented in [15], allows the project manager to automatically monitor the project's progress by viewing web pages and exploring information about the previously defined metrics. These pages provide status information of the process and also the values of the metrics collected dynamically. During the workflow execution, at the end of each task defined in the metrics model, its duration is calculated by performing an action fired after an *end-task* event, and this value is stored and displayed to the project manager.

The project manager can use the collected data to support continuous process improvement and contingency actions, avoiding the occurrence of future problems. Examples of information that can be provided by such metrics are: the execution time of each task step; which task step has a longer duration in the timeline; what is the estimation accuracy; quality or productivity benchmarks, such as function point or use case point per man-hour.

6) Statistic Process Control in Workflow Systems

Our approach promotes the integration of statistical process control into workflow systems. To enable automatic monitoring using SPC, at the end of each monitored task in the workflow, the new value obtained is compared to the last ones in order to determine if it is within the expected range defined for the statistical control. In other words, the observed value for the metric is compared to the LCL and UCL values. If the observed value is lower than the LCL or greater than the UCL, a warning message is issued for the project manager to analyze the cause of this outlier (values significantly different from the expected).

To calculate and implement the control limits, one possible way is the following (other procedures can also be implemented):

(a) If there are not historical data, expertise can be used to set limits on changes expected for each metric;

(b) If there are historical data, calculates the range as follows:

(i) If the data distribution follows the normal distribution (as verified by statistical tool integrated with monitoring), uses a number of standard deviations, which by default is 3 (includes approximately 99.7% of the population data) and that can be changed by the user to increase or decrease the range. Increasing the range is meant to include more extreme values that could be the problem and will not generate warning. On the other hand, increasing the range reduces the amount of false-positive (indicating problems that are not a problem);

(ii) if the system does not identify the normal distribution, uses the Chebyshev's theorem to calculate the number of standard deviations to cover the same 99.7% of the population data;

In both cases (i) and (ii), the user can make adjustments to the number of standard deviation to be considered. Based on that, the tool indicates the percentage of data encompassed (expected/normal values) according to data distribution observed. The user can also indicate a percentage of interest and the number of standard deviations that should be used will be calculated by the tool.

To facilitate the monitoring and visualization of attributes being monitored, an *X chart* is updated on the screen of the project manager after each new collected value. The outliers are shown in red color in the graphic, representing a possible anomaly.

After the collection of new values, they become part of the historical basis of the process, contributing to the

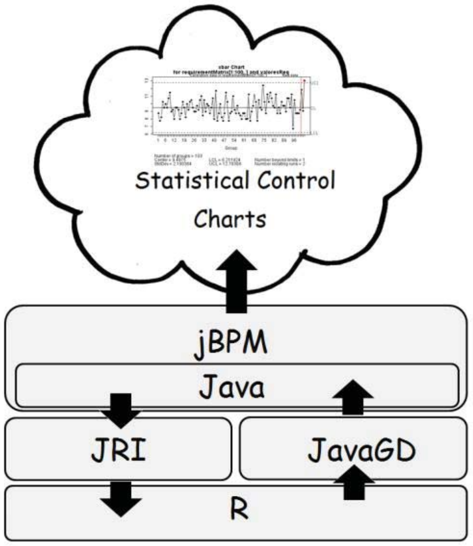


Figure 2 Approach Implementation

adjustment of LCL and UCL values, the known dynamically tuned monitoring sensibility promoted by SPC. Outliers representing problems occurred in the process are not considered for this adjustment, since other occurrences should also be detected.

IV. Approach Implementation

The implementation of our approach was accomplished through the integration of the JBPM workflow engine with the computational statistics tool R [24]. This integration is implemented with the *API Java/R Interface* (JRI) [25] that enables *R* function calls within Java code, which is the language used by JBPM. Figure 2 illustrates the approach implementation.

During the M2M and M2T transformations, events and actions handlers are generated and they are responsible for collecting data during workflow execution according to the metrics defined in the model. These action handlers call the R statistical functions to build process control charts (Shewhart, Cusum etc.). However, these functions return specific R charts implementations that need to be treated in the Java code in order to be displayed by jBPM. To enable the Java interpretation of these graphics, an R library called *Java Graphics Device* (JavaGD) [26] is used. This library provides Java canvas objects equivalent to the graphics produced by *R*. Once the canvas objects are obtained, they can be treated and transferred to a view framework such as the *JavaServer Faces* (JSF) [27] used by jBPM.

V. Approach in Action

To illustrate the approach proposed in this paper, we present the modeling of a software process and its metrics according to the approach depicted in [15]. The following subsections will describe the approach in action following its respective steps presented in Figure 1.

1) *Process and Metrics Modelling (Steps 1 and 2)*

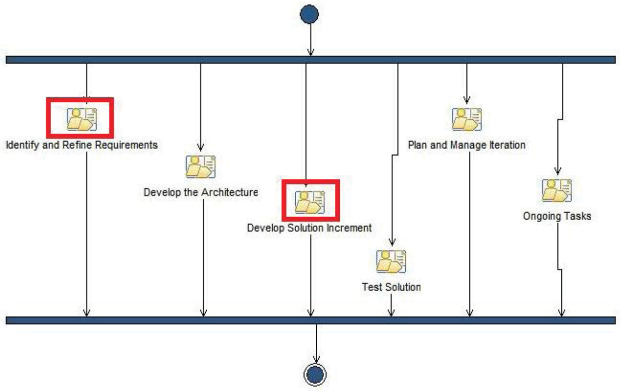
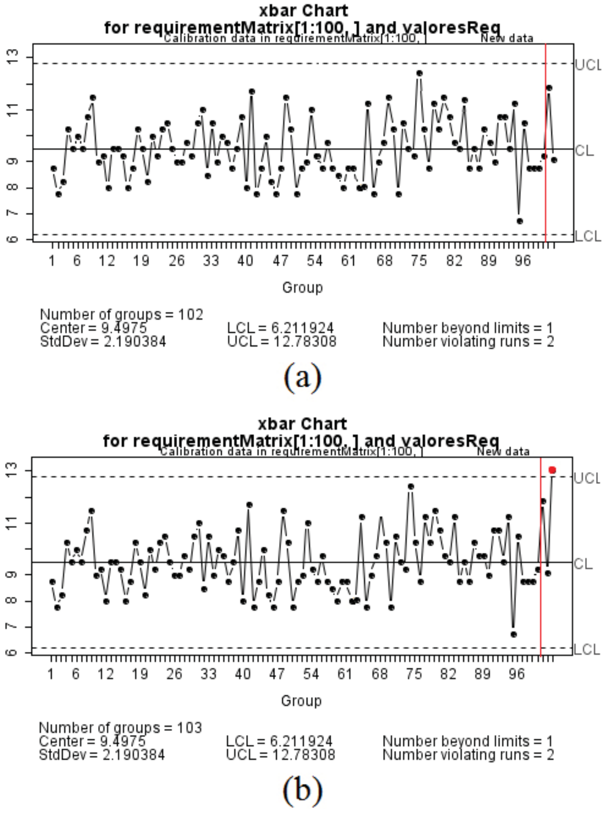

Identify and Refine Requirements
Develop the Architecture
Develop Solution Increment
Test Solution
Plan and Manage Iteration
Ongoing Tasks

Figure 3: Process Fragment Example

The process modeled to illustrate the approach is an OpenUP based process and it is presented in Figure 3. The metrics were modeled to monitor the highlighted activities *identify and refine requirements* and *develop solution increment*, aiming collecting the time spent in each activity.

2) *Workflow Generation and Execution (Steps 3 and 4)*

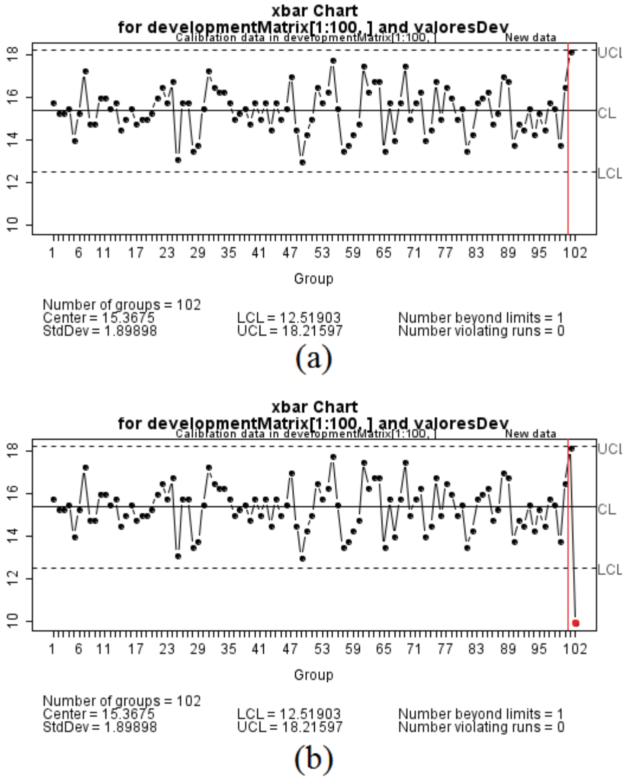
Once the two model transformations were held and the workflow was deployed in the jBPM engine, the workflow may be calibrated with historical organizational information regarding the metrics before starts the process execution. For example, if the metric is about implementation, then the calibration information would be the time developers take to implement simple or complex functionalities. Also, the limits must be specified and may attend the project requirements of quality or productivity. This is an important step as the approach intends to alert deviations along the process execution and needs to know if a collected metric value is a



(a)

(b)

Figure 4: Requirement Elicitation Metric Collection



(a)

(b)

Figure 5: Development Time Metric Collection

deviation indeed.

3) Automatic Monitoring and Statistical Process Control (Steps 5 and 6)

At this stage, the *X chart* is generated and the value of the attribute is graphed on the project manager screen at the end of each monitored activities instances. In the graphic, the x-axis represents the activities instances and the y-axis represents the attribute values collected. Figure 4(a) depicts the first collection of the *time spent per use case* metric after the calibration step. Note that the new collected value fits in the *Upper Control Limit* (about 6 days) and the *Lower Control Limit* (about 13 days). One can also include new limits to represent specific user quality requirement. During the process execution, the limits (UCL and LCL) can be recalculated including the value of the last execution to reflect adjustments made possible in the process at runtime. Figure 4(b) illustrates a case where the collected value supersedes the *upper control limit*. This fact could be explained, for example, as a case when the development company is eliciting requirements to a new business that was not explored in previous projects. In that case, the workflow can trigger a warning notification to interested stakeholders (e.g. an e-mail to a project manager) in order to help planning scope, resources or deadline changes and avoid unwanted situations such as iteration or deployment delays.

Figure 5(a) and Figure 5(b) depict the collect values for the *implementation time per use case* metric. In contrast to Figure 4(b), Figure 5(b) shows one case that the value is beyond the *lower control limit*. This could happen, for example, because of the development of a new functionality that is very simple compared to the functionalities previously developed (e.g. the implementation of a simple CRUD or a simple login functionality), or a case when the developer did not perform other related fundamental activities like testing or documentation. In some cases the metric value is just suffering a natural change and the control limits may be adjusted accordingly. For example, the chart is often graphing values outside the LCL when the development time metric is collected. Those occurrences may not be just outliers but they may happen because of a training course the development team received about technologies being used and so the time spent decreased considerably. In those cases, the project manager may decide if the new values collected may represent a new trend in the process and adjust the limits to handle accordingly. Figure 6 shows an example of limits adjustment in which the LCL and UCL were updated to handle the new values of the metric.

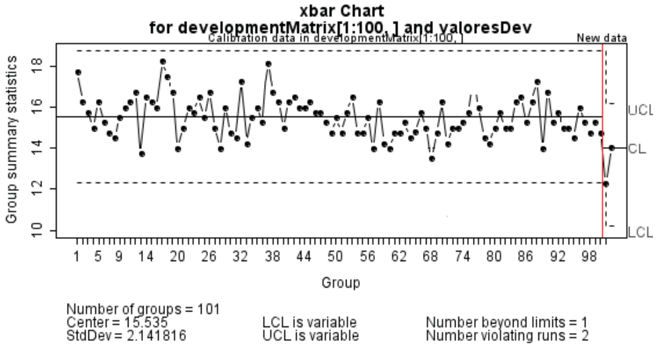


Figure 6: Development production shift

The current implementation presented in this work supports only the *X charts*, but the integration between other control charts (e.g. CUSUM) and workflows is also possible.

VI. Related Work

Several studies have been proposing and discussing the use of SPC in software process management to promote continuous improvement. Baldassarre et al [18] discuss the use of SPC from the results found after empirically use this technique in the industry. The paper discusses four synthesized major problems encountered in the software process monitoring showing how SPC can answer each one. It contributes for guiding practitioners towards a more systematic adoption of SPC. Komuro [20] describes experiences of applying SPC techniques to software development processes showing several real examples. The paper points out issues that need to be addressed in order to apply SPC and shows that the key for the successful process improvement is the alignment with business goal.

However, these related works mainly emphasize how to adapt SPC to control software projects and also point out its advantages and disadvantages. None of them focuses on the automated support for monitoring of software processes. Our approach provides support to the automatic and statistical monitoring of software processes in w orkflow systems through the generation and customization of software processes in workflow systems. The automatic monitoring using SPC transparently during the execution of the process in workflow systems contributes directly to m inimizing the complexity issues traditionally involved in work with statistical tools in software projects.

VII. Conclusion

In this paper, we have proposed an approach that integrates statistical process control with workflow systems to monitor software processes. The use of workflow systems to our integration promotes the collection and analy sis of metrics quickly and automatically, adds the benefits of statistical process control without the additional effort to integrate and use statistical tools, and enables the automatic recalibration of the control limits used. Our proposal allows project managers to identify problems early during the process execution, enabling quickly reactions (process improvements, training, etc) to reduce costs and ensure software quality, in other words, allows the fast monitoring. In addition, it also reduces the effort to use automatic monitoring and SPC in an integrated manner.

Currently, our model-driven framework is being increased and adapted to also support process monitoring of software engineering experimental studies. The process monitoring in this domain is fundamental to identify problems that could invalidate all the collected data and study conclusions. If the problem is i dentified early, actions can be performed to correct the problem, avoiding the loss of all data and giving to the software researcher a chance to better understand the software engineering technique, method or process under investigation.

Acknowledgment

This work was partially supported by the National Institute of Science and Technology for Software Engineering (INES), funded by CNPq, grants 573964/2008-4 and PDI - grandes desafios, 560256/2010-8, and by FAPERN.

References

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

[2] Eclipse Foundation. (2009) Eclipse. [Online]. http://www.eclipse.org/epf/

[3] IBM. Rational solution for Collaborative Lifecycle Management. [Online]. https://jazz.net/projects/rational-team-concert/

[4] R. Bendraou, J.M. Jezequel, and F. Fleurey, "Achieving process modeling and execution through the combination of aspect and model-driven engineering approaches," in *J. of Softw. Maintenance and Evolution: Research & Practice Preprint*., 2010.

[5] R. Bendraou, J.M. Jezequel, and F. Fleurey, "Combining Aspect and Model-Driven Engineering Approaches for Software Process Modeling and Execution," in *Proc.Intl. Conf. on Softw. Process*. Vancouver, Canada, 2009, pp. LNCS, vol. 5543, pp. 148-160.

[6] 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.

[7] W.A. Shewhart, *Statistical Method from the Viewpoint of Quality Control*. Mineola, New York: Dover Publications, 1986.

[8] J C Benneyan, R C Lloyd, and P E Plsek. (2012, Feb.) Statistical process control as a tool for research and. [Online]. http://qualitysafity.bmj.com

[9] F ZORRIASSATINE and J. D. T. TANNOCK, "A review of neural networks for statistical," *Journal of Intelligent Manufacturing*, pp. 209-224, 1998.

[10] Monalessa Perini Barcellos, Ana Regina Rocha, and Ricardo de Almeida Falbo, "Evaluating the Suitability of a Measurement Repository for Statistical Process Control," *International Symposium on Empirical Software Engineering and Measurement* , 2010.

[11] E. Di Nitto, A. Fuggetta G. Cugola, "The JEDI event-based infrastructure and its application to the development of the OPSS WFMS," *IEEE Trans. Softw. Eng.*, vol. 27, pp. 827-850, 2001.

[12] X. Franch and J. Rib, *A Structured Approach to Software Process Modelling*.: in Proceedings of the 24th Conference on EUROMICRO - Volume 2, 1998, pp. 753-762.

[13] S.-C. Chou, *A process modeling language consisting of high level UML diagrams and low level process language*.: Journal of Object-Oriented Programming, vol. 1, no. 4, pp. 137-163, 2002.

[14] R. Bendraou, M-P. Gervais, and X. Blanc, "UML4SPM: An Executable Software Process Modeling Language Providing High-Level Abstractions," *10th IEEE EDOC*, pp. 297-306, 2006.

[15] Marília Freire, Fellipe Aleixo, Kulezsa Uira, Eduardo Aranha, and Roberta Coelho, "Automatic Deployment and Monitoring of Software Processes: A Model-Driven Approach," in *SEKE*, 2011.

[16] JBOSS. jBPM Process Definition Language (JPDL). [Online]. http://docs.jboss.org/jbpm/v3/userguide/jpdl.html

[17] JBOSS. JBoss jBPM. [Online]. http://www.jboss.org/jbossjbpm/

[18] Maria Baldassarre, Nicola Boffoli, Giovanni Bruno, and Danilo Caivano, "What Statistical Process Control can really do for Software Process Monitoring: lessons from the trench," in *Trustworthy Software Development Processes*.: Springer Berlin / Heidelberg, 2009, pp. 11-23.

[19] Nicola Boffolli, G. Bruno, D. Caivano, and G Mastelloni, "Statistical process control for software: a systematic approach.," in *ESEM*, 2008, pp. 327-329.

[20] Mutsumi Komuro, "Experiences of applying SPC techniques to software development processes,"*28th ICSE*, New York, NY, USA, 2006, pp. 577-584.

[21] Eclipse. Eclipse EPF Project. [Online]. http://epf.eclipse.org/wikis/openupsp/base_concepts/guidances/concepts/introduction_to_uma,_94_eoO8LEdmKSqa_gSYthg.html

[22] OMG. Software Process Engineering Meta-Model. [Online]. http://www.omg.org/technology/documents/formal/spem.htm

[23] IBM Corp. (2009) OpenUP Process Version 1.5.0.4. [Online]. http://epf.eclipse.org/wikis/openup/

[24] Wien, Institute for Statistics and Mathematics of the WU. (2012, Fevereiro) R project. [Online]. http://www.r-project.org/

[25] RForge.net. (2012, Fevereiro) JRI - Java/R Interface. [Online]. http://www.rforge.net/JRI/

[26] S. Urbanek. (2012, Fevereiro) JavaGD. [Online]. http://rosuda.org/R/JavaGD/

[27] Java Community. (2012, Fevereiro) JavaServer Faces. [Online]. http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html

[28] Weller, E.F.; Bull HN Inf. Syst., Phoenix, AZ , "Practical applications of statistical process control [in software development projects] ," *Software, IEEE*, vol. 17, pp. 48-55, May/Jun 2000.

[29] Eclipse. Acceleo. [Online]. http://wiki.eclipse.org/Acceleo

[30] Geppert A, Tombros D, "Event-based distributed workflow execution with EVE," *Middleware'98 Workshop*, 1998.

Model Transformation for Frameworks using Logical Planning

Guilherme A. Marchetti, Edson S. Gomi
Department of Computer Engineering,
School of Engineering, University of São Paulo, São Paulo, Brazil.
Email : {guilherme.marchetti, gomi}@usp.br

***Abstract*—Frameworks are an important tool for current software development methods. However, due to frameworks complexity, the time required to learn how to use them increases. In order to help developers use new frameworks, a model to model transformation method named Model Transformation for Frameworks is developed. It is based on a logical planner that automatically identifies components from the target framework that will be added to the target application model.**

I. Introduction

Reducing cost and time to develop systems is one of the goals of software engineering. One important concept in software engineering that helps reduce them is reusability. By reusing elements, either code or models, it is possible to reduce the cost and time for implementing solutions.

One of the most used reuse technique is the framework. Frameworks are a reusable set of components, forming an incomplete application, which can be specialized to create new solutions [1], usually targeted to specific domains, such as user interfaces or smartphone software development. But despite being a useful approach, the use of frameworks also incurs costs [2]. One of the costs associated with a framework is the time required to learn it before being able to use it.

Also trying to reduce development costs, in 2001 the Object Management Group (OMG) launched the Model Driven Architecture (MDA) initiative. The main goal of this initiative is to obtain an executable code from high level models of the system, through various transformations.

These transformations are called Model Transformations, and can be divided into two groups [3]: model-to-model and model-to-code. The first group, model-to-model transformations, contains techniques that range from refining a model through the different abstraction layers used during the specification of a system[4], to the transformation between different meta-models [5], for example transforming models described in UML to Entity-Relationship models for databases. The second group, model-to-code transformations, focuses mainly on ways to generate executable codes from models. For example, in this group, techniques to create JAVA code from Collaboration Diagrams [6] or from State-Machine Diagrams [7] can be found.

But, even with all the existing model transformation techniques, the use of a framework still requires a programmer to learn which services it provides, and how to use them.

In this paper we present a model-to-model transformation method that identifies which components from a desired framework should be used to meet the application requirements. Our transformation method requires a framework model as one of the inputs. This model should describe the abstract services it provides. These services are described through class and sequence diagrams. The second input required is the application model containing the description of the desired behavior. These models are used by a logical planner that identifies which methods and classes from the framework should be used. Once the required framework components are identified, all the sequence diagrams necessary to describe the use of the framework are built and all the framework classes used are added to the application class diagram.

To create all the sequence diagrams that describe framework services, and to choose or to create all the necessary classes, we use a logical planner, based on the Strips [8] and NONLIN [9] approaches.

This paper is organized as follows: in section II we describe how a framework should be modeled in order to make it suitable to be converted to a logical representation. Section III shows how to create the logical representation of the framework. In section IV, we show how the logical planner creates service sequence diagrams. In section V, we show how our method compares with similar works and our conclusion is presented in Section VI.

II. Modeling the Framework

In this section, we show how a framework is modeled so that it can be transformed into a logical representation. The framework model must contain at least a Class Diagram containing interfaces (or services) it provides to the user. Also, the model may contain one or more Sequence Diagrams representing the methods that will be implemented by the user in order to access the framework services.

If an interface has no method description, we assume that the interface is executed whenever an object of this type is created. Besides the services, the class diagram must also show all the methods the user can access. These methods are identified as constructors or static. Using the method signature, it is possible to identify the types of the input argument, as well as the return type, if any, of the method. Regarding the names given to the relationships of the classes, we assume that if one of the input objects or the return object of the method has the same name of one of the classes attribute or relationship, then it is the same object and, therefore, a relationship exists between these two objects.

The Sequence Diagrams are used to describe any internal behavior of the framework, also called mechanisms. These diagrams should describe any service of the framework that requires the use of callbacks. The callback function that must be provided by the user, either by implementing an interface or overwriting an existing method, is identified by an empty behavior description in the diagram.

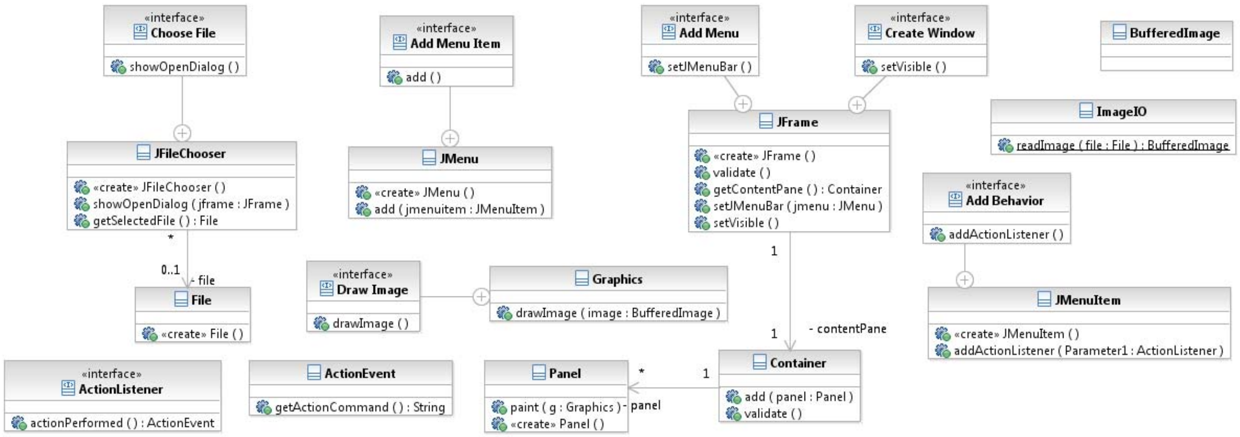


Fig. 1: Swing Framework Class Diagram

As an example, we present the use of JAVA SWING. This framework is used to create user interfaces for JAVA programs. A simplified model of the framework presented in Figure 1 (Class Diagram), and in Figure 2 (Sequence Diagram).

In the class diagram, it is possible to identify several services, such as “Create Window” and “Choose File”. Each service is associated with a class, enabling the identification of which class provides each interface. The services descriptions also describe which method from the class should be invoked in order to use that service. For example, the interface “Create Window” is provided by the class “JFrame” through the method “setVisible”. In this diagram, it is also possible to identify constructors methods, through the label “create” (such as the method “JFrame”, from the “JFrame” class), and the static methods (represented as underlined methods), such as the method “readImage” from the class “ImageIO”.

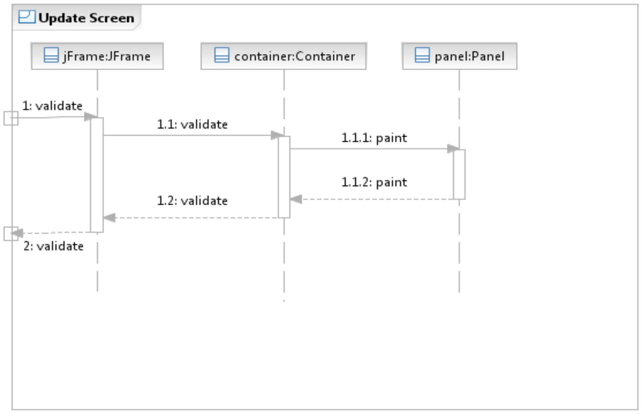


Fig. 2: Swing Framework Sequence Diagram

The Sequence Diagram of figure 2 contains the methods associated with updating a screen. To initiate the update process, the user has to call the method “validate” from the “JFrame” class, which will, in turn, call another method also called “validate”, but from the “Container” class, to finally call the method “paint” from the “Panel” class. Using the information in this diagram, a user could draw elements on a Swing window by specializing the “Panel” class and overwriting its “paint” method. After this, he must make sure this new object is reachable by a “JFrame” object and call the “validate” method of the object. The reason for going through these steps is to obtain a “Graphics”-type object, and to access its methods, since this class has no constructor visible to the user; thus the only way to obtain an object of this type is within the “paint” method from the “Panel” class, and letting the framework construct the object.

III. Logical Representation of the Framework

In order to use a logical planner, the framework model must be represented using logical propositions. A logical proposition describes a fact about the state of the world being worked on. It is composed of a predicate and its variables, in which the predicate represents a “fact” and the variable indicates in which element(s) the “fact” is true. In the case presented herein, the world represents which elements of the framework exist and are being used at a given moment. To describe a given state, the following 5 predicates are used:

- object(Type) : represents the existence of an object of Type;
- interface(Interface) : represents that the Interface has been executed;
- sequence(Name) : represents that the sequence with Name was initiated;
- link(Class1, Class2) : represents the existence of a link between an object from Class1 and an object from Class2;
- specialization(Class) : represents the existence of a Class specialization

These predicates represent the state of the world. For the world to change between states, it must be possible to execute actions on it. Since it is through the use of methods that a software state is changed, each action, also called operator, will represent a framework method. Each one of these operators has the form:

```
Name :
 PreCondition :
 Effect:
```

where Name is the name of the operator, PreCondition is the set of propositions that must be true in the current state so that the operator is usable, and Effect is the set of propositions that will become true after the execution of the operator.

It is now possible to translate the information contained in the framework model into a set of operators. The framework model has two types of diagrams: the Class Diagram and Sequence Diagrams. Each of these is translated in a different way, since they contain different types of information, even though some information is shared between them. We will begin with the class diagram translation.

In the class diagram, it is possible to identify all the methods the framework makes available to the user. For each of these methods, a different operator will be made. The name of the operator will contain the name of the method being used and, in order to allow methods with the same name in different classes, the operator name will also contain the class name. The resulting operator name will then be "ClassName.MethodName".

Following the name, the set of preconditions is built. This set contains all the elements required to use the method. This set will contain one proposition "object(Object Type)" if the method is from an object, that is, the method requires an object to be used, as opposed to constructors and static methods that can be used without creating objects first. Next, one proposition "object(Argument Type)" is added to the set for each input argument the method has, in which the Argument Type will be replaced by the corresponding object type. To conclude this set, a proposition "interface(Used Interface)" is added if the method requires any interface to be called before it can be used, and these interfaces are identified by the "use" dependency relationship.

It is worth noting that, if a method does not have one of these elements, the respective rule can just be ignored. It may happen that, in some cases, such as some constructors, the Precondition set will be empty.

To complete the operator, the Effects set must be built. This set will contain all the propositions that will become true once the operator is used. This set will contain one proposition "object(Return Type)" should the method return any object. A proposition "interface(Name)" is added to the set if the method is responsible for implementing a service. The set will also contain a proposition "link(Type1, Type2)" if the method creates a link between objects of Type1 (the object that has the method being used) and Type2 (the second object). This link can be "created" either with the input argument, as in the "set" methods of objects, or if it returns an object already associated with the target object, "get" methods for example. Lastly, a proposition "sequence(Name)" is added if the method starts any of the Sequence Diagrams.

To exemplify this process, let us use the class "JFrame" from the SWING framework, shown in figure 3.

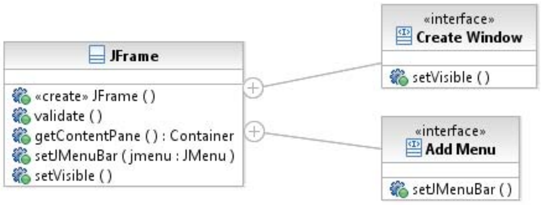


Fig. 3: JFrame Class from the SWING framework

Starting by the "JFrame" constructor method, we get that the operator name should be "JFrame.JFrame", since it is composed of both class and method names. Since this is a constructor, that has no input argument and does not require any other interface, its precondition set will be empty. The effect set will be composed only of its return type, a "JFrame" object. This will result in the following operator:

```
Name : JFrame.JFrame
 Precondition (PC)  : {}
 Effect (E) : object(JFrame)
```

Next, using the "setVisible" method, we get an operator named "JFrame.setVisible". Its precondition set will include one proposition "object(JFrame)", since this is an object method (neither static or constructor). Although this method does not return an object, it is responsible for the interface "Create Window", and a proposition "interface(Create Window)" is added to the effect set. This will result in the following operator:

```
Name : JFrame.setVisible
 PC : object(JFrame)
 E: interface(Create Window)
```

As a last example, let us use the "validate" method. As the method used before, this method has no input argument and is an object method, so the precondition set will be composed of the proposition "object(JFrame)". This method has no return type and it is not responsible for any interface, but it starts the "Update Screen" mechanism, shown in figure 2. Hence the effects will include one proposition "sequence(Update Screen)", resulting in the operator:

```
Name : JFrame.validate
 PC : object(JFrame)
 E : sequence(Update Screen)
```

This process is repeated for each method described in the Class Diagram, with each of them generating one operator. To complete the set of operators representing the framework, those created from any Sequence Diagram included in the model must be included. These diagrams contain information on functionalities that should be accessed through specialized classes and overwritten methods.

Each Sequence Diagram will be inspected in order to identify which of the methods it contains can be overwritten. It is assumed that those methods that do not invoke any other methods inside their behavior description can be overwritten without introducing errors in the behavior of the framework. Once the method to be overwritten is identified, an operator named "overwrite:ClassName.MethodName" is created.

The precondition set for this operator will be composed of one proposition "specialization(Class)", in which Class represents the class that has the method to be overwritten. The set also contains a proposition "link(Type1, Type2)" where Type1 is the type of object that starts the mechanism and Type2 is the type of object being overwritten, and a proposition "sequence(Mechanism)", indicating that the Sequence Diagram must be used for the method to be called.

The effect set of the operator will include one proposition "interface(Name)", if the method is responsible for any interface and one proposition "object(Argument Type)" for each input argument the method has. The argument type is included in the effect set, instead of the precondition set like the other operators, because when overwriting a method, it is possible to use the input methods, instead of having to supply them when calling a method.

Then, for each "overwrite" operator made, one additional operator is included. This operator will be named "specialize:Class”, with an empty precondition set, and with the only effect being the proposition “specialization(Class)”, where Class is the name of the class that contains the method being overwritten.

Let us use the Diagram from figure 2 as an example. This diagram represents the sequence for updating a screen in the SWING framework. To use this function, it is necessary to overwrite the method paint of a class that specializes a “Panel”. Then this mechanism must be started by the method “validate” from a “JFrame” object. Once overwritten, it is possible to use an object of the type “Graphics”. Using this information, the following operators are made:

```
Name : overwrite:Panel.paint
 PC : sequence(Update Screen),
link(JFrame, Panel), specialization(Panel)
 E : object(Graphics)


Name : specialize:Panel
 PC : {}
 E : specialization(Panel)
```

IV. Creating Sequence Diagrams

To make the sequence diagrams, a partial order planner is used. The planner requires three inputs: a set of possible operators, a goal state and an initial state. The possible operators are the ones constructed based on the framework model. The goal state will be a state composed of a proposition “interface(Target Interface)”, representing the interface that the user wishes to execute and an empty initial state will be used, so that it is considered that no element of the framework is currently being used.

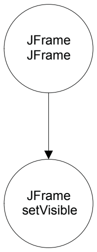


Fig. 4: Plan for the goal “interface(Create Window)”

Using these inputs, the planner will generate a sequence of operators that will allow the correct use of the desired interface. Using this plan, a sequence diagram is made in such a way that the methods it contains reflect the order of the operators used in the plan.

For example, let us consider a user that wishes to use the interface “Create Window” from the SWING framework. First, all the operators from the framework would be extracted, following the steps described in section III. Next, the planner would be initialized with an empty state, the framework operators and the goal “interface(Create Window)”. This would result in the plan shown in figure 4. Next, based on this plan, the sequence diagram shown in figure 5 can be constructed.

This is a very simple example of how this method can be used. It becomes more interesting when a sequence of interfaces is desired, using it as the description for the behavior of an application.

Suppose, then, a simple application for visualizing images. One possible way to model this application behavior is by the Activities Diagram shown in figure 6. This diagram shows the application needs to first create a window, then ask the user to choose an image file, then print the selected file in the screen.

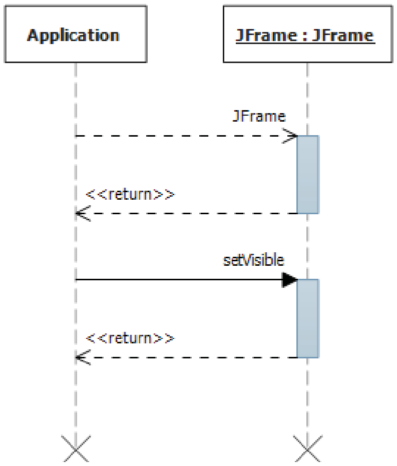


Fig. 5: Sequence diagram resulting from the plan from figure 4

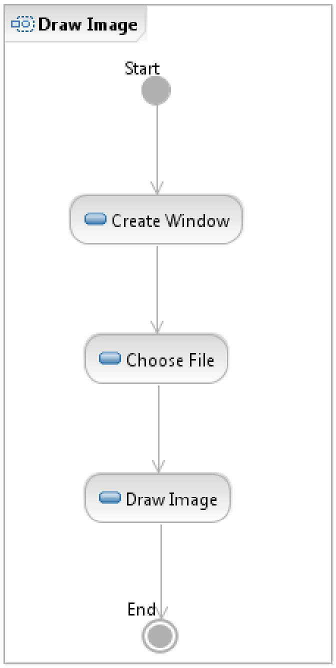


Fig. 6: Activity Diagram of the Image Viewer Application

To complete this application model, let us use a simple Class Diagram, composed of only one class with a single method, responsible for the sequence of actions. Such diagram is shown in figure 7.

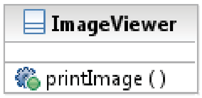


Fig. 7: Class Diagram of the Image Viewer Application

Using the activities described in figure 6 as the goals for the application, we get the sequence of propositions:

interface(Create Window), interface(Choose File) and interface(Draw Image). After presenting each of these goals to the planner, a tree representing the plan is obtained (fig. 8).

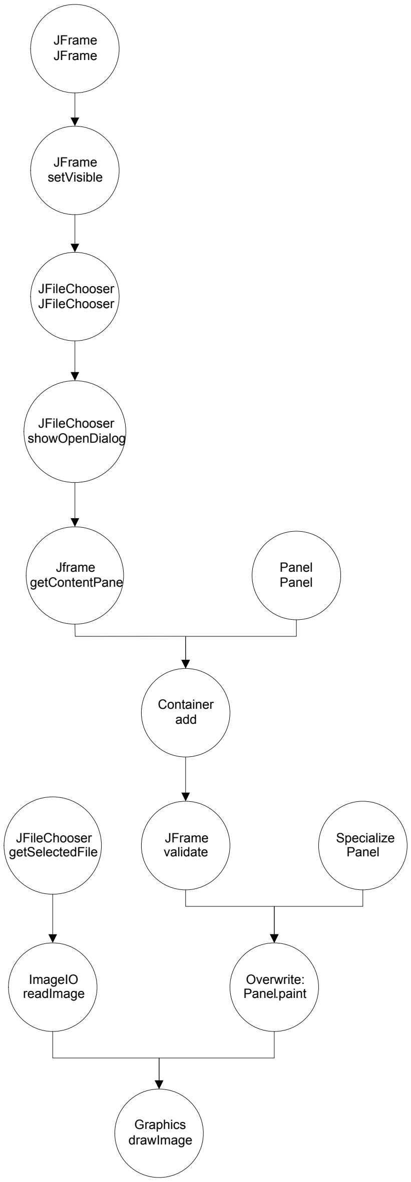


Fig. 8: Tree representing the plan obtained using the application goals

Since this tree has an "overwrite" operator, it will have to be broken down into three pieces. Each of these pieces will make a different sequence diagram. The nodes below the "overwrite" operator, in this case only the "Graphics.drawImage" operator, will compose the sequence diagram describing the overwritten method. The nodes besides the "overwrite" operator and their parents, in this case the operators "ImageIO.readImage" and "JFileChooser.getSelectedFile", will form the constructor of the new specialized class. The other operators will form the main sequence diagram, the one that will be invoked by the application. The resulting "main" sequence diagram, the constructor sequence diagram and overwritten method sequence diagram are shown in figures 9, 10, 11 respectively.

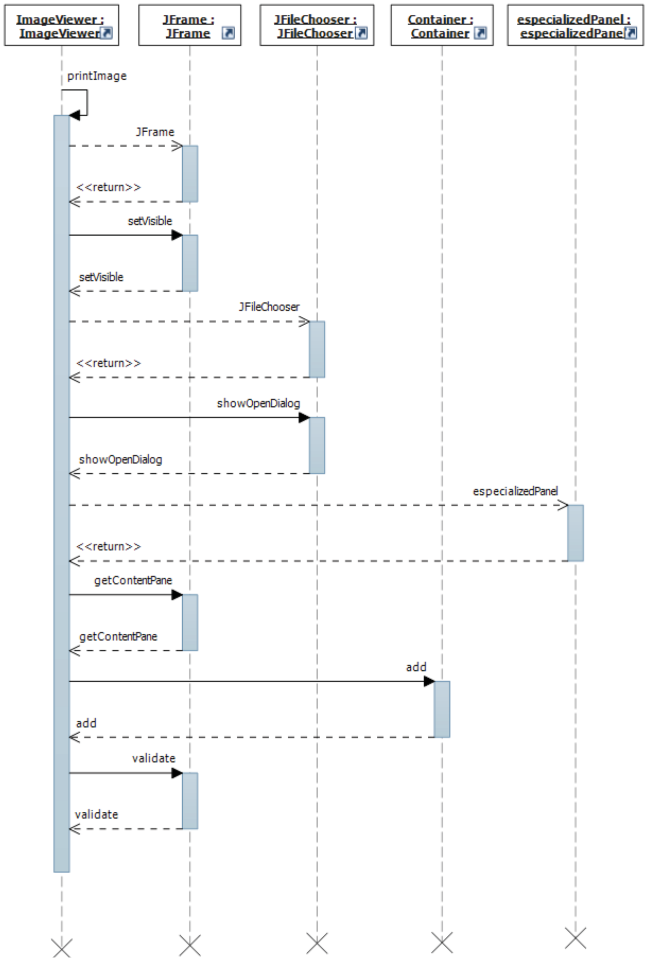


Fig. 9: Main Sequence Diagram

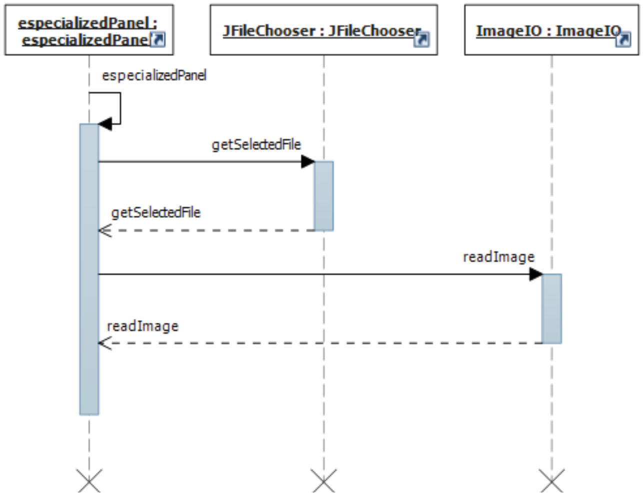


Fig. 10: New Constructor

After all the Sequence Diagrams are built, it is possible to alter the initial Class Diagram of the application, to include the used classes from the framework, as well as the new classes made. The resulting class diagram is shown in figure 12.

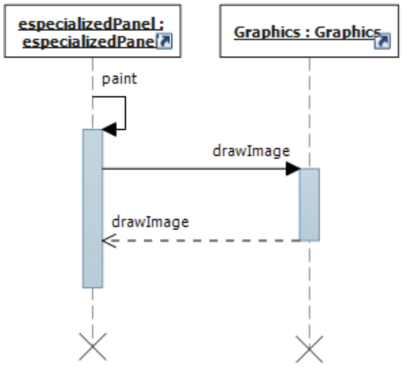


Fig. 11: Overwritten Method

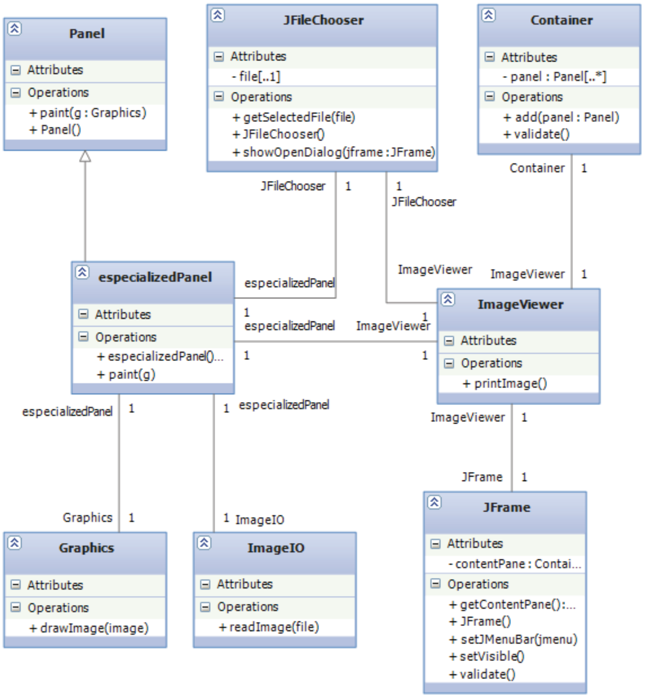


Fig. 12: Altered application Class Diagram

With the model obtained, it is possible to implement the desired application using the SWING framework.

This is only a simple model. The framework model could be refined, adding more classes and services to allow more complex applications, although, currently, the method only supports applications with a linear sequence of activities, that is, loops and branches are not yet supported. But the possibility of reusing a framework model, specially considering that different software engineers can use the same model in multiple projects, presents an opportunity to reduce the costs of using frameworks.

V. Related Works

This works is based on the web-service composition area. This area seeks a way to automatically create new Service Oriented systems, using as input a set of available web-services, usually described in BPEL.

In [10] the authors describe a way to use model-checking based planning to build new systems, using as input a set of BPEL descriptions of web-services, while [11] uses a knowledge based approach to a similar problem. Another approach is found in [12], where the authors show a method that can create a finite state machine representation of the desired system, based on the behavior description of the available services.

There is an advantage when working with web-services, instead of object-oriented systems, because web-services are well encapsulated components, that can more easily be chained together than their object-oriented counterparts. Moreover, the services provided can be considered black-boxes, that is, only their input and output have to be considered during development, making possible to ignore the internal workings and state of the service during usage, which is not true when working with frameworks.

Frameworks frequently have strict conditions under which their functionalities can be used. The method presented herein not only identifies which components from the framework must be used, but also creates a sequence of methods that ensure those components will be in the correct state at the time of their invocation.

Other works have also addressed the reusability problem. For instance, in [13], a method to identify which parts of a system can be reused is designed, but it focus more on identifying ’which’ components can be used, as opposed to ’how’ to reuse already available components, as is the case with frameworks. There is also [14], which addresses the problem in an algebraic manner. The paper presents an interesting approach, but since it predates the rise of the UML, it does not address the use concepts of classes, objects and the exchange of messages.

VI. Conclusion

A model-to-model transformation method is presented that identifies which components from a desired framework should be used to meet the application requirements.

This method allows for applications to be described as a sequence of services executions, identifying how to combine them to implement the desired solution. The aim of this method, is to make easier to reuse high level models, both framework and application models. The same application model could be used several times, with different framework models, in the same way as the framework model could be used with several applications.

This method has limitations in its current implementation. The model generated by the transformation, albeit correct, is not complete. It lacks the description of how to initialize the frameworks, using the example shown; for instance, it does not include a “main” method, which is required in any JAVA application to be correctly initiated.

Another restriction concerns the input model. The framework should be modeled in a very precise manner, in order not to introduce errors in the logical planner, and a non-standard way of using the sequence diagrams to describe the method that can be overwritten without introducing errors in the framework behavior. The models, both framework and application, are also very dependent on the use of fixed names to describe the interfaces (for the framework) and activities (for the application). In the current state, the application model is also constrained in the Activity description, in which it is not currently possible to use loops or branches.

Finally, the model generated through this transformation process does not consider some of the best practices of software design, such as decoupling of elements or ease of maintenance. The final model will, most likely, be very different from the one created by a person. Yet, apart its limitations, this is a novel approach to framework use, in which a transformation method identifies how the framework can be used by an application. It allows the reuse of framework knowledge by different people, since once modeled, the framework can be used by any other person in different projects.

Future work will focus on ways to remove, or at least reduce, the limitations described above. We also plan to investigate what impacts the use of more complex planners may have on the transformed models. Another question that will need to be addressed in the future is the use of multiple frameworks. The use of multiple frameworks is common enough in software development for it not to be ignored, but it brings its own set of problems [15], such as possible architectural incompatibilities for instance. Future versions of this method will seek ways to identify incompatibility between frameworks, and ideally, ways to circumvent them.

REFERENCES

[1] M. E. Fayad and D. C. Schmidt, "Object-Oriented Application Frameworks," *Communications of the ACM*, vol. 40, no. 10, pp. 32–38, 1997.

[2] R. E. Johnson, "Frameworks = Patterns + Components," *Communications of the ACM*, vol. 40, no. 10, pp. 39–42, 1997.

[3] K. Czarnecki and S. Helsen, "Classification of model transformation approaches," in *Proceedings of the 2nd OOPSLA Workshop on Generative Techniques in the Context of the Model Driven Architecture*. Citeseer, 2003, pp. 1–17.

[4] S. Schönberger, R. K. Keller, and I. Khriss, "Algorithmic support for model transformation in object-oriented software development," *Concurrency and Computation: Practice and Experience*, vol. 13, no. 5, pp. 351–383, 2001. [Online]. Available: http://dx.doi.org/10.1002/cpe.555

[5] D. Varró and Z. Balogh, "Automating model transformation by example using inductive logic programming," in *Proceedings of the 2007 ACM symposium on Applied computing*, ser. SAC '07. New York, NY, USA: ACM, 2007, pp. 978–984. [Online]. Available: http://doi.acm.org/10.1145/1244002.1244217

[6] G. Engels, R. Hucking, S. Sauer, and A. Wagner, "UML Collaboration Diagrams and Their Transformation to JAVA," pp. 473–488, 1999.

[7] T. Behrens and S. Richards, "StateLator - Behavioral Code Generation as an Instance of a Model Transformation," in *Advanced Information Systems Engineering*, ser. Lecture Notes in Computer Science, B. Wangler and L. Bergman, Eds. Springer Berlin / Heidelberg, 2000, vol. 1789, pp. 401–416. [Online]. Available: http://dx.doi.org/10.1007/3-540-45140-4_27

[8] R. E. Fikes and N. J. Nilsson, "Strips: A new approach to the application of theorem proving to problem solving," *Artificial Intelligence*, vol. 2, no. 3-4, pp. 189–208, 1971. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/0004370271900105

[9] A. Tate, "Generating project networks," in *Proceedings of the 5th international joint conference on Artificial intelligence - Volume 2*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1977, pp. 888–893.

[10] P. Bertoli, M. Pistore, and P. Traverso, "Automated composition of Web services via planning in asynchronous domains," *Artificial Intelligence*, vol. 174, no. 3-4, pp. 316–361, Mar. 2010. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0004370209001489

[11] E. Martínez and Y. Lespérance, "Web service composition as a planning task: Experiments using knowledge-based planning," in *Proceedings of the ICAPS-2004 Workshop on Planning and Scheduling for Web and Grid Services*, 2004, pp. 62–69.

[12] D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Mecella, "Automatic service composition based on behavioral descriptions," *International Journal of Cooperative Information Systems*, vol. 14, no. 4, pp. 333–376, 2005.

[13] G. Caldiera and V. R. Basili, "Identifying and qualifying reusable software components," *Computer*, vol. 24, no. 2, pp. 61–70, 1991.

[14] F. Parisi-Presicce, "A rule-based approach to modular system design," in *Software Engineering, 1990. Proceedings., 12th International Conference on*, Mar. 1990, pp. 202–211.

[15] D. Garlan, R. Allen, and J. Ockerbloom, "Architectural Mismatch: Why Reuse Is Still So Hard," *Software, IEEE*, vol. 26, no. 4, pp. 66–69, 2009.

Investigating the use of Bayesian networks as a support tool for monitoring software projects

Fábio Pittoli[1], Abraham L. R. de Sousa[1, 2]
[1]Centro Universitário La Salle - Unilasalle
Canoas, Brazil
{fabio.pittoli@gmail.com, rabelo@unilasalle.edu.br}

Daltro J Nunes[2]
[2]Instituto de Informática
Universidade Federal do Rio Grande do Sul - UFRGS
Porto Alegre, Brazil
{rabelo, daltro@inf.ufrgs.br}

Abstract— **The monitoring of software development is one of the most important activities of software projects management. In this context, this paper proposes a Bayesian approach integrated with a software process management environment. The aim is to investigate how this probabilistic approach can be used for projects monitoring. Preliminary results indicate that the use of Bayesian networks brings the power of quantitative and qualitative evaluation of some common scenarios of project management, leading to the manager a greater power of decision making.**

Software Development Process; Bayesian Network; Project Management

I. INTRODUCTION

One of the main challenges of any project manager is trying to ensure that software development project will be concluded within the constraints of time, cost, scope, and quality. Establishing these constraints means working with estimates. But the software development is an inherently uncertain endeavor, because there is no way to ensure that during the progress of the project delays will not occur, will not lack resources, or the scope will not change. Within this context of reasoning under uncertainty arise the Bayesian networks [1][2][3] that are used in situations where there are causal relations, but our understanding of what is really happening is incomplete, requiring probabilistic description for a better understanding. This type of network can be used for different types of reasoning, as predictive analysis, to investigate the impact of changes (cause and effect), and support decision-making. The main goal of this paper is to s how the preliminaries results of a study that aims to integrate an environment for project management with Bayesian networks during the monitoring of software development projects. In this sense, the manager interacts with a Bayesian model to identify potential behaviors of the project and thereafter, decisions are made.

II. THEORETICAL GROUNDING

Estimated time and cost development is an activity that requires attention and that has great influence in the process of software development. It's the estimation definition which ensures that a project will succeed or not during its execution. Effort estimates are useful for clients and developers. [5]. Based on these estimates, the organization that wants to hire the project can assess and monitor the implementation costs, evaluate proposals, and develop realistic budgets and schedules.

Whichever method you choose to run estimates, it is always important to observe that the estimation process is a complex domain where the causal relationship among factors are non-deterministic and with an inherently uncertain nature [6]. For example: we can assume that there is a cl ear relationship between development effort and team experience, and that when team experience increases, the effort decreases, although, there are no concrete data proving that. So, would be correct to say that handle estimates, necessarily, result in dealing with uncertainties.

A. *Bayesian Networks*

Bayesian networks (BN) were developed in the early '80s to facilitate the task of predicting and diagnosing in Artificial Intelligence (AI) systems [1]. The name Bayesian Network derives from the use of the mathematical formula for the calculation of probabilities established in 1763 by Thomas Bayes. According to [2], the BN allow us to express complex cause-effect relationships based on the problem investigated. The graphical representation of a Bay esian network is composed of nodes that represent random variables that assume discrete values or continuous. The arches represent the causal relationships among nodes. For example, we can consider a classic case of BN presented by [7] about a new burglar alarm. This alarm is very reliable; however, it can also trigger if an earthquake occurs. Two neighbors, John and Mary, pledged to call if the alarm is ringing. John always calls when he hears the alarm, but, sometimes confuses the alarm with the phone and call also in these cases. Maria, however, likes to listen to loud music and sometimes does not hear the alarm. Fig. 1 shows the defined the network topology to this case and tables of conditional probabilities to each node. This form of representation can be used to represent discrete variables or the continuous variables. Each of the lines in the table contains the conditional probability for each conditional case parent nodes. A conditional case represents a possible combination of values for the parent nodes [8].

B. *Related Work*

Within several studies that use BN to support software process development, we tried to divide the references among some of the key areas of management and software development, such as: risk management, predicting failures and effort estimation.

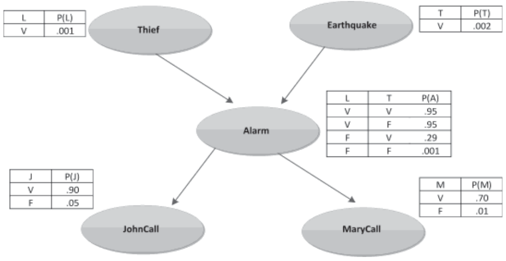


Figure 1. Bayesian network with the conditional probabilities

- Risk Management: the study presented in [9] proposes a standard architecture for risk identification called Risk Identification Pattern model. The use of Bayesian networks as the main component of the model made it possible to represent the relationships among risk factors present in web projects.
- Predicting Failures: in [12] is presented a review of the use of BN for predicting faults and software reliability. Beside this, it proposes an approach that allows us to use dynamic discretisation algorithms for continuous nodes.
- Effort Estimate: stands out the comparative study of models of Bayesian networks focused on the effort estimation in web projects presented in [6] that disseminates the results of an investigation where eight Bayesian network models were compared for their accuracy in estimating effort for web projects. The results showed that the Bayesian networks represent a suitable approach for the treatment of effort estimates.

III. Bayesian Networks as a Tool to Support Monitoring of Projects

Monitoring software projects using mechanisms to detect changes during its progress contributes to that unexpected events do not deviate the planning. If this happens, it is possible to m ake changes in order to adapt to new reality imposed a l ess traumatic and fastest possible. Unlike other approaches used to make estimates that use parametric measure, the use of BN suggests a statistical value (approx).

The following are the main software components used to compose the monitoring model proposed. After this, is shown a conceptual model of the proposed solution, which seeks to provide how will the interaction among the tools that form the solution, besides the characteristics and details of implementation:

- WebAPSEE[14]: through this environment is possible to model a de velopment process, defining the activities, the sequence among them, the papers involved, and the execution time. The environment allows its execution through a machine that coordinates the activation of activities and agenda of the developers. It was chosen as a supportive environment for project management because allows a r igorous control over the variables that form a software process.
- Bayesian networks tool: in this study we used GeNIe[1], developed by Decision Systems Laboratory, from Pittsburgh University. The GeNIe software was used to model the Bayesian networks for the evaluation of the proposed model, because of its ease of use and that the free version has no restrictions on the maximum possible size for a BN.

A. *Presentation of the monitoring model scenarios*

For better understanding of the monitoring and control model proposed, it is necessary to presentation and analysis of scenarios where the model will work and to observe the pre and post conditions required for proper operation. Fig. 2 presents the scenario 1 that handles the BN configuration required for the correct functioning of the model. As prerequisites, it is necessary a previous modeled process in the WebAPSEE; furthermore, the topology of the network should already be set according to the aspect that you wish to monitor in the process (time, cost, quality,etc.). From this, is shown a web interface through which you can select relevant data from the running process according to the aspect that you wish to monitor. For example: considering that you want to monitor the process on the aspect of time, would be possible to select items such as "number of agents involved" and "total hours remaining" and so on. From this, the selected data will be extracted from the running process. After extraction of this information, a table containing the information extracted is generated automatically. Through this table will be possible to identify information used to configure the evidence in the BN, which is the post condition of the scenario 1.

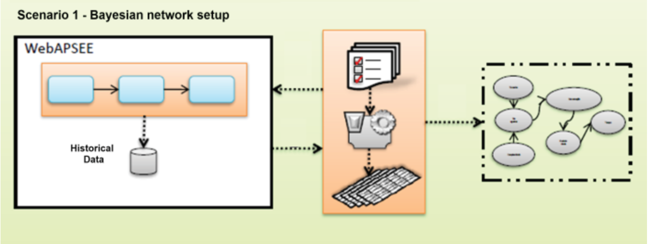


Figure 2. Bayesian network configuration

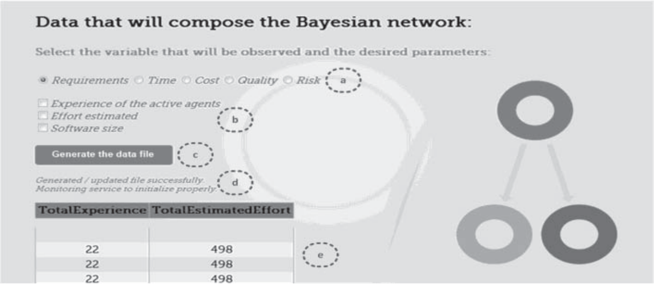
The scenario 2 refers to the monitoring process. As a prerequisite for proper functioning of scenario 2 we have the fact that the process is running and the BN is configured. Initially, it is necessary to identify changes of state in the running process. Thus, a Windows Service has been developed in order to monitor the data involved in the process. This action of checking changes in the process is done in time intervals previously established. So, it make possible to use the latest data from running process directly in the web interface. After this, we can update the BN, spreading the current state of the

[1] http://genie.sis.pitt.edu/

process for the network. Thus, this is the post-condition required for scenario 2.

B. *Prototype developed*

After the presentation of the scenarios observed, it is necessary to present a prototype of the proposed model. It is essential that, at first, we have a software process modeled and running in the WebAPSEE tool. Another important aspect concerns the selection of the network model to be u sed by software components modeling Bayesian networks GeNIe. For example: if the aspect which you want to monitor in the process is the Time, it is necessary that the model of Bayesian network analysis has some node with characteristics related to Time. The same analogy applies when you want to monitor other aspects of the process, as Cost, for example. From this it becomes possible to b egin the process of monitoring and use the web interface developed. The Fig. 3 shows the web interface developed.



Data that will compose the Bayesian network:

Select the variable that will be observed and the desired parameters:

● *Requirements* ○ *Time* ○ *Cost* ○ *Quality* ○ *Risk*

□ *Experience of the active agents*
□ *Effort estimated*
□ *Software size*

Generate the data file

Generated / updated file successfully. Monitoring service to initialize properly.

TotalExperience	TotalEstimatedEffort
22	498
22	498
22	498

Figure 3. Web interface for data collection

This web interface is organized basically as follows: *a) Characteristics of the process*: at the top of the page are various HTML controls type radio, where each identifies and makes it able to be selected each of the different characteristics relate to a software process, as Time, Cost and Quality; *b) Data related*: after selecting a f eature, are displayed in the field below, various controls type checkbox where eache of them with respect to a given related to the selected feature. It is possible to select various data related to the same characteristic; *c) Generate data file*: control type button whose essential function is to confirm the selection of parameters and initiate the process of generation of the file containing the process data, subsequent display of data in the informative table containing the data evidence and boot the Windows Service responsible for monitoring the database process and update the data file; *d) Informational message*: if everything went as expected and without occurrence of errors, the data file was created/updated correctly, the Windows Service was successfully started and the following message is displayed: “File created/updated successfully. Monitoring service initialized correctly”. However, if na error occurs during generation of the data file or during Windows Service boot, the following message is displayed: “An error occurred while attempting to generate the file. The monitoring service was not initialized”; and *e) Table containing the data from the monitored process*: after the creation / update of the file containing the data from the monitored process, a table in the web interface is responsible for listing the data. This update is always done periodically and is performed after each Windows Service execution. The goal is to ensure the ability to view directly in the table the latest data from the process to enable the identification of evidence and use them in their Bayesian network used in software GeNIe or Netica.

IV. Model Evaluation

In the evaluation is modeled a development process in the WebAPSEE software. The idea is that this process be as similar as possible with a specified process to model the development of a real software project, but, being a procedure evaluation for a model created for proof of concept, the process used is simplified, containing only features that are fundamental to the proposed assessment and with reference to the tasks defined by the RUP (Rational Unified Process) for small projects methodology. Some Bayesian network models present in the literature were used and adapted to make them in accordance with the data that is extracted from the process and are present in the data file generated by the web interface developed. So, independent of the chosen characteristic to be monitored , the Bayesian network defined will allow a true representation of the running process in the WebAPSEE. The evaluation seeks to answer some key questions of project managers referring to changes in estimates that are recur in software projects.

A. *Software process*

In order to evaluate the proposed model, a process has modeled considering some of the main activities defined in the RUP for Small Projects methodology. Using the WebAPSEE software as a tool for process modeling, aimed to organize the activities by disciplines, according to what is proposed by methodology. The used disciplines by the process were: Requirements, Project Management, Analysis, Implementation, Tests, Change Management. It is also important to mention that this is an iterative and incremental process [15], in other words, each stage is executed several times during the development process. This allows that our understanding about the problem increases through successive refinements, making an effective solution is obtained after many iterations.

B. *Bayesian networks models*

The modeling of Bayesian networks used for implementation and evaluation of the evaluated scenarios were constructed based on the model known as MODIST [16], which cares about the quality of predictions and wi th risk management in large software projects. The MODIST project is based on Bayesian network and it tries to produce development models and testing process that take into account statistical concepts missing in traditional approaches to development. It was decided to develop a Bayesian network to monitor the Requirements activities set, especially, taking into account the aspect of time. One reason for having been chosen by the requirements lies in fact that is in requirements, which normally has the main problems with estimates. It is also important to mention that it is completely feasible that new networks are designed to monitor other aspects and set of activities, such as Tests, Change Management or Analysis, for example. Furthermore, could also consider the monitored process as whole as a single activity and, so, develop a single network responsible for taking care of all aspects. Since the goal is, from the Bayesian network, can to answer some key questions of project managers in relation to possible changes in estimates, it has searched divide the evaluation scenarios, where each will show different situations in relation to requirements and, from the results indicated by the network, will be suggested a response to pointed questioning.

C. Evaluated Scenarios

To make the inference of Bayesian network for the subsequent definition and identification of evidence, in each of the evaluated scenarios the network was initially trained with historical data from 100 completed projects and that have some similarity with the process analysis. The historical data base used is constituted by data from laboratory simulation and purpose of the scenarios is to simulate real situations that may occur during a monitoring a software process. Is shown, a step by step example of a scenario used to verify the efficiency of the developed model: *A software process that has the following characteristics: A high degree of novelty in what will be developed (Novelty); b) High Complexity (Complexity); c) Big size (Size); d) Team with a low degree of experience (Staff Experience); and e) High estimated effort (Estimated Effort). Questioning: the inclusion of more experienced professional in the project will make the effort required to complete the project decrease?*

After configuring the situations mentioned above, where in the modeled Bayesian network will be the evidence, shows that, when we have a situation as described above, there is a high probability (51%, in this case) that the Required Effort (Required Effort) to complete the project is high (High). The Fig. 4 demonstrates the behavior of the Bayesian network after setting evidence. The question asked about this scenario concerns the possible reduction of required effort if they are allocated to the process new professionals who possess a greater degree of experience.

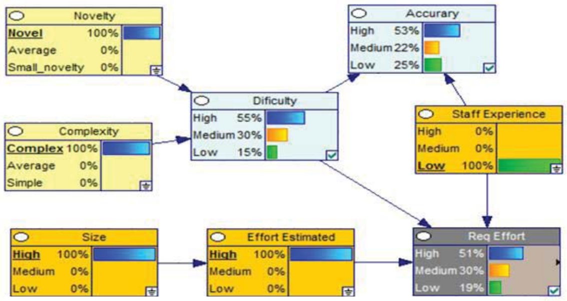


Figure 4. Bayesian network with evidence of the scenario

The inclusion of more experienced professionals in the monitored process made the Required Effort continue high, but, it decreases from 51% to 47%. Furthermore, the Accuracy, that identifies how close the estimates are in relation to the actually required, also improved, from 53% to 54%. Thus, we can understand that on large projects and have a high degree of complexity and are also completely new in terms of technology or segment, considering an inexperienced team initially, by including in that team new professionals with a high level of experience in relation to the proposed problem, the required effort, still continuing high, have experienced a decrease.

V. Final Considerations

This paper presented a probabilistic approach to support the monitoring of software development projects. From the model here presented, becomes possible to develop a series of other experiments and improvements. For future work, we can mention the possibility of checking and measuring the impact that changes in estimates during the progress of the project will have on the quality of the final product. Another point that could be further explored in future concerns the possibility to centralize in one place both the data collection process monitored and the graphical display of Bayesian networks for the subsequent configuration of inferences, allowing ease of use of the model. Furthermore, new Bayesian networks can be developed in order to monitor activities of other groups of a given project, different from those that have been mentioned here and presented. It is important to mention that the model presented here is constantly evolution, because it can be improved and adapted to the software process used according with the models of Bayesian networks used, can be more or less complex depending of aspect that want to evaluate and depending on the number of variables that will be involved in.

References

[1] Charniak, E. Bayesian Networks Without Tears. AI MAGAZINE, v. 12, n. 4, p. 50-63, 1991.

[2] Fenton, N. E.; Neil, M. A Critique of Software Defect Prediction Models. IEEE Transactions on Software Engineering, v. 25, p. 675-689, 1999.

[3] Stamelos, I. et al. Estimating the development cost of custom software. Inf. Manage., v. 40, p. 729-741, 2003.

[4] Mendes, E.; Mosley, N. Bayesian Network Models for Web Effort Prediction: A Comparative Study. IEEE Transactions on Software Engineering, v. 34, p. 723-737, 2008.

[5] Russell, S. J.; Norvig, P. Artificial intelligence: a modern approach. [S.l.]: Prentice-Hall, Inc., 1995. 415-429 p.

[6] Marques, R. L.; Dutra, I. Redes Bayesianas: o que são, para que servem, algoritmos e exemplos de aplicações. Universidade Federal do Rio de Janeiro. [S.l.]. 2000.

[7] Al-Rousan, T.; Sulaiman, S.; Salam, R. A. A risk identification architecture pattern based on Bayesian network. [S.l.]: [s.n.]. 2008. p. 1-10.

[8] Jeet, K. et al. A Tool for Aiding the Management of Schedule Overrun. IEEE 2nd International Advance Computing Conference, v. 2, p. 416-421, 2010.

[9] Xiaocong, H.; Ling, K. A risk management decision support system for project management based on bayesian network. 2nd IEEE International Conference on Information Management and Engineering, v. 2, p. 308-312, 2010.

[10] Fenton, N. E.; Neil, M.; Marquez, D. Using Bayesian Networks to Predict Software Defects and Reliability. [S.l.]: [s.n.]. 2007.

[11] Sanchez, A. J. Software maintenance project delays prediction using Bayesian Networks. Expert Syst. Appl., v. 34, p. 908-919, 2008.

[12] Ambiente WebAPSEE. Simpósio Brasileiro de Engenharia de Software Florianópolis: Informática-UFSC, v. 1, p. 1-6, 2006.

[13] Kruchten, P. The Rational Unified Process: An Introduction. 3. ed. [S.l.]: Addison-Wesley Longman Publishing Co., Inc., 2003.

[14] Modist. Models of Uncertainty and Risk for Distributed Software Development. EC Information Society Technologies Project IST-2000-28749. [S.l.]. 2003.

Reuse of Experiences Applied to Requirements Engineering: An Approach Based on Natural Language Processing

Adriano Albuquerque, Vládia Pinheiro
Programa de Pós-Graduação em Informática Aplicada
University of Fortaleza - UNIFOR
Fortaleza, Brazil
{adrianoba,vladiacelia}@unifor.br

Thiago Leite
Programa de Pós-Graduação em Informática Aplicada
University of Fortaleza - UNIFOR
Fortaleza, Brazil
thiagolcarvalho81@gmail.com

***Abstract*— Concerning software development projects, it is known that many of the problems originating from the phase of Requirements Engineering are recurring, and happen repeatedly within the same project or in different projects. Given this context, we defined a process to support the reusability of previous experiences that is based on Artificial Intelligence approaches - Case Based Reasoning and Natural Language Processing. This paper presents the proposed approach and the initial results obtained in a proof of concept.**

Keywords-* *Requirements Engineering, Knowledge Management, Experiences reuse, Natural Language Processing.

I. INTRODUCTION

In software development projects, many commonly reported problems involve recurring situations, especially problems originating in the phases of Requirements Engineering (RE). Although this fact is known and expected by all those involved in a development process, the reuse of previous experiences to solve new problems is not a reality at IT (Information Technology) companies. Inefficiency in the management of knowledge about the cause of problems, and about the solutions applied in RE, leads companies to deviations of lead-times and costs in their projects.

Pereira [1] and Standish [2] identified the most important causes of the most commonly recurring problems in RE. We argue that, for most of the problems addressed in the literature, the representation and retrieval of similar problems and suggestions for application of solutions already experimented in such cases are able to minimize the impact on project indicators (lead-time, effort, cost, etc.) and enable the dissemination of knowledge among the various projects of an IT company. In this regard, we propose an approach for the reuse of experiences in RE that uses techniques from the area of Artificial Intelligence (AI): Case Based Reasoning (CBR) and Natural Language Processing (NLP).

A distinguishing feature of the proposed approach is the possibility of representing and retrieving cases (problems and solutions) described in natural language, for example: in the Portuguese or English language. Traditional approaches in CBR use the form of description of cases by attribute-value, whereby structured fields will abstract aspects of the cases restricting their characterization and representation *a priori*. The description of a case in the form of text makes the representation thereof more natural and complete, because users become freer to express themselves about the case, and specific aspects of a case can be made explicit. In contrast to the ease and completeness in the textual description of cases, one has difficulty in retrieving similar cases, since the computational treatment of non-structured representations such as natural language text is not trivial. Related to this, we support the reuse of experiences using techniques from the area of Natural Language Processing (NLP), such as morphosyntactic analysis and similarity analysis.

This paper is organized as follows. Initially, we present studies that utilized reuse of experiences in Software Engineering. In the second section, the process of supporting the reuse of experience in RE is presented. In the third section, we present the methodology for evaluating the proposed process and we analyze the results. Finally, we conclude our study and present possibilities for future work.

II. RELATED WORKS

In general, CBR systems are commonly used in Software Engineering to support activities such as CRM, help desk, assessment tools, etc. Joshi and McMillan [3] defined a tool called MESCA (MEnu BrowSer using Case bAsed reasoning) that enables the creation and/or reuse of graphical interface for applications. MESCA focuses on the reuse of code that supports the software implementation phase and applies primarily to user-developers. Gabineski and Lorenzi [4] defined a tool to support project management, which is responsible for controlling the number of activities assigned to a person, as well as control the execution of these activities. Jani and Mostafa [5] proposed a tool called SRSQAS (Software Requirement Specification Quality Analysis System), whereby it is possible to measure the quality of requirement specifications based on 11 items. Basically this tool compares a requirement specification with the case base and gauges the level of quality thereof. Praehofer and Kerschbaummayr [6] defined a tool called CASA (Computer Aided Systems Architecting) focused on reusing artifacts of Requirements Engineering. The similarity calculation method takes into account the structural organization of the documents. Similarities are thereby found between an input artifact and the Artifacts Base, to find the artifacts that are structurally similar to the input artifact. Based on this, the tool tries to reuse similar structures to create new artifacts. In most of the approaches that use CBR in the activities and phases of Software Engineering, we identified the single and prevalent use of the attribute-value formalism for representation and retrieval of similar cases previously stored in the case base.

III. AN ARTIFICIAL INTELLIGENCE APPROACH FOR REUSE OF EXPERIENCES IN REQUIREMENTS ENGINEERING

Intelligent Systems that use techniques of Case Based Reasoning (CBR) are based on the simple idea that, in similar problems, similar solutions apply. CBR is an approach to problem solving and experience-based learning, solving problems by retrieving and adapting past experiences – called cases – stored in a Case Base [7]. Specifically in the Software Engineering community, there is a consensus that most problems are recurring, therefore managing cases from previous projects enables the implementation of successful solutions in several new projects, avoiding waste of resources to address repeating situations.

The knowledge base of a CBR system is represented in the form of cases, a Case Base, that describe concrete experiences of *problems* that have occurred and the *solution*(s) that were applied. In this paper, we propose a hybrid formalism for representation of a case: object-oriented and natural language. Figure 1 shows a class diagram in UML for the representation of a case. A case is described by a *Problem* and one or more objects of the *Solution* class.

The basic attributes of a *problem* are: (i) project — name of project where the case was verified; (ii) phase — phase of software engineering where the problem occurred, for example: "Requirements"; (iii) sub-phase — sub-phase where the problem occurred, for example: "Requirements Elicitation," "Requirements Analysis," etc.; (iv) artifact — artifact where the problem was found, for example: "Use Case," "Business Rules," etc.; (v) causer — who or what caused the problem, for example, "Person," "Technology," "Process," etc.; (vi) cause — cause(s) of the problem. Some examples include: "usage case scenario unspecified," "business rule unspecified," "change in business rule unspecified," "traceability matrix incomplete," etc.

These basic attributes serve as an initial qualification of the problem. However, they are not enough to express it fully and with all the specificities necessary for possible reusability. Additionally, in order to enable a complete and natural representation of a *problem*, we propose the attribute called ***description*** that contains a textual report on the problem related to the case.

Additionally, the basic attributes of a *solution* are: (i) action — description of what was done to solve the problem; (ii) executor — who performed the action; (iii) artifact — artifacts to be changed to the solution to be implemented; and (iv) involved — people, processes or organizations who have been affected by the deployment of the solution.

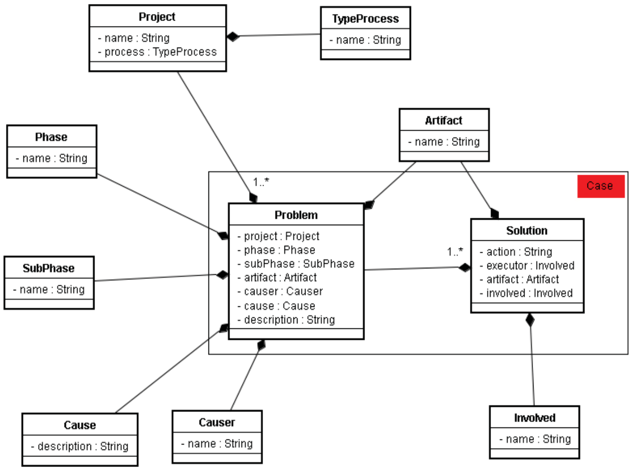


FIGURE 1. CLASS DIAGRAM OF THE CASE BASE, IN UML.

The process proposed to support the reuse of experiences in RE using techniques of CBR and NLP is shown in Figure 2 and detailed below.

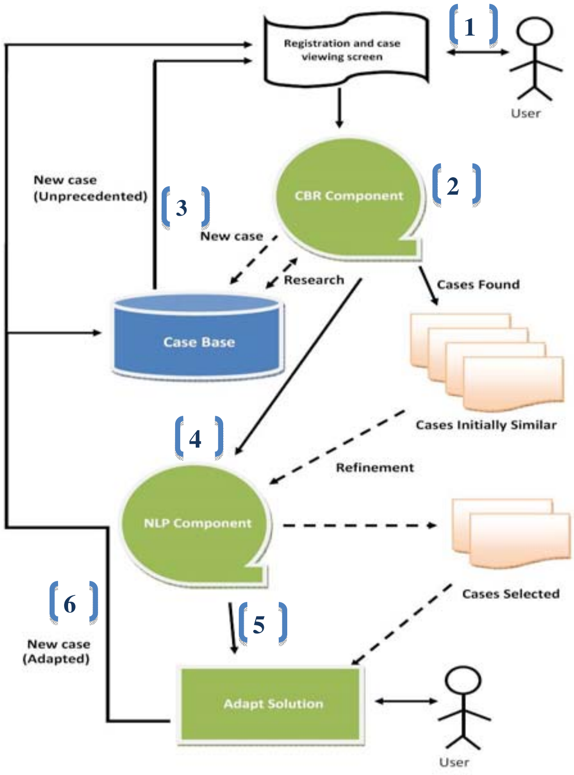


FIGURE 2. ARCHITECTURE OF THE PROCESS TO SUPPORT THE REUSE OF EXPERIENCES IN ER.

Step 1: To start the process, the user (requirements engineer, requirements analyst, project manager, developer, etc.) enters basic information about the new *problem*. Figure 1 shows the UML class diagram of the representation of a *problem*.

Step 2: In the "CBR Component" component, the data of the new *problem* are compared with the Case Base. The Attribute-Value pairs of the *problem* object are compared with the various cases existing in the Case Base, aimed at an initial selection of similar cases. The similarity between two cases c_1 and c_2 is calculated by the formula below, based on Wangehheim [7, p.112], which defines the similarity by the weighted average of the similarity between the values of each *index* of c_1 and c_2:

$$Sim_A(c_1,c_2) = \sum^n \frac{\underline{(\boldsymbol{Vs_i}\ \boldsymbol{x}\ \boldsymbol{P_i})}}{\sum \boldsymbol{P_i}} \qquad (1)$$

Where,

- ***n***: number of discriminatory attributes of a case (indexes)
- $\boldsymbol{Vs_i}$: similarity between the values of index $\boldsymbol{i}$ in c_1 and c_2, which is attributed per parameter.
- $\boldsymbol{P_i}$: weight of index $\boldsymbol{i}$, attributed per parameter.

For each index $\boldsymbol{i}$, must be set a weight $\boldsymbol{P_i}$, which defines the importance of the index in the calculation of similarity between cases, and a value of similarity $\boldsymbol{Vs_i}$ between the possible values of each index. In this work, in order to define the values of similarity between the possible values of each index, specialists in engineering requirements of a company from the federal government of Brazil analyzed the level of relationship or dependence between each value. For example, in Table 1 we present the similarity values (***Vs***) defined by specialists for index *Artifact*: the artifact "Use Case" is related – from the highest to lowest level –to the artifacts "Use Case" (100%), "Business Rule" (80%), "Functional Requirements" (70%), "Non-Functional Requirements" (30%), "Traceability Matrix" (30%) and "Technology" (0%). It is important to point out that these values were used in the evaluation of this proposed approach, but may be customized for each company or situation of use.

TABLE 1. SIMILARITY BETWEEN THE POSSIBLE VALUES OF THE INDEX *ARTIFACT*.

	UC	RN	RF	RNF	DV	MR
UC	1	0.8	0.7	0.3	0	0.3
RN	0.8	1	0.6	0.1	0	0.3
RF	0.7	0.6	1	0.5	0.5	0.3
RNF	0.3	0.1	0.5	1	0.5	0
DV	0	0	0.5	0.5	1	0
MR	0.3	0.3	0.3	0	0	1

CU = Case of Use | RN = Business Rule |
RF = Functional Requirements|
RNF = Non-Functional Requirements|
DV = Vision Document | MR = Traceability Matrix

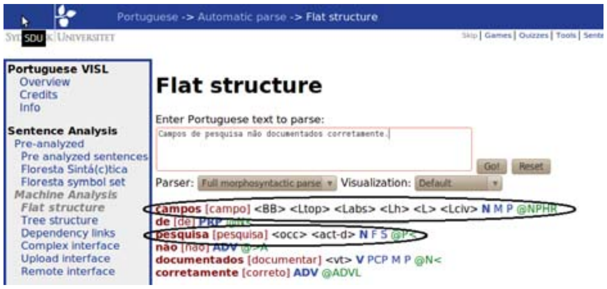
In this work, the *indexes* are *artifact*, *causer* and *cause*. These were chosen because they are more discriminatory of a case. After all, a cutoff value must be applied to select similar cases. To illustrate the calculation of similarity, let us suppose the following values for the indexes of a new *problem* (case c_1): *Causer* = "Organization" (OR); *Artifact* = "Business Rule" (RN); *Cause* = "Wrong Business Rule Specified" (RNEE). Considering the following values for a case c_2, existing in the Case Base: *Causer* = "Process" (PR); *Artifact* = "Traceability Matrix" (MT); *Cause* = "Wrong Traceability Matrix Specified" (MREE). Applying the values in formula (1), we have the following:

$$Sim_A(c_1,c_2) = \frac{\underline{(\boldsymbol{Vs}\text{OR.PR x } \mathbf{1}) + (\boldsymbol{Vs}\text{RN.MR x } \mathbf{3}) + (\boldsymbol{Vs}\text{RNEE.MREE x } \mathbf{5})}}{9}$$

$$Sim_A(c_1,c_2) = \frac{\underline{(\mathbf{0.5}\text{ x }\mathbf{1}) + (\mathbf{0.3}\text{ x }\mathbf{3}) + (\mathbf{0}\text{ x }\mathbf{5})}}{9} = \mathbf{0{,}15}$$

Step 3: If no similar case is returned in step 2, the input case is a new case and will be stored in the Case Base.

Step 4: In this step, the Cases Initially Similar are refined through using NLP techniques. Actually, the "NLP Component" defines the cases for reuse based on the number of words in common between the textual *description* of the input problem and the *description* of the pre-selected cases in step 2. For this, the Vector Space Model technique and the morphosyntactic analysis are applied to the texts in question. In detail, the frequency of each word in the "Noun" and "Verb" word classes, contained in the texts is calculated and those cases with *m* (defined by parameter) words in common with the input case are defined for reuse. The word classes "Noun" and "Verb" were chosen because they express things and actions, respectively, and therefore are the words that matter most to the semantic value of the text [8]. For example, Figure 3 shows the morphosyntactic analysis [9] of the descriptive text of one of the cases from the Case Base (pre-selected in step 2). Considering the description of the input *problem* "*Search functionality defined in wrong fields*." [*Funcionalidade de pesquisa definida com campos errados*], the words (nouns and verbs in Portuguese) of the input case are {*funcionalidade*, ***pesquisa***, ***campo***}, all with a frequency of 1. In the pre-selected case (Figure 3) the words are {***campo***, ***pesquisa***}, all with a frequency of 1. Two words were identified as being in common between the cases.



Flat structure

Enter Portuguese text to parse:

Campos de pesquisa não documentados corretamente.

Parser: Full morphosyntactic parse Visualization: Default

campos [campo] <BB> <Ltop> <Labs> <Lh> <L> <Lciv> N M P @NPHR
de [de] PRP @N<
pesquisa [pesquisa] <occ> <act-d> N F S @P<
não [não] ADV @>A
documentados [documentar] <vt> V PCP M P @N<
corretamente [correto] ADV @ADVL

FIGURE 3. MORPHOSYNTACTIC ANALYSIS OF THE DESCRIPTION OF A PRE-SELECTED CASE.

Step 5: In this step, the user adapts the solutions of similar cases, defined in step 4, for the new *problem*.

Step 6: Finally, the new case adapted is stored in the Case Base.

IV. Proof Of Concept

Table 2 shows the experimental results of our approach when two new real input *problems* were compared with a Case Base containing 46 cases, occurring in software projects of a company run by the federal government of Brazil. The retrieval of cases by attribute-value similarity, processed by CBR component, decreases the universe of cases to be processed by the NLP component. For **Case 1**, we had a 29% decrease and for **Case 2**, we had a 75% decrease. Furthermore, it was possible to corroborate that the selection by similarity of the descriptive texts of the *problem* is what, in fact, determines the similar cases for reuse of solutions. **Case 2** was, in fact, a new case and refining process performed by the NLP component correctly not selected any of the seven cases initially similar for reuse. However, it is noteworthy that the initial search by attributes, although not a determining factor, is important when the Case Base is large. This experiment also allowed us to identify a point of improvement in the NLP component: one of the three cases selected for reuse in **Case 1** was a false positive case.

The parameters used in this experiment were: $P_{Artifact}$ = 1, P_{Causer} = 3 e P_{Cause} = 5 (Step2); Cutoff value for measure Sim_A = 50% (Step 2); m = 2 (number of words in common) (Step 4). Iterations of this experiment were carried out by randomly varying the values of these parameters and, for the Case Base in question, the aforementioned values were those that presented the best results.

Table 2. Results of usage of the experience reuse tool.

Case	Attributes	Process Component	Selected Cases
1	**Artifact**: *Business Rule* **Causer**: *Person* **Cause**: *Wrong Business Rule Specified*	CBR Component	33
	Description: *Search functionality specified in wrong fields*	NLP Component	3
2	**Artifact**: *Vision Document* **Causer**: *Technology* **Cause**:*Vision Document changed*	CBR Component	7
	Description: *Changes in functionality made it impossible to use technology previously chosen*	NLP Component	0

V. Conclusion

This paper presents an ongoing approach to support the reuse of experiences in RE. The distinguishing feature of the approach is the use of AI techniques, specifically, Case Based Reasoning and Natural Language Processing techniques. The latter enables the representation of cases to take place in a more natural and complete manner, that is, by describing the case in natural language, whereby it is possible to make explicit the essence and details of each case. We executed experiments that corroborate our argument that the similarity by the textual description of the cases is more determining than the attribute-value similarity, making the process of reuse of solutions more effective and useful. As an evolution of this proposal, we are working on incorporating the semantic analysis of texts through a common-sense conceptual base in the Portuguese and English languages – InferenceNet [8], thus improving the accuracy of the proposed approach.

References

[1] Pereira, S. C. (2007) "Um estudo Empírico sobre Engenharia de requisitos em Empresas de Produtos de Software". Dissertação de Mestrado em Ciências da Computação. Centro de Informática, Universidade Federal de Pernambuco.

[2] Standish, Standish Group. (1994) The Chaos Report. http://www.standishgroup.com/sample_research/chaos_1994_1.php

[3] Joshi, S.R., McMillan, W.W. (1996). Case Based Reasoning Approach to Creating User Interface Components. Proceedings of the CHI '96 conference companion on Human factors in computing systems: common ground

[4] Gabineski, R., Lorensi, F. (2007) Sistemas Multiagente Baseados em Casos de Apoio á Gerência de Projetos. VIII Salão de iniciação Científica e Trabalhos Acadêmicos.

[5] Jani, H. M., Mostafa, S. A. (2007) Implementing Case-Based Resorsing Technique to Software Requirement Specifications Quality Analysis. International Journal of Advancements in Computing Technology. Volume 3, Number 1.

[6] Praehofer, H., Kerschbaummayr, J. (1999) Case-Based Resorsing techniques to support reusability in a requirement engineering and system desing tool. University Linz, departement os systems theory and information technology. Institute of Systems Science. Austria.

[7] Wangehheim, C.G., Wangehheim, A. (2003) Raciocínio Baseado em Casos. Editora Manole. 1º edição.

[8] Pinheiro, V., Pequeno, T., Furtado, V., Franco, W. (2010) InferenceNet.Br: Expression of Inferentialist Semantic Content of the Portuguese Language. In: T.A.S. Pardo et al. (eds.): PROPOR 2010, LNAI 6001, pp.90-99. Springer, Heidelberg.

[9] Bick, E. The Parsing System "Palavras". (2000) Automatic Grammatical Analysis of Portuguese in a Constraint Grammar Framework. Aarhus University Press. http://beta.visl.sdu.dk/visl/pt/index.php

Specification of Safety Critical Systems with Intelligent Software Agent Method

Vinitha Hannah Subburaj
Computer Science Department
Texas Tech University
Lubbock, Texas 79409
vinitha.subburaj@ttu.edu

Joseph E. Urban
Industrial Engineering Department
Texas Tech University
Lubbock, Texas 79409
joseph.urban@ttu.edu

Manan R. Shah
Computer Science Department
Texas Tech University
Lubbock, Texas 79409
manan.r.shah@ttu.edu

Abstract — **Specifying and developing critical components using intelligent software agent technology has been recent practice employed with aircraft software systems. There are approaches used in developing aircraft software components that face the problem of inconsistency and unreliability. This paper addresses the need for using an automated intelligent software agent method and a formal specification language to specify the components of aircraft software systems for developing reliable software. Formal methods in specification of a system and their advantages over the traditional software development process in context to current real-time safety critical systems have been addressed in this paper. The Descartes specification language, an executable specification language, is used to address the need for the use of formal methods in safety critical systems and the advantages over traditional approaches of development. The Descartes specification language was designed for use throughout the software life cycle. Descartes is used to describe concrete architectures of intelligent software agents with structure details and the operation of such agents. Properties of intelligent software agents that include learning, planning, past history of experience, and knowledge-level are specified using the Descartes specification language. The application of such formally specified intelligent software agents in the development of aircraft software system components is addressed in this paper.**

Keywords-component; aircraft software; formal specification language; intelligent software agents;

I. Introduction

Aircraft development is a complex, dynamic, and fascinating process which gives the descriptions of aircraft and component design [1]. The development methods usually deal with the aircraft conceptual, preliminary, and detail development activities. The introduction of conflicting requirements in aircraft development gives a view to improve understanding and the integration of sound overall development. The structures and propulsion, airframe systems, avionics, flight control, and weapons describe the interiors of aircraft. The aircraft system interiors range from structures to weapon systems through airframe systems, avionics systems and landing gears. Usually the aircraft costs much due to the areas of acquisition and operating costs. Also, the importance of good reliability and maintainability makes the cost high.

The Belief – Desire – Intention (BDI) architecture has been applied in this research effort to describe critical aircraft software components that involve intelligence [2]. The BDI architecture involves reasoning, deliberation, and means-ends reasoning. The reasoning aspect of the system helps to identify the action that needs to be performed while carrying out the goals of a system. A major concern that needs to be addressed is that different agents operate in different environments and hence require different reasoning strategies. Considering aircraft software component design also addresses the same issues of components operating in different environments.

The architectures have roots in understanding practical reasoning which consists of two parts, deciding what goals to achieve and how to achieve the goals. The decision process begins by trying to understand what options are available. After generating the alternatives, a choice needs to be made between the alternatives and commit to one of the chosen alternatives. The chosen options become intentions which determine an agent's actions. Intentions are crucial in the practical reasoning process. Intention has a major role in the BDI model. The basic components of a BDI architecture are beliefs, desires, intentions, and functions that represent its deliberations. A major issue is in striking a balance between the committed to and over committed to one's intentions. The BDI model is attractive for several reasons as intuitive and clear functional decomposition. The main difficulty is knowing how to efficiently implement these functions.

The remainder of the paper is organized as follows. Section 2 briefly describes the related work carried out in the area of specifying aircraft software components using formal methods. Section 3 gives the description of the Descartes specification language and the extensions made to the Descartes specification language for specifying the BDI architecture for intelligent software agents. The aircraft software components that were developed along with their specification details using the extended Descartes specification language is described in Section 4. Section 5 provides a summary and future research.

II. Related Work

Salas and Townsend [3] conducted a study that examined four existing frameworks against the requirements obtained from the Multidisciplinary Optimization Branch (MDOB) at the National Aeronautics and Space Administration (NASA). The Framework for Interdisciplinary Design Optimization (FIDO) project [3] investigated the use of a distributed, heterogeneous computing system to enhance communication, apply computer automation, and introduce parallel computing.

A major limitation in FIDO as determined by Salas and Townsend [3] was that it was used for a specific application. Hence, the sequences of processes become hard to code and difficult to modify. The lack of documentation makes FIDO inaccessible for the use of researchers. Salas and Townsend [3] identified the necessary frameworks during the evaluation process and the relevant MDO framework requirements were also briefly described in the paper.

Heitmeyer and Jeffords [4] translated software requirements of mission critical components for three NASA systems to specification, which were useful throughout the system life cycle; using a method called Software Cost Reduction (SCR). The automation process of a flight control system (AFCS), must meet strict fault tolerance requirements. Gobbo and Milli [5] describe formal specification for an analytical redundancy based fault tolerant flight control system. The specifications for the De Havilland DHC-2 general aviation aircraft was developed using relational algebra as the formal framework. The requirements are decomposed on a functional basis into elementary specification and then composition operators of relational algebras are used to build higher level requirements to develop the whole specifications. Requirements can be interpreted as a relationship among some relevant quantities.

The next section describes the Descartes specification language, a formal specification language that is executable. Existing work done on extending the Descartes specification language to specify complex systems has been described in the next section.

III. The Descartes Specification Language and the Extensions

The Descartes specification language was designed to be used throughout the software life cycle. The relationship between the input and the output of a system is functionally specified using this specification language [6]. Descartes defines the input data and output data and then relates them in such a way that output data becomes a function of input data. The data structuring methods used with this language are known as Hoare trees. These Hoare trees use three structuring methods namely direct product, discriminated union, and sequence. Direct product is the default and provides for the concatenation of sets of elements. Discriminated union is denoted by a plus sign (+) suffixed to the node name. Sequence is indicated by an asterisk (*) suffixed to the node name. By definition of Hoare trees, a sequence node is followed by a subnode. A single node can accommodate a sequence of direct product or a sequence of discriminated union.

Specifying reactive agents using the extended Descartes specification language was the first attempt made to specify agents using the Descartes specification language [7] [8]. Extensions to the Descartes specification language for supporting intelligent software agents were given by Subburaj and Urban [9]. The new constructs introduced by Subburaj and Urban capture the requirements of intelligent software agents using the Descartes specification language. Real time semantics can be specified using the Descartes specification language by the extensions introduced by Sung and Urban [8]. Some of the basic extensions already made to the Descartes specification language to specify intelligent software agents [7] that are also used to specify the aircraft software components are as follows:

1. estate;
2. astate; and
3. action

The above existing constructs are used in this paper to give a complete specification of intelligent software agents. In this paper, extensions to the Descartes specification language are based on the concrete BDI architecture proposed by Wooldridge [2]. The BDI architecture works on the principle of practical reasoning, wherein the actions to be performed are decided momentarily in order to achieve the goal. The motive to specify and develop critical aircraft software components can be realized using the extensions made to the Descartes specification language. The extended constructs to specify intelligent software agents based on the BDI architecture include:

1. belief;
2. belief revision function (brf);
3. option generation function (options); and
4. current options;

1) *Belief*

In order to represent an agent's current environment, belief has been used. The extensions already made to the Descartes specification language [7] were used to represent agents environmental states. The reserved word 'estate' will be used to specify the different possible environment states. Based on the agent state and environment state, the next action will be triggered. The following specification describes the cruise control to be in one of the two states either active or passive.

```
agent (CRUISE_CONTROL)_AGENT
        CRUISE_CONTROL
                'file'
        estate+
                ACTIVE
                PASSIVE
```

The agent and environment state information is important while specifying intelligent software agents.

2) *Belief revision function*

The second extension to Descartes is a new primitive, "belief revision function", which uses the inputs given and the current belief of agents in order to realize a new set of beliefs for the system. The belief revision function allows a user to refine and come up with a concrete set of beliefs to specify intelligent software agents. The belief desire function is based on the inputs passed on to the system and the agent's current information on its environment. The following example illustrates the belief revision function. The subnode that comes along with the belief revision function is astate that contains the agents environment information.

```
agent(AIRCRAFT SOFTWARE_INPUTS)_AGENT
        …
        belief _desire _function
                input
                        …
                astate
                        …
```

The above example gives the structure of a belief desire function in an agent system. Analysis and synthesis trees follow the decision construct to specify the intelligent agent system completely and correctly.

3) *Option generation function*

An option generation function is used to generate options, in other words, desires of the system based on environmental state and set of intentions. Intentions are nothing but the set of options that the system intends to achieve in the near future. An option generation function is specified under the "ogf" reserved word. This new extension to Descartes allows for specifying the option generation function in accordance with the environmental states. Consider the following example:

```
  ogf
        list_of_options_for_the_agent_system
        …
        estate
                'S0'
                'S1'
                'S2'
        intentions
                'I1'
                'I2'
                'I3'
```

The output from the "ogf" construct will be the set of options in accordance with the beliefs and set of intentions.

4) *Current options*

Current options represent the list of actions that are expected from an agent. Actions are the set of actions that an agent can perform based on an agents environmental state, desires, and intentions. The reserved word "action" was already added to the Descartes specification language to derive a specific action that an intelligent software agent performs under a certain circumstance. Based on the current environment, the set of beliefs framed, and the set of intentions, the corresponding action is taken. In this way of specification of an intelligent agent system, the environment states which holds the information of agents from the past, is used to respond to the surroundings in the form of actions. Action is represented through a series of analysis and synthesis trees. Consider the following example:

```
  action
              ACTION_BASED_ON_ENVIR_STATE
```

IV. Aircraft Software Component Specification

The extensions made to the Descartes specification language introduced in Section 4 have been used to write sample specifications for aircraft software components. The requirements described for Federal Aviation Regulation (FAR) climb in a multiengine aircraft [5] has been converted into specifications written using the extended Descartes specification language. The BDI architecture for specifying intelligent agent structures has been used to write specifications for the aircraft design using the extended Descartes constructs. An aircraft climb management system follows to meet the Federal Aviation Regulations.

1) *Sample specification 1*

In sample specification 1, using extended Descartes constructs, an aircraft climb management system is specified. Operations are divided into two main categories namely takeoff climb, and landing. A Hoare tree structure that uses analysis and synthesis trees, specifies the first segment of takeoff climb where the aircraft speed will be in Liftoff (LOF) mode with gear down and flaps in the takeoff position. The input and output to specifications 1 and 2 are described as follows. Input to the two specifications are the values of the current positions of the parameters of the aircraft management system and the output of the system will be the next position the aircraft needs to take based on the intelligent agent decisions.

```
agent (TURBINE_ENGINE_AIRCRAFT)_AGENT
        TURBINE_ENGINE_AIRCRAFT
         aircraft_sensors_input_signals *
          input
                          speed +
                                  FLOAT
                                  LOF
                          flap +
                                  takeoff
                                  up
                                  approach
                                  landing
          number_engine
                                  INTEGER
          min_climb_gradient
                                  FLOAT
          landing_gear +
                                  up
                                  down
        estate
                estate_name
                        'V_s0'
                        'V_s1'
                        'V_s2'
```

```
belief_revision _function
        INPUT
         SPEED
                FLAP
         NUMBER_ENGINE
         MIN_CLIMB_GRADIENT
         LANDING_GEAR
ogf +
        aircraft_on_air
        …
        aircraft_on_land
        …
action +
        takeoff +
         first_segment
          success_or_failure +
                        success
                                text
                        failure
                                text
         second_segment
                    …
         third_segment
                    …
          landing +
         go_ard_in_aprch_conf
          success_or_failure +
                        success
                                text
                        failure
                                text
         go_ard_in_landig_conf
                    …
return
        TAKEOFF +
         FIRST_SEGMENT
                  SUCCESS_OR_FAILURE +
                        SUCCESS
                                TEXT
                        FAILURE
                                TEXT

         SECOND_SEGMENT
                    …
         THIRD_SEGMENT
                    …

LANDING +

                GO_ARD_IN_APRCH_CONF
                    …

                GO_ARD_IN_LANDIG_CONF
                    …
```

V. Summary and Future Research

Applying formal methods while specifying aircraft software components that are safety critical, is an emerging field of science. This research effort has introduced extensions to the Descartes specification language using the BDI architecture for specifying aircraft software components. The extended Descartes allows specifying sophisticated systems like the aircraft software systems using the intelligent software agent architecture. Descartes, being an executable specification language, has been used in this research effort to analyze the input and output of sample aircraft component development. Application of formal methods to specify components during development of a component will result in a safe, reliable system outcome. The extensions made to the Descartes specification language have been justified to be appropriate for aircraft software development by writing specifications for such a component in this paper. From an engineering standpoint, this research effort will lead to future incorporation of formal methods in software development. Also, the future directions from this research will be an automated conversion of the low level specifications written using Descartes into high level development specifications using AUML. This transition will introduce a new perspective in the field of software engineering while designing and developing software products.

References

[1] Raymer, D. P., *Aircraft Design: A Conceptual Approach*, 4th ed.: AIAA Education Series, 2006.

[2] Wooldridge, M., "Intelligent Agents," in *A Modern Approach to Distributed Artificial Intelligence*.: MIT Press, 1999, pp. 27-78.

[3] Salas, A. O., and Townsend, J. C., "Framework Requirements for MDO Application Development," in *7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, St.Louis, MO, 1998.

[4] Heitmeyer, C. L., and Jeffords, R. L., "Applying a Formal Requirements Method to Three NASA Systems: Lessons Learned," in *Proceedings of IEEE Aerospace Conference*, 2007.

[5] Gobbo, D. D. , and M ili, A., "“Re-engineering Fault Tolerant Requirements: A Case Study in Specifying Fault Tolerant Flight Control Systems," in *Proceedings of the 5th IEEE International Symposium on Requirements Engineering*, Toronto, Canada, 2001, pp. 236-247.

[6] Medina, M. A., and Urban, J. E., "An Approach to Deriving Reactive Agent Designs from Extensions to the Descartes Specification Language," in *Proceedings of the 8th International Symposium on Autonomous Decentralized Systems*, Sedona, Arizona, 2007, pp. 363-367.

[7] Subburaj, V. H., and Urban, J. E., "Issues and Challenges in Building a Framework for Reactive Agent Systems," in *Proceedings of the 3rd International Workshop on Frontiers in Complex Intelligent and Software Intensive Systems*, 2010.

[8] Sung, K. Y., and Urban, J. E., "Real-time Descartes: a Real- time Specification Language," in *Proceedings of the 3rd Workshop on Future Trends of Distributed Computing Systems*, 1992, pp. 79-85.

[9] Subburaj, V. H., and Urban, J. E., "Intelligent Software Agent Design Issues with Extensions to the Descartes Specification Language," in *Proceedings of the 22nd International Conference on Software Engineering and Knowledge Engineering*, 2010, pp. 668-671.

Using the Results from a Systematic Mapping Extension to Define a Usability Inspection Method for Web Applications

Luis Rivero and Tayana Conte
Instituto de Computação, Universidade Federal do Amazonas (UFAM)
Manaus, AM - Brazil
{luisrivero,tayana}@icomp.ufam.edu.br

Abstract— **Usability is one of the most crucial factors in Web applications, allowing the successful usage of such systems. Many Usability Inspection Methods (UIMs) have been proposed to guarantee that Web applications provide a friendly, direct and easy to understand interface to their users. Nevertheless, some of these methods are not being used due to the lack of information about them. In this paper we describe the actual state of UIMs for Web applications through the extension of a systematic mapping study about Usability Evaluation Methods. Besides providing background knowledge to UIMs, our results showed that in order to meet the actual needs of the software development industry, emerging UIMs should: (a) find problems in early stages of the development process; (b) find specific problems and suggest solutions; and (c) provide automation. Using these features and usability criteria from other UIMs, we developed the Web Design Usability Evaluation (Web DUE) technique. The Web DUE aims to improve the quality of Web applications by allowing the usability evaluation of paper based prototypes by using pieces of Web pages called Web page zones. We have provided a proof of concept of the Web DUE by evaluating a paper based mock-up of a real Web application.**

Keywords-Usability Inspection Methods; Web Applications; Systematic Mapping Extension

I. INTRODUCTION

In recent years, Web applications development demand has grown considerably [1]. These applications are currently the backbone of business and information exchange, and are being used to present products and services to potential customers [2]. According to Matera et al . [6], the acceptability of such applications is determined by their degree of usability. If a Web application possesses poor usability, it will be quickly replaced by a more usable one as soon as its existence becomes known to the target audience [7].

Usability Inspection Methods (UIMs) have emerged as a cost-effective way to improve the usability of such systems [14]. According to Matera et al. [6], the software development industry has been investing in the development of a variety of UIMs to address Web usability issues. Nevertheless, companies are not using these methods [4]. This lack of usage can be the cause of low quality regarding the usability aspect within Web applications [9].

In [12], we extended a systematic mapping study on usability evaluation methods for Web applications by selecting and thoroughly analyzing the papers that addressed new UIMs. This systematic mapping extension allowed us to identify the state of art of UIMs for Web applications and to provide researchers and practitioners with a knowledge background for choosing a determined UIM.

In this paper, we have used the findings presented in [12] to suggest a set of desirable features that a new UIM should possess in order to meet the actual needs of the software development industry. Based on these features, we have proposed the Web Design Usability Evaluation (Web DUE) technique. This technique aims to guide inspectors through the evaluation of paper based prototypes by dividing Web pages into Web page zones. According to Fons et al. [3], Web page zones are pieces of Web pages with specific types of contents. We crafted the Web DUE by selecting usability criteria from UIMs within the studies in [12] and relating them to the Web page zones. Furthermore, this paper also presents a proof of concept of the Web DUE technique by using it to evaluate the usability of a prototyped Web page of the Journal and Event Management System (JEMS[1]).

This paper is organized as follows. Section II presents the background to Usability Inspection Methods. In Section III we summarize the planning and execution of the systematic mapping extension on UIMs for Web applications. Section IV shows the results from this extension, our findings and suggests a set of desirable features for emerging UIMs for Web applications. In Section V we show the Web DUE technique proposal, while Section VI provides a proof of concept by using it to evaluate the usability of a paper based prototype of a r eal Web application. Finally, Section VII presents our conclusions and future work.

II. USABILITY INSPECTION METHODS

The term usability is defined in the ISO 9241-11 [5] as "*the extent to which a product can be used by specified users to achieve specific goals with effectiveness, efficiency and satisfaction in a specified context of use*". Regarding Web applications, usability is one of the most relevant quality aspects because of its own features, as cited in [10]: "*Web applications are interactive, user-centered, hypermedia-based applications, where the user interface plays a central role*".

Many usability evaluation methods (UEMs) have been proposed in the technical literature in order to improve the usability of different kinds of software systems [4]. According to Rocha and Baranauska [14], UEMs are procedures

[1] https://submissoes.sbc.org.br/

composed by a set of well-defined activities that are used to evaluate the system's usability. UEMs are divided into two categories: (a) user testing, in which empirical methods, observational methods and question techniques can be used to measure usability when users perform tasks on the system; and (b) inspections, which make use of experienced inspectors to review the usability aspects of the software artifacts [2]. In this research, we focus on usability inspections as they can lower the cost of finding usability problems since they do not need any special equipment or laboratory [14].

The main generic UIMs that can be used to increase the system's usability and therefore its quality are: the Heuristic Evaluation [8], the Cognitive Walkthrough [11], and the Perspective-based Usability Inspection [15].

The Heuristic Evaluation, proposed by Nielsen [8], assists the inspector in usability evaluations using guidelines. The evaluation process consists of a group of evaluators who examine the interface using heuristics, which are rules that seek to describe common properties of usable interfaces.

The Cognitive Walkthrough, proposed by Polson et al. [11], is a method in which a set of reviewers analyze if a user can make sense of interaction steps as they proceed in a pre-defined task. In order to identify usability problems the inspectors ask questions to answer if: (a) the action is sufficiently evident; (b) the action will be connected with what the user is trying to do; and (c) if the user will understand the system's response.

According to Zhang et al. [15], it is difficult for an inspector to detect all kind of problems at the same time. Consequently, they proposed a usability inspection technique based on perspectives (Usability Based Reading - UBR). In the UBR, the inspector focuses in a sub-set of questions according to the usability perspective to find problems.

In the following Section we show how we carried out the extension of the systematic mapping on UIMs for the Web.

III. Extension of the Systematic Mapping about UIMs for Web Applications

Fernandez et al. [2] presented a systematic mapping on UEMs for Web applications. However, in order to thoroughly describe how UIMs for the Web had been applied, it was necessary to independently analyze them. In this Section, we briefly explain the planning and execution of the extension of the systematic mapping in [2]. In this extension, we extracted specific information from [2] regarding the new UIMs for Web applications. We used the obtained results to answer the following research question: "*What new Usability Inspection Methods have been employed to evaluate Web artifacts and how have these methods been used?*" Readers must take note that a thoroughly described version of the execution process of this extension can be found in [12].

Selection Process: Fernandez et al. [2] analyzed 206 papers about usability evaluation methods for Web applications and classified them into categories. We have used this classification as a starting point in the selection of papers. We selected papers that, according to [2], described new inspection methods for Web applications. From this initial set of papers we only selected the studies that thoroughly described UIMs at a mature stage. Consequently, we discarded papers that met at least one of the following exclusion criteria:

- Papers presenting usability problems and no methodology to identify them.
- Papers describing only ideas for new research fields.
- Papers presenting techniques with no description of their execution process.

Categorization of Studies: We created a set of research sub-questions to better address the state of art of UIMs for Web applications. We used the answers to these sub-questions to categorize the analyzed papers. Table I shows our research sub-questions, their motivation, and the possible answers that can be obtained when analyzing a selected research paper.

TABLE I. RESEARCH SUB-QUESTIONS, POSSIBLE ANDSWERS AND MOTIVATIONS FROM THIS SISTEMATYC MAPPING EXTENSION.

Research Sub-Questions and Answers	Motivation
Q1. Theoretical Basis Method: (a) Heuristic Evaluation (b) Cognitive Walkthrough (c) Perspective Based (d) Other Basis	To discover whether the Usability Inspection Methods for the Web have been developed considering well known Generic Usability Inspection Methods or whether they have been using new basis.
Q2. Type of Evaluated Artifact: (a) HTML code (b) Model (c) Application/Prototype	To discover which is the most commonly evaluated artifact in Usability Inspection Methods for the Web.
Q3.- Type of Application Evaluated by the Inspection Method: (a) Generic (b) Specific	To discover whether the Usability Inspection Methods for the Web have been crafted to find generic usability problems or usability problems of a specific type of Web application.
Q4.- Use of Inspectors in the Inspection Process: (a) Yes (b) No	To discover whether the Usability Inspection Methods for the Web have been automated to a point where inspectors are no longer necessary.

Execution: Fig. 1 shows how we executed this systematic mapping extension. From the initial set of 206 papers in Fernandez et al. [2], we selected 37 papers that, according to the classification in [2], presented new usability inspection methods for Web applications. However, as 5 papers were unavailable for download, we reduced the initial set to 32.

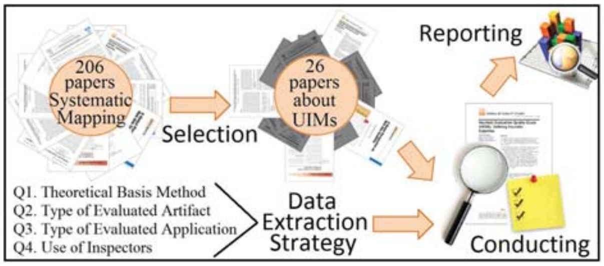

206 papers Systematic Mapping
Selection
26 papers about UIMs
Reporting
Q1. Theoretical Basis Method
Q2. Type of Evaluated Artifact
Q3. Type of Evaluated Application
Q4. Use of Inspectors
Data Extraction Strategy
Conducting

Figure 1. *Execution process of this Systematic Mapping Extension.*

After reading each study, we discarded 6 studies for meeting the exclusion criteria we defined in the selection process stage. The Selected Primary Studies List in this paper shows the 26 selected papers that we analyzed in this literature review. In the next Sections we explain how we used the data obtained from the categorization of studies, to address the current stage of UIMs for the Web and suggest a new UIM.

IV. RESULTS DISCUSSION AND IMPLICATIONS FROM THE SYSTEMATIC MAPPING EXTENSION

This Section summarizes the principal findings from the analysis of the results of this literature review. Furthermore, we used these findings to propose a set of d esirable features for emerging UIMs seeking to meet the actual needs of the software development industry.

A. Results and Principal Findings

Table II shows the classification of p rimary studies according to the research sub-questions we defined above. Readers must take note that the summation of the percentages of sub-questions Q1 and Q2 is over 100% as a paper can be classified in one or more answers. In this sub-section we will discuss our findings regarding each of the sub-questions.

Q1 - Theoretical Basis Method: Around 60% of t he reviewed papers based the new UIM for Web applications on already known usability inspection methods. UIMs based on Nielsen's [8] Heuristic Evaluation (27%), mainly focused on better describing or self explaining how or in which situation each heuristic could be applied. Regarding the use of the Cognitive Walkthrough (19%), two new inspection methods were developed: Blackmon's CWW described in papers S06, S07 and S08; and Filgueiras' RW described in S14. Studies S09, S12, S23 and S25 (15%) made use of perspectives to help focus on each usability attribute when carrying out the inspection. Moreover, the remaining techniques (58%) are being based on heuristics specifically proposed for the Web.

UIMs for the Web are based on: (a) generic usability inspection methods; and (b) new specific evaluation criteria for the Web. However, none of them can address all circumstances and types of Web artifacts. A combination of these methods can be used to enhance the evaluation results.

Q2 - Type of Evaluated Artifact: Around 77% o f the reviewed papers reported UIMs analyzing prototypes/systems. Inspectors carry out the evaluation process by analyzing the interaction provided by the prototype or product while executing a task on it. Moreover, 15% (S02, S21, S23 and S26) of the analyzed papers describe automated techniques in which HTML code was verified. Regarding Model analysis, 15% (S04, S18, S21 and S23) of the studies evaluated if the model met interaction rules within the Web domain.

The main artifacts used during the inspection process are: models, HTML code and prototype/application. Our results show that prototype/application is the most common evaluated artifact. However, the evaluation is being held with functional prototypes, which means that the cost of correcting usability problems is high. Therefore, there is a shortage of UIMs for the Web able to identify usability problems during the initial stages of the development process.

Q3 - Type of Web Application Evaluated by the Inspection Method: The results revealed that around 88% of the UIMs could be applied to any Web application. The remaining studies (S01, S05 and S24), around 12% of the selected UIMs, focused on a specific type of Web application. Allen et al. (S01) describe a paper based technique for medical Web applications. Basu (S05) proposes a new framework to evaluate e-commerce applications. In paper S24, Thompson and Kemp evaluated Web 2.0 applications.

Generic UIMs focus on finding usability problems that can be applied to every Web application. However, most of them do not provide feedback on how to treat a violation of usability. On the other hand, UIMs that evaluate specific types of Web applications provide evaluators with more data regarding that type of application. Our results show that there is a higher number of UIMs for generic Web applications compared to the number of UIMs for specific Web applications; and that there is a need for UIMs that suggest solutions for the identified problems.

TABLE II. RESULTS OF THE CLASSIFICATION ANALYSIS.

	Q1				Q2			Q3		Q4	
Paper	a	b	c d		a b		c	a	b	a	b
S01	X						X		X	X	
S02				X	X			X			X
S03				X			X	X		X	
S04				X		X		X		X	
S05				X			X		X	X	
S06		X					X	X		X	
S07		X					X	X		X	
S08		X					X	X		X	
S09			X	X			X	X		X	
S10	X						X	X		X	
S11	X						X	X		X	
S12	X		X				X	X		X	
S13				X			X	X		X	
S14		X					X	X		X	
S15				X			X	X		X	
S16		X					X	X		X	
S17	X						X	X		X	
S18				X		X		X		X	
S19				X			X	X		X	
S20	X			X			X	X		X	
S21				X	X X			X		X	
S22				X			X	X		X	
S23			X	X X	X			X		X	
S24	X						X		X	X	
S25			X	X			X	X		X	
S26				X	X			X			X
Total Studies	7	5	4	15	4	4	20	23	3	24	2
%	26.9	19.2	15.4	57.7	15.4	15.4	76.9	88.5	11.5	92.3	7.7

Categorization of Primary Studies
Q1.- Theoretical Basis Method: (a) Heuristic Evaluation
(b) Cognitive Walkthrough (c) Perspective Based (d) Other Basis
Q2.- Type of Evaluated Artifact: (a) HTML code (b) Model
(c) Application/Prototype
Q3.- Type of Application Evaluated by the Inspection Method:
(a) Generic (b) Specific
Q4.- Use of Inspectors in the Inspection Process: (a) Yes (b) No

Q4 - Use of Inspectors in the Inspection Process: Our results show that the automation of the inspection process is not yet possible in techniques involving judging and human interaction. Consequently, techniques using model and prototype analysis are not being automated (92%), but enhanced by using tools to provide inspectors with means of reducing evaluation effort. However, 8% of the reviewed studies described UIMs that did not use any inspectors at all.

There is a relationship between the UIM's evaluated artifact and the degree of automated process. UIMs evaluating HTML code are being fully automated. Nevertheless, their evaluated usability aspects are less than the usability aspects of UIMs that make use of inspectors.

B. *Meeting the needs of the Software Development Industry*

We used the sub-questions from this systematic mapping extension to suggest three features an emerging UIM should possess in order to meet the actual needs of the software development industry. We did not consider sub-question Q1 (Theoretical Basis Method) because its goal was to explore the bases of new UIMs for Web applications, rather than identifying research opportunities. We present three features and their relationship to our research sub-questions as follows:

Feature 1 - Ability to find problems in early stages of the development process: The results for sub-question Q2 indicated that there is a need for UIMs evaluating the usability of artifacts related to the early stages of the development process. Consequently, in order to reduce the cost of correcting usability problems new UIMs should be able to evaluate models or prototypes.

Feature 2 - Ability to find specific problems and suggest solutions: Research sub-question Q3 indicated that most Generic UIMs do not provide feedback on how to treat a usability problem once it is found. Consequently, emerging UIMs should be able to aid in both the identification and solution of usability problems.

Feature 3 - Automation: Our results for sub-question Q4 indicated that there is a shortage of automated UIMs. Therefore, in order to enhance the performance of the evaluation, new UIMs should be automated or provide assistance by means of a tool. Automated UIMs reduce the cost of carrying out inspections. Nevertheless, not every UIM can be automated. In this case, the UIM should provide means to reduce the inspector's effort.

V. USING FINDINGS TO DEFINE A NEW UIM

The Web Design Usability Evaluation (Web DUE) technique is an inspection method that proposes to meet the needs of the software development industry regarding UIMs for Web applications. Therefore, the Web DUE technique was crafted by adopting the suggested features obtained from this systematic mapping extension. In this Section, we relate each of the characteristics of the Web DUE technique to the suggested features.

The Web DUE technique **evaluates the usability of paper based low-fidelity prototypes (or mock-ups)**. This means that the evaluation can be carried out in the early stages of the development process (Feature 1).

The Web DUE technique **guides the evaluation through Web page zones**. Web page zones are pieces of Web pages with specific types of contents [3]. Table III shows the Web page zones used by the Web DUE technique and their contents. For each of the Web page zones we crafted a set of usability verification items based on the Heuristic Evaluation [8] and the Web Design Perspectives-Based Usability Evaluation – WDP [1] technique. The purpose of this usability verification items is to address usable characteristics within each of the Web page zones. Some of these verification items are shown in Table IV. Readers must note that the usability verification items also include examples in order to aid inspectors in the identification and solution of the identified problems. The complete list of the usability verification items per Web page zone can be found in [13].

Once the inspector verifies which Web page zones are contained within the paper based prototypes, he/she uses the usability verification items to find specific problems affecting the Web page zones that are related to the evaluated Web application (Feature 2). Furthermore, the inclusion of the violated usability verification items within the paper based prototypes can aid in the correction of the encountered usability problems.

TABLE III. WEB PAGE ZONES USED BY THE WEB DUE TECHNIQUE BASED ON [3].

Zone	Contents
Navigation	Navigation Links
System's State	Information about the application's state and how we got there.
Information	Information from the application's data base.
Services	Access to functionalities related to the information zone.
User Information	Information about the logged user.
Institution	Information about the institution that is responsible for the Web application.
Direct Access	Common Web functionalities (Login, Logout, Home).
Data Entry	Provides a form to input data to execute certain operations.
Custom	Domain-independent content
Help	Information about how to use the application and how it works.

A simplified version of the evaluation process of the Web DUE technique can be seen in Fig. 2. Initially, inspectors must divide the provided mock-ups into their respective Web zones (Stage 1). For each Web page zone the inspector verifies if the application meets usability rules by checking the technique's usability verification items (Stage 2). Finally, the non conformity of any verification item implies in a usability problem (Stage 3). In order to correct usability problems, the Web application must consider including the violated items.

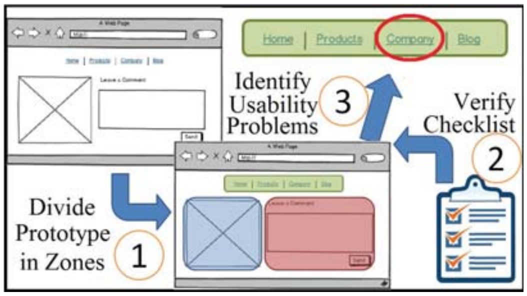

Home | Products | Company | Blog
Identify Usability Problems
3
Verify Checklist
2
Divide Prototype in Zones
1

Figure 2. *Simplified inspection process of the Web DUE technique.*

Another important feature is that the Web DUE technique **makes use of tools to reduce inspectors' effort**. As it involves inspectors' judgment in order to identify usability problems, the Web DUE is not fully automated. However, it aids inspectors during the evaluation process by using a tool that automatically presents the usability verification items checklists to aid inspectors during the inspection of paper based prototypes (Feature 3).

VI. Applying the Web DUE technique

In this Section we provide a proof of concept by evaluating the usability of a paper based prototype using the Web DUE technique. Readers must take note that we only show part of the example due to lack of space.

Part 1 of Fig. 3 shows the paper based prototype of a Web page of the Journal and Event Management System (JEMS). This Web page is used in the JEMS system to edit user data. In Fig. 3 part 2 we have identified all Web page zones: System's State Zone, Data Entry Zone and Navigation Zone. In Fig. 3 part 3 we have also zoomed in some of the components within the Web page zones of the mock-up. These components have been augmented because they represent usability problems within the paper based prototype of the Web page.

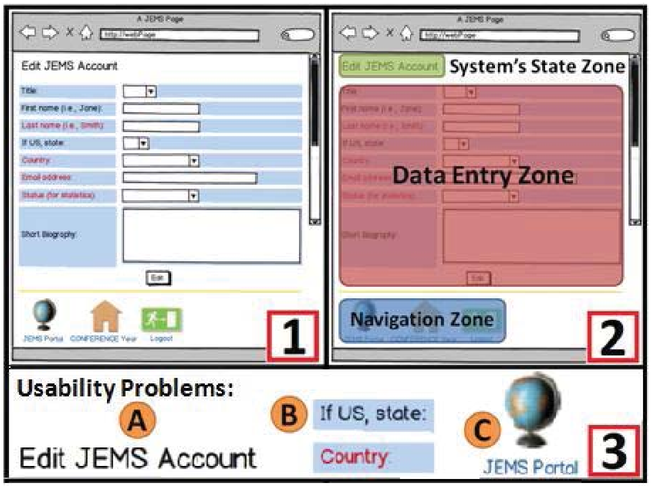


Figure 3. *Example of the Web DUE's Evaluation Process using a Mock-up.*

In Table IV we present some of the usability verification items of the Web DUE technique and their corresponding Web page zones. These verification items are nonconformities regarding the evaluated mock-up in Fig. 3. In other words, the paper based prototype of the Web page has not met these usability verification items. If we look at Fig. 3 part 3 and Table IV simultaneously, we can relate the nonconformity of the usability verification items in Table IV with the augmented elements A, B and C in Fig. 3. part 3. Readers must take note that we have shown one augmented element per Web page zone. We will address each of the encountered usability problems as follows.

In the System's State zone, the usability verification item 01 indicated that, despite showing the actual state of the system, the prototype does not show it logically (see Fig. 3 part 3 element A). In other words, the prototype does not show how the user reached that state. Furthermore, in the Data Entry zone, we identified nonconformity 02. This usability verification item indicates that the mock-up does not request data in a logical way. Asking for a country's state (even if this input data must be filled only for members from the US) before filling the user's country does not follow a logical order (see Fig. 3 part 3 element B). Finally, in the Navigation zone we encountered nonconformity 03. This verification item indicates that the symbols used within the navigation zone are difficult to understand. A user would find it difficult that the "globe" symbol would leave to the JEMS portal (see Fig. 3 part 3 element C).

TABLE IV. Usability Verification Items that the Evaluated Mock-Up does not meet within the Web DUE Technique.

N°	Web Page Zone	Usability Verification Items
01	System's State	The System's State is naturally and logically presented to the user. For example, when visualizing the system's state the user must be able to understand what path led him to that state.
02	Data Entry	The data to be input by the user are requested in a natural and logical order. In other words, when inputting data the sequence of input data must be logical.
03	Navigation	It is easy to understand the words and symbols used in the system. In other words, the user must be able to link the information being showed (labels or images) with what he/she is trying to do.

We have shown the simplified inspection process of the Web DUE technique by evaluating a low fidelity prototype of a real Web page. Readers must note that in order to evaluate the entire Web application, all Web pages within the Web application must be evaluated. Furthermore, in order to correct these problems, the prototype must integrate the violated verification items from Table IV.

VII. Conclusions and Future Work

This paper has discussed the results from a systematic mapping extension addressing UIMs for the Web. The analysis of 26 studies from [2] showed that in order to meet the actual needs of the software development industry, emerging UIMs should posses the following features: (a) find problems in early stages of the development process; (b) find specific problems and suggest solutions; and (c) provide automation or assistance to reduce the inspector's effort.

We used these features to propose the Web DUE technique that guides inspectors through the inspection process using Web page zones. The Web DUE also seeks to evaluate usability attributes in early stages of the development process by evaluating paper based prototypes. We have presented a proof of concept by evaluating a mock-up of a real Web page.

We hope that our findings will be useful to provide an outline to which usability evaluation methods can be applied. We also hope that the set of desirable features for emerging UIMs become adopted in new UIM proposals for Web applications. Moreover, as future work, we pretend to carry out empirical studies in order to evaluate the feasibility of the Web DUE technique, and verify if the tool support influences in a positive way in the results of the evaluation. Furthermore, we pretend to use the results of the studies to refine the technique and understand how it will be used by inspectors in the context of a real development environment.

ACKNOWLEDGMENTS

We would like to acknowledge the support granted by CAPES process AEX 4982/12-6.

REFERENCES

[1] T. Conte, J. Massollar, E. Mendes and G. Travassos, "Web usability inspection technique based on d esign perspectives," IET Software, Volume 3, Issue 2, 2009.

[2] A. Fernandez, E. Insfran and S. Abrahao, "Usability evaluation methods for the Web: A systematic mapping study," Information and Software Technology, Volume 53, Issue 8, 2011.

[3] J. Fons, V. Pelechano, O. Pastor, P. Valderas, and V. Torres, "Applying the OOWS model-driven approach for developing Web applications: The internet movie database case study," In: G. Rossi, D. Schwabe, L. Olsina and O. Pastor, Web Engineering: Modeling and Implementing Web Applications, Springer, 2008.

[4] E. Insfran and A. Fernandez, "A Systematic Review of Usability Evaluation in Web Development," Proc. Second International Workshop on Web Usability and Accessibility, New Zealand, 2008, pp. 81-91.

[5] International Organization for Standardization, ISO/IEC 9241-11: Ergonomic Requirements for Office work with Visual Display Terminals (VDTs) – Part 11: Guidance on Usability, 1998.

[6] M. Matera, F. Rizzo and G. Carughi, "Web Usability: Principles and Evaluation Methods," In: E. Mendes and N. Mosley, N, Web Engineering, Springer, 2006.

[7] E. Mendes, N. Mosley and S. Counsell, "The Need for Web Engineering: An Introduction. Web Engineering," In: E. Mendes and N. Mosley, Web Engineering, Springer, 2006.

[8] J. Nielsen, "Finding usability problems through heuristic evaluation," Proc. CHI'92, UK, 1992, pp. 373-380.

[9] J. Offutt, "Quality attributes of Web software applications," IEEE Software: Special Issue on Software Engineering of Internet Software, Volume 19, Issue 2, 2002.

[10] L. Olsina, G. Covella and G. Rossi, "Web Quality," In: E. Mendes and N. Mosley, Web Engineering, Springer, 2006.

[11] P. Polson, C. Lewis, J. Rieman and C. Wharton, "Cognitive walkthroughs: a method for theory-based evaluation of user interfaces," International Journal of Man-Machine Studies, Volume 36, Issue 5, 1992.

[12] L. Rivero and T. Conte, "Characterizing Usability Inspection Methods through the Analysis of a Systematic Mapping Study Extension," Proc. IX Experimental Software Engineering Latin Workshop, Argentina, 2012.

[13] L. Rivero and T. Conte, "Web DUE technique: Usability Verification Items per Web Page Zone," Technical Report RT-USES-2012-001, 2012. Available at: http://www.dcc.ufam.edu.br/uses/index.php/publicacoes/cat_view/69-relatorios-tecnicos.

[14] H. Rocha and M. Baranauska, "Design and Evaluation of Human Computer Interfaces," (Book), Nied, 2003. *(in Portuguese)*

[15] Z. Zhang, V. Basili and B. Shneiderman, "Perspective-based Usability Inspection: An Empirical Validation of Efficacy," Empirical Software Engineering, Volume 4, Issue 1, 1999.

SELECTED PRIMARY STUDIES LIST

[S01] M. Allen, L. Currie, S. Patel and J. Cimino, "Heuristic evaluation of paper-based Web pages: A simplified inspection usability methodology," Journal of Biomedical Informatics, Volume 39, Issue 4, 2006.

[S02] D. Alonso-Rios, I. Vazquez, E. Rey, V. Bonillo and B. Del Rio, "An HTML analyzer for the study of Web usability," Proc. IEEE International Conference on Systems, Man and Cybernetics, USA, 2009, pp. 1224-1229.

[S03] C. Ardito, R. Lanzilotti, P. Buono, and A. Piccinno, "A tool to support usability inspection," Proc. Working Conference on Advanced Visual Interfaces, Italy, 2006, pp. 278-281.

[S04] R. Atterer and A. Schmidt, "Adding Usability to Web Engineering Models and Tools," Proc. 5th International Conference on Web Engineering, Australia, 2005, pp. 36-41.

[S05] A. Basu, "Context-driven assessment of commercial Web sites," Proc. 36th Annual Hawaii International Conference on System Sciences, USA, 2003, pp. 8-15.

[S06] M. Blackmon, P. Polson, M. Kitajima and C. Lewis, "Cognitive walkthrough for the Web," Proc. SIGCHI Conference on Human Factors in Computing Systems, USA, 2002, pp. 463-470.

[S07] M. Blackmon, M. Kitajima and P. Polson, "Repairing usability problems identified by the cognitive walkthrough for the Web," Proc. SIGCHI Conference on Human Factors in Computing Systems, USA, 2003, pp. 497-504.

[S08] M. Blackmon, M. Kitajima and P. Polson, "Tool for accurately predicting Website navigation problems, non-problems, problem severity, and effectiveness of repairs," Proc. SIGCHI Conference on Human Factors in Computing Systems, USA, 2005, pp. 31-40.

[S09] D. Bolchini and F. Garzotto, "Quality of Web Usability Evaluation Methods: An Empirical Study on MiLE+," Proc. International Workshop on Web Usability and Accessibility, France, 2007, pp. 481-492.

[S10] C. Burton and L. Johnston, "Will World Wide Web user interfaces be usable?," Proc. Computer Human Interaction Conference, Australia, 1998, pp. 39-44.

[S11] J. Chattratichart and J. Brodie, "Applying user testing data to UEM performance metrics," Proc. of the Conference on Human Factors in Computing Systems, Austria, 2004, pp. 1119-1122.

[S12] T. Conte, J. Massollar, E. Mendes and G. Travassos, "Web usability inspection technique based on design perspectives," IET Software, Volume 3, Issue 2, 2009.

[S13] M. Costabile and M. Matera, "Guidelines for hypermedia usability inspection," IEEE Multimedia, Volume 8, Issue 1, 2001.

[S14] L. Filgueiras, S. Martins, C. Tambascia, and R. Duarte, "Recoverability Walkthrough: An Alternative to Evaluate Digital Inclusion Interfaces," Proc. Latin American Web Congress, Mexico, 2009, pp. 71-76.

[S15] P. Fraternali, and M. Tisi, "Identifying Cultural Markers for Web Application Design Targeted to a M ulti-Cultural Audience," Proc. 8th International Conference on Web Engineering, USA, 2008, pp. 231-239.

[S16] Y. Habuchi, M. Kitajima and H. Takeuchi, "Comparison of eye movements in searching for easyto-find and hard-to-find information in a hierarchically organized information structure," Proc. Symposium on Eye Tracking Research & Applications, USA, 2008, pp. 131-134.

[S17] S. Kirmani, "Heuristic Evaluation Quality Score (HEQS): Defining Heuristic Expertise," Journal of Usability Studies, Volume 4, Issue 1, 2008.

[S18] F. Molina and A. Toval, "Integrating usability requirements that can be evaluated in design time into Model Driven Engineering of Web Information Systems," Advances in Engineering Software, Volume 40, Issue 12, 2009.

[S19] M. Moraga, C. Calero and M. Piattini, "Ontology driven definition of a usability model for second generation portals," Proc. 1st International Workshop on Methods, Architectures & Technologies for e-Service Engineering, USA, 2006.

[S20] A. Oztekin, A. Nikov and S. Zaim, "UWIS: An assessment methodology for usability of Web-based information systems," Journal of Systems and Software, Volume 8, Issue 12, 2009.

[S21] L. Paganelli and F. Paterno, "Automatic reconstruction of the underlying interaction design of Web applications," Proc. 14th International Conference on Software Engineering and Knowledge Engineering, Italy, 2002, pp.439-445.

[S22] P. Paolini, "Hypermedia, the Web and Usability issues," Proc. IEEE International Conference on Multimedia Computing and Systems, Italy, 1999, pp. 111-115.

[S23] O. Signore, "A comprehensive model for Web sites quality," Proc. 7th IEEE International Symposium on Web Site Evolution, Hungary, 2005, pp. 30-36.

[S24] A. Thompson and E. Kemp, "Web 2.0: extending the framework for heuristic evaluation," Proc. 10th International Conference NZ Chapter of the ACM's Special Interest Group on Human-Computer Interaction, New Zeland, 2009, pp. 29-36.

[S25] L. Triacca, A. Inversini and D. Bolchini, "Evaluating Web usability with MiLE+," Proc. 7th IEEE International Symposium on Web Site Evolution, Hungary, 2005, pp. 22-29.

[S26] J. Vanderdonckt, A. Beirekdar, and M. Noirhomme-Fraiture, "Automated Evaluation of Web Usability and Accessibility by Guideline Review," Proc. 4th International Conference on Web Engineering, Munich, 2004, pp. 28-30.

Improving a Web Usability Inspection Technique through an Observational Study

Priscila Fernandes, Tayana Conte
Grupo de Usabilidade e Engenharia de Software (USES)
Universidade Federal do Amazonas
Manaus, Brazil
{priscila.fernandes, tayana}@icomp.ufam.edu.br

Bruno Bonifácio
Nokia Institute of Technology - INdT
Manaus, Brazil
{bruno.bonifacio}@indt.org.br

***Abstract*— Given the growth in the usage of Web Applications, the usability of these applications has become a key success factor. Several technologies have been developed to evaluate and improve this quality factor. However, the usability inspections results still depend on the inspector's experience. We have proposed a Web usability inspection approach, called WE-QT (Web Evaluation – Question Technique), a question based technique that aims to reduce the difficulties of inspectors with little knowledge of usability. We are following an experimentation-based methodology to support its development and improvement. This paper presents an observational study, aimed at eliciting how inspectors apply the WE-QT technique. We discuss the quantitative and qualitative results and their impact on improving the WE-QT. Results indicated that our technique assists novice inspectors uncovering usability problems effectively; despite qualitative data indicate the need for improvement.**

Keywords – web application, usability evaluation, inspection technique; observarional study; qualitative analysis

I. INTRODUCTION

The increasing popularity of Web applications has allowed an intensive use of these applications in current society [1]. The success of Web applications can be determined by two features: their fast evolution and their usability [2]. Usability is considered to be one of the most important quality factors for Web applications, along with others such as reliability and security [3].

The acceptability of Web applications relies strictly on the usability of the applications [4]. Web applications with poor usability will be quickly replaced by other ones more usable, as soon as its existence becomes known to the target audience [5]. Still, users often face errors while using these applications, caused by the not intuitive interfaces [6]. Therefore, improving usability of Web application can substantially minimize the users' interaction difficulty and improve the quality of these applications [7].

The challenge of developing Web applications with more intuitive interface has made the usability evaluation of these applications an important research area. Several methodologies to ensure a good usability specific for Web applications have been proposed, some of them based on user testing and others based on inspections performed by experts [8]. However, usability evaluations on Web development processes are often avoided by developers or companies due to their lack of experience in the field. Despite the high demand for usability evaluations of Web applications, developing methods to assist novice inspectors detecting defects without compromising the inspection result is not a simple task. According to Conte *et al.* [9], inspectors' skills such as experience on usability and inspection can affect the outcome of the inspection.

For this reason, we have proposed a new usability inspection technique, called WE-QT (*Web Evaluation Question-Technique*) [10]. The WE-QT technique was specifically tailored for usability evaluation of Web applications for novice evaluators. It uses a question based approach to guide the inspectors uncovering usability problems. The main goal of our solution is to minimize the difficulties and efforts of software developers with little knowledge in usability when executing inspections.

To support the development and validation of the WE-QT technique, we adopted the experimental methodology presented in [11]. This methodology comprises four stages: 1) feasibility studies to determine the usage possibility of the technology; 2) observational studies to improve the understanding of the technology, that aims to answer the question "*Do the steps of the process make sense?*"; 3) case studies in real lifecycle to characterize the technology application during a real lifecycle, and; 4) case studies in industry to identify if technology application fits into the industrial setting. Following this methodology, Fernandes *et al.* [10] presented an overview of the development of our technique and the feasibility study results, including discussions on how the methodology guided this phase of its development.

This paper describes the conducted observational study, aimed at eliciting how inspectors apply the WE-QT technique. We analyzed the qualitative data using coding procedures [12]. We discuss the results of the quantitative and qualitative data analyses and their impact in the improvement of the WE-QT technique. The results show that our technique assists novice inspector detecting usability defects effectively.

The remainder of this paper is structured as follows: Section 2 presents some background information on Web usability evaluation. Section 3 presents our technique, as the results of a feasibility study aimed at improving the technique. In Section 4, the observational study to evaluate our technique is discussed in detail, including our goals and experimental design. In Section 5 and 6 the results obtained by a quantitative and qualitative analysis are presented, respectively. Finally, our conclusions and future work are given in Section 7.

II. RELATED WORK

According to Offutt [3], one of the three quality criteria on the dominant Web development is usability. ISO 9241 standard [13] defines usability as “the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use”. Although a lot is known concerning the development of usable Web applications, many of these applications still do not meet most customers’ usability expectations [8].

Due to the importance of usability, the software development industry is investing in techniques and methods for the design and evaluation of web applications aiming at improving the interaction quality [4]. Usability evaluation methods can be divided into two categories [14]: Usability Inspections, evaluation methods based on experts’ analyses; and Usability Tests, in which usability defects are discovered by observation and interaction with users while they perform tasks or provide suggestions about the interface design [4].

According to Fernandez *et al.* [8], most of the published studies concerning Web usability evaluation are based on user participation, while inspections methods are being less applied, even being naturally less expensive, since it does not need any special equipment [4]. This scenario indicates some research opportunities on usability inspections techniques, aiming to reduce the cost of evaluations and encouraging a more frequent usage of usability evaluation in the industrial environment.

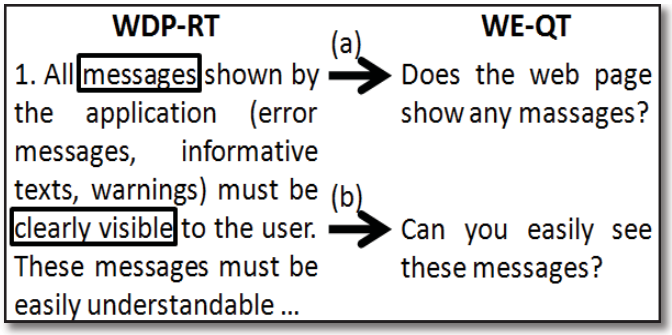
Several usability inspection techniques specific to web applications have been proposed [6, 9, 15, 16, 17, 18, 19, 20]. Conte *et al.* [9] proposed the WDP technique (Web Design Perspective-Based Usability Evaluation Technique). Despite the feasibility of the WDP technique to detect usability defects on web applications, novice inspectors had difficulties using the technique caused by lack of skills such as experience on usability and inspection, which can affect the outcome of the inspection [9]. Aiming to guide novice inspectors uncovering usability problem, Gomes *et al.* [6] proposed the WDP-RT (Web Design Perspectives-based Inspection – Reading Technique). The results of empirical studies to evaluate the WDP-RT technique indicated that, despite the WDP-RT technique helping novice inspectors finding usability problems efficiently and effectively, the inspectors still have difficulty on applying it [21].

The need to provide an easier usability inspection approach for novice inspectors motivated our research. Therefore, we proposed a new usability inspection technique for inspectors with little knowledge on usability and inspections. Our technique is called WE-QT (Web Evaluation Question Technique), and we present it in the next Section.

III. WE-QT (WEB EVALUATION – QUESTION TECHNIQUE)

In order to provide an easier and effective usability inspection approach for novice inspectors, we evolve the WDP-RT technique into another type of inspection technique: the question based approach, WE-QT [10].

The WE-QT technique consists of a set of questions that guide inspectors uncovering usability problems. The question based approach provided by WE-QT simplifies the inspection process, aiming to reduce the difficulties of the novice inspectors executing the inspection. Our technique does not require training on usability, inspection or on the technique itself before utilizing it. The WE-QT technique hides the Web perspective concepts used on WDP-RT, as well as any other information that is not being needed at a certain time of the inspection, aiming not to confuse the inspectors with irrelevant information. The questions that compose our technique were developed extracting the main goal from the instructions of WDP-RT and converting them into questions (Fig. 1).



1. All messages shown by the application (error messages, informative texts, warnings) must be clearly visible to the user. These messages must be easily understandable ...

Does the web page show any massages?

Can you easily see these messages?

Figure 1. Mapping: WDP-RT’s instructions into the WE-QT’s questions

The questions are classified into Decision Questions (DQ) and Sub-Questions (SQ). The DQs are responsible for verifying the elements existence, while SQs are responsible for the evaluation of these elements [10]. Item (a) of Fig. 1 illustrates the mapping from an instruction of the WDP-RT technique into a DQ of the WE-QT. Once we mapped a DQ, the features that must be evaluated regarding the element of DQ were extracted and turned into a SQ (item (b) of Fig. 1). Depending on the inspector’s answers to the questions, according to the mapping of WE-QT, the DQs define if the SQs will be displayed. Therefore, the inspection is customized by the elements of the application, due to the adaptive flow of the questions. This process is automated by a support the tool [10]. Table I shows an extract of the first version of the WE-QT technique.

TABLE I. EXTRACT OF THE WE-QT TECHNIQUE - FIRST VERSION

Id	Question	Mapping	
		Yes	*No*
0	Does the page show any messages?	1	10
1	Can you easily see the messages?	2	2
2	Can you easily understand the messages?	3	3
3	Are the messages consistent with the local culture?	4	4
4	Do the messages follow the visual pattern of the application?	5	5
⋮	⋮	⋮	⋮
10	Does the page inform you in which part of the application you are at?	11	13
11	Can you easily see this information?	12	12
12	Can you easily understand this information?	13	13
13	Are the information and options provided by the page displayed in a natural and logic order?	14	14

Following the experimental methodology (see Section 1), we executed a feasibility study in September 2010 to evaluate the first version of our technique [10]. Its goal was observe the feasibility of the WE-QT technique regarding efficiency (the ratio between the number of detected defects and the time spent in the inspection process); effectiveness (the ratio between the number of detected defects and the total number of existing); and inspectors' perception of our technique. Subjects were 12 undergraduate students attending a Software Engineering course at the Federal University of Amazonas. The results suggested that the WE-QT technique is feasible to assist novice inspectors detecting usability defects. The feasibility study is detailed in [10]. The results of this study were also used as input to further improve the WE-QT technique, resulting in its second version (WE-QT v2). Summarizing, the improvements made in the technique were: (1) grouping questions related to the same object by transforming them into multiple answer questions; (2) removing redundant questions; (3) allocating certain questions to be displayed only once during the inspection flow; and (4) detailing certain terms of the technique. Table II shows an extract of the WE-QT v2.

TABLE II. EXTRACT OF THE WE-QT TECHNIQUE - SECOND VERSION

Set	Question	Mapping		
		Yes	*No*	*Next*
1	Does the page show any messages (error, warning messages,...)?	2	7	---
2	Regarding the messages:	---	---	3
	I cannot see the messages easily			
	I cannot understand the messages easily			
	The messages do not follow the visual pattern of the application (same colors, text fonts,…)			
⋮	⋮	⋮	⋮	⋮
7	The page:	---	---	8
	Is not what I expected			
	Does not have a pleasurable interface, in general			
	Has texts, images, buttons that I cannot easily see			
8	Does the page inform you in which part of the application you are at?	9	10	---
9	Regarding your location:	---	---	10
	I cannot see this information easily			
	I cannot understand this information easily			
10	The page:	---	---	11
	Does not emphasize important information to reach my goal			
	Emphasizes irrelevant information to my goal			
	Does not support frequent performed tasks			
	Does not provide an option to return to the previous page (do not consider the browser option)			
	Does not provide an option to return to the home page			
11	Regarding the contents of the page:	---	---	12
	The information, options and menus of the page are not displayed in a natural and logic order			
	The words, texts, figures and symbols from the page are not easy to be understood			
	The page does not provide the definition of images, symbols and unusual words where they are displayed			
	The language of the page is not in accordance with its topic			

IV. THE OBSERVATIONAL STUDY

Since the results obtained from the visibility study indicated the WE-QT's feasibility, we went one step further on following the experimental methodology, in order to elicit the process used by the usability inspectors when applying the technique during a usability evaluation. With this purpose we performed an observational study in October 2011 using the second version of the WE-QT technique. Our aim was to understand deeply the WE-QT process, so we did not compare the WE-QT with any other technique. The observational study is detailed below:

A. Study Planning

The goal of this observational study, presented using the GQM paradigm [22], can be formalized as:
Analyze: Web Evaluation Question-Technique (WE-QT v2)
For purpose of: understanding
With respect to: how the inspectors apply the WE-QT technique
From the point of view: of Software Engineering researchers
In the context of: the evaluation of the usability of a real Web application by undergraduate and postgraduate students.

In this study, we gathered two types of qualitative data: observational and inquisitive data. The observational data was collected during the inspection process, we used the Think-Aloud Method that is an observation technique in which the user is asked to 'think out loud' and describe the activities he/she is performing as they are undertaken. This technique helps understanding the difficulties experienced by the user and also how they apply the technique [23]. We also used a usability testing tool to capture the inspection section of each inspector and to assist the collection of the perceptions of each inspector during the evaluation. Inquisitive data was gathered at the completion of inspection using follow-up questionnaires.

Despite the main goal of this study being eliciting the process used by the inspectors when applying our technique, we decided that executing quantitative analysis would provide us with useful data. Therefore we used the effectiveness and efficiency indicators, defined as: effectiveness (ratio between the number of detected problems and the total of existing problems); and efficiency (ratio between the number of detected problems and the time spent in the inspection). These indicators have been employed in other studies to evaluate Web application usability inspection methods as well [9, 10, 24].

The object of this study was the Graduate student Portal (www.ppgi.ufam.edu.br) of Federal University of Amazonas. This application is used to support M.Sc. and PhD students as well as teachers with academic matters. Three relevant use cases were defined: (1) User authentication; (2) Registration in two offered courses; and (3) Update personal data. Seven subjects participated in the study, one undergraduate student and six M.Sc. students. According to Nielsen [25], three to five inspectors from each user profile are enough to detect most of the usability defects.

During the planning stage we also elaborated the Inspection Guide; the Consent Form; the Post Inspection Questionnaire to collect the inspectors' opinion about the technique; and the Characterization Form to characterize subjects' experience in inspection and in usability. Table III shows the data obtained from Characterization Form.

B. *Study Execution*

The inspection phase was carried out with each subject individually. They were provided with the instruments to accomplish the inspection and received instructions about the evaluation by the moderator. Once the inspector understood the procedures, the inspection process began. One researcher acted as the observer, being responsible for conducting the detection phase. It is worth to mention that the subjects did not receive training on WE-QT technique, usability or inspections.

At the end of the inspection phase, a meeting attended by the researchers and a control group formed by usability specialists took place. A list of all usability problems identified by the subjects was discriminated to classify these problems into real defects or false positives. The authors of the technique did not influence the discrimination.

V. Quantitative Data Analysis

As a result of the inspection, we identified a total of 85 usability defects. We computed the number of detected defects, time spent during the inspection phase, efficiency and efficacy for each inspector. Table III presents these results including and their experience level.

TABLE III. Summary of the inspection results per subject

N°	Usability Experience	Inspection Experience	Defects	Time (hours)	Defects /Hours	% Founds Defects
01	None	None	25	1,85	13,51	29,41
02	None	None	48	0,77	62,61	56,47
03	Low	Low	26	0,67	39,00	30,59
04	Low	None	27	0,92	29,45	31,76
05	Medium	High	22	0,72	30,70	25,88
06	Low	Medium	21	0,87	24,23	24,71
07	Medium	Low	35	1,47	23,86	41,18

Table IV shows the averages for the time, and effectiveness and efficiency indicators. Regarding the efficiency indicator, inspectors detected an average of 31.91 defects per hour using the WE-QT technique.

TABLE IV. Discrepancies and detection time

Total Known Defects	Effectiveness Average Indicator (%)	Average Time (Hours)	Efficience Average Indicator
85	34,29%	1,04	31,91

Regarding the effectiveness indicator, each inspector found an average of 34.29% of known defects. Considering the time of inspection, each inspector spent an average of 1.04 hours (or 62 minutes) detecting the usability problems.

As mentioned before, despite the goal of this study being eliciting the process used by the inspectors when applying our technique, the quantitative analysis results provide an important opportunity to evaluate the performance of the technique. Considering the effectiveness indicator in the last study [10], the WE-QT technique resulted in 29,37%, while the new version of WE-QT technique resulted in 34,29% (see Table IV). These results are an indication that the WE-QT v2 is more effective than its first version. Still considering this indicator, we compared our results to the results of a study presented in [6]. According to Gomes *et al.* [6], the effectiveness indicator of the WDP technique resulted in 13% and WDP-RT technique resulted 29%. This indicates that our technique is considerably more effective than WDP and lightly more effective than WDP-RT. As the number of usability defects depends on the application, it is not recommended to compare the efficiency indicator with previous studies.

VI. qualitative Data Analysis

We analyzed the inquisition data using coding procedures [12] in order to understand the subjects' perception about the technique. We a lso merged the inquisition data with the observational data.

At the end of this analysis, the coding processes produced altogether 21 codes which were associated to 3 categories: Positive Aspects of the technique, Negative Aspects of the technique and Improvements Suggestions. Although the study's goal was to identify how the inspectors apply our technique, the Post Inspection Questionnaire had questions about the appropriateness and ease of use of the technique, which was useful to provide us with the subjects' perception of the technique.

Regarding the category "Positive aspects of the technique", we identified codes as: "It is simple and easy to apply the technique"; "Using the technique contributes to detect a greater number of usability problems" and "I would use and recommend the WE-QT technique".

Concerning the category "Negative aspects of the technique" (Fig. 2) we identified the code "The questions have ambiguous items", we observed that the subject had doubt with 38% of the questions of the WE-QT technique. Another cited code is "There is no reference to pop-ups and some elements present in Web applications", the observation data show that the technique also does not cover elements such as logins authentication, despite this element being evaluated as regular data forms, there is n o specific evaluation criteria for its features. The code "It was not trivial knowing what kind of answer to provide to the questions" was considered as a delay factor, and it also indicates that the technique requires more guidance. The code "Using the technique is a little tiring" was considered a harm factor to the inspector enjoyment when using our technique.

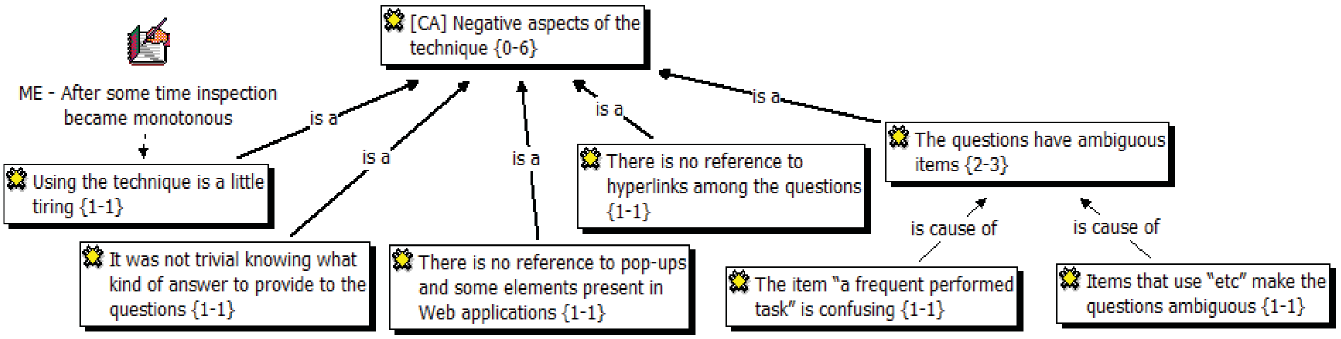

[CA] Negative aspects of the technique {0-6}
ME - After some time inspection became monotonous
Using the technique is a little tiring {1-1}
It was not trivial knowing what kind of answer to provide to the questions {1-1}
There is no reference to pop-ups and some elements present in Web applications {1-1}
There is no reference to hyperlinks among the questions {1-1}
The questions have ambiguous items {2-3}
The item "a frequent performed task" is confusing {1-1}
Items that use "etc" make the questions ambiguous {1-1}
is a
is a
is a
is a
is a
is cause of
is cause of

Figure 2. Negative aspects of the technique

About the category "Improvements suggestions" (Fig. 3) we identified the code "The technique should have an option so the user could visualize the symbol that represents the mandatory fields", indicating the need for more exemplification. Even the items being described on the questions, it was not enough to assist novice inspectors to identify the elements to be evaluated; it strengthens the need for more efficient ways to exemplify the items, as cited in the code "The technique should provide examples that would help understanding each question". Another code identified in this category was "An improvement should be made on how to apply the technique, so it does not result in lack of attention when detecting problems", we observed that applying the technique was the main cause of doubts among the subjects. Information on the inspection flow was provided to the subjects, however all of them had difficulty executing the inspection.

Knowing that problem detection must be executed for every page of the application, answering the questions of the technique and in parallel observing the interface; the main flows we identified were: complete the task and evaluate only the last page; complete the task and execute a single evaluation to all the pages; and explore different pages (ignoring the tasks) and answer each question to each visited page. This shows that the WE-QT technique is not applied correctly by the inspectors and indicates that the inspection flow must be better explained and detailed.

This observational study provided us with information on how the inspectors apply the WE-QT technique. That allowed us not only to elicit how inspectors apply our technique but to deeply identify flaws and w eakness in it. We obtained important feedback to improve the WE-QT technique, as it pointed out specific problems in the technique.

Threats to Validity

As in all studies, there were various threats that could affect the validity of our results. In this Section, we discuss those threats; break them down into the following categories [26]: internal, external, conclusion and construct.

Internal validity: in our experiment, we considered two main threats that represent a risk for an improper interpretation of the results: experience classification and subject observation. The experience classification was based on the number of experiences with inspection and usability. When subjects are observed, they might act differently as if they were not under observation, we tried to minimize this threat by encouraging them to see themselves as collaborators and freely criticize the evaluation object and the technique.

External validity: even though experiments using students as subjects and running on an academic environment are valuable as a way to characterize or understand aspects of new proposals, two issues must be taken into consideration: (1) probably students are not good substitutes for real world inspectors; and (2) usually, academic environments are far from simulating the existing conditions in industrial development environments. However, the Graduate student Portal represents a real application. Despite students not representing the entire population of software professionals, it has been shown that the differences between students and real developers may not be as large as assumed by previous research.

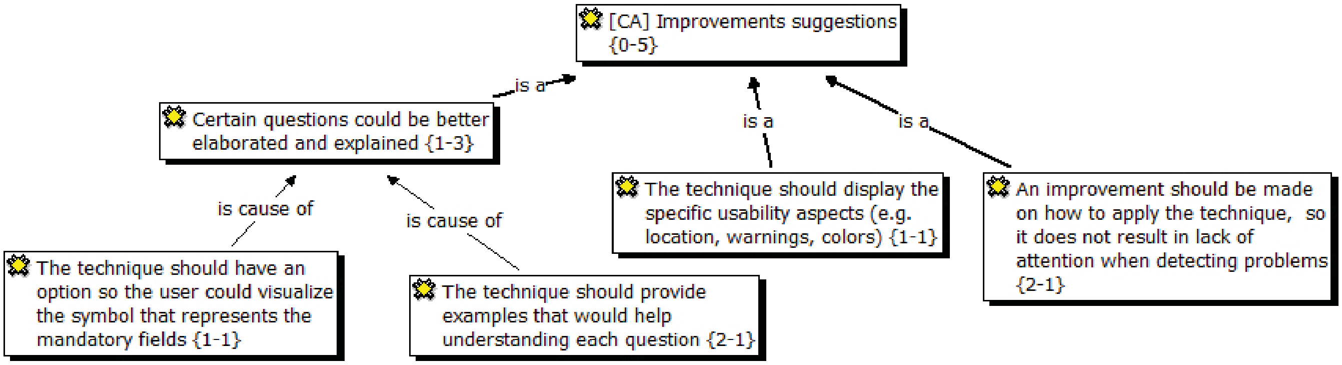

[CA] Improvements suggestions {0-5}
Certain questions could be better elaborated and explained {1-3}
The technique should display the specific usability aspects (e.g. location, warnings, colors) {1-1}
An improvement should be made on how to apply the technique, so it does not result in lack of attention when detecting problems {2-1}
The technique should have an option so the user could visualize the symbol that represents the mandatory fields {1-1}
The technique should provide examples that would help understanding each question {2-1}
is a
is a
is a
is cause of
is cause of

Figure 3. Improvements suggestions

Conclusion validity: the small number of data points is not ideal from the statistical point of view, however small sample sizes are a known problem difficult to overcome.

Construct validity: we measured efficiency and effectiveness using the same approach proposed in [27]. These two measures are often used in studies that investigate defect detection techniques [27], which is also our case.

VII. Conclusion And Further Work

This paper described an observational study aimed at eliciting how the usability inspectors apply the WE-QT technique, a usability inspection technique specifically tailored for novice inspectors. Both quantitative and qualitative results of this study provided us with important feedback to improve the WE-QT technique further.

The quantitative analysis showed that the effectiveness indicator computed for WE-QT v2 was higher than the effectiveness measured in the feasibility study. However, other factors may have influenced this outcome besides the improvement of WE-QT technique.

Despite the positive result of the quantitative data, the qualitative analysis showed that novice inspectors still have difficulties understanding determined questions of the technique. In addition, the way subjects applied the WE-QT technique did not correspond to the natural/correct inspection flow, indicating that the inspection process is not well defined in the technique. Therefore, according to the adopted methodology, it is necessary to improve our technique and execute another observational study.

Future work include: (1) improvement of the technique based on a detailed analysis of the influence of each verification in the final list of detected defects; (2) the replication of the experiment in an industrial environment, and, at last, (3) further studies comparing the WE-QT technique with other inspection techniques specific for evaluate Web applications usability.

Acknowledgments

The authors would like to acknowledge the support granted by CAPES process AEX 4982/12-6.

References

[1] G. Kappel, B. Proll, W. Retschitzegger, W. Schwinger, "Customization for ubiquitous web applications: a comparison of approaches", International Journal of Web Engineering and Technology, v. 1, pp. 79-111, 2003.

[2] E. Luna, J. Panach, J. Grigera, G. Rossi, O. Pastor, "Incorporating usability requirements in a test/model-driven web engineering approach", Journal of Web Engineering, v. 9, n. 2, pp. 132-156, 2010.

[3] J. Offutt, "Quality Attributes of Web Software Applications", IEEE Software, v. 19, n. 2, pp. 25-32, 2002.

[4] M. Matera, F. Rizzo, G. T. Carughi, "Web Usability: Principles and Evaluation Methods", E. Mendes, N. Mosley, (eds), Web Engineering, Spinger Verlag, 2006.

[5] E. Mendes, N. Mosley, S. Counsell, "The need for web engineering: an introduction", Mendes E., Mosley N. (eds.): 'Web engineering' Springer, pp. 1-26, 2006.

[6] M. Gomes, D. Viana, L. Chaves, A. Castro, V. Vaz, *et al*., "WDP-RT: A usability inspection reading technique for web applications". VI Experimental Software Engineering Latin American Workshop, v. 1, pp. 124 - 133, 2009. (In Portuguese).

[7] B. Bonifácio, D. Santos, C. Araújo, S. Vieira, T. Conte, " Applying Usability Inspection Techniques for Evaluating Web Mobile Applications". IX Brazilian Symposium of Human Factors on Computer Systems, v. 1., pp. 189-192, 2010. (In Portuguese).

[8] A. Fernandez, E. Insfran, S. Abrahão, "Usability evaluation methods for the web: A systematic mapping study", Journal Information and Software Technology, v. 53, n. 8, pp. 789-817, 2011.

[9] T. Conte, J. Massolar, E. Mendes, G. H. Travassos, "Web Usability Inspection Technique Based on Design Perspectives", IET Software Journal, v. 3, n. 2, pp. 106-123, 2009.

[10] P. Fernandes, L. Rivero, B. Bonifácio, D. Santos, T. Conte, " Evaluating a New Usability Inspection Approach: a Quantitative and Qualitative Analysis". VIII Experimental Software Engineering Latin American Workshop, v. 1, p. 67-76, 2011. (In Portuguese)

[11] F. Shull, J. Carver, G. H. Travassos, "An empirical methodology for introducing software processes", ACM SIGSOFT Software Engineering Notes, v. 26, n. 5, pp. 288-296, 2001.

[12] A. Strauss, J. Corbin, "Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory", 2ed., SAGE Publications, 1998.

[13] ISO (1997). ISO 9241-11: Ergonomic requirements for office work with visual display terminals (VDTs). Part 11 — Guidelines for specifying and measuring usability. Gènève: International Organisation for Standardisation.

[14] H. Rocha, M. Baranauskas, "Design and Evaluation of H uman-Computer Interfaces", M.C.C., 1. ed. Campinas: Emopi Publisher and Graphic, v. 1, p. 244, 2003. (in Portuguese).

[15] M. Costabile, M. Matera, "Guidelines for Hypermedia Usability Inspection", IEEE Computer Society Press, v. 8, n. 1, pp. 66-69, 2001.

[16] L. Triacca, A. Inversini, D. Bolchini, "Evaluating Web usability with MiLE+". VII IEEE International Symposium on, pp. 22-29, 2005.

[17] M. Blackmon, P. Pols on, M. Kitajima, "Cognitive walkthrough for the web". Conference on Human Factors in Computing Systems, v. 5, pp. 463 – 470, 2002.

[18] L. Filgueiras, S. Martins, C. Tambascia, R. Duarte, "Recoverability Walkthrough: An Alternative to Evaluate Digital Inclusion Interfaces". Latin American Web Congress, pp. 71-76, 2009.

[19] M. Allen, L. Currie, S. Bakken, V. Patel, J. Cimino, "Heuristic evaluation of paper-based Web pages: A simplified inspection usability methodology", Journal of Biomedical Informatics, v. 39, n. 4, pp. 412 – 423, 2006.

[20] A. Oztekin, A. Nikov, S. Zaim, "UWIS: An Assessment Methodology For Usability Of Web-based Information Systems", Elsevier Science Inc., v. 82, n. 12, pp. 2038-2050, 2009.

[21] M. Gomes, F. Santos, D. San tos, G. Travassos, T. Conte, "Evolving a Usability Evaluation Technique through In Vitro and In Vivo Studies". IX Brazilian Symposium on Software Quality, v. 1, pp. 229 – 244, 2010. (In Portuguese)

[22] V. Basili, H. Rombach, "The tame project: towards improvement-oriented software environments", IEEE Transactions on Software Engineering, v. 14, n. 6, pp. 758 –773, 1988.

[23] A. Dix, J. E. Finlay, G. D. Abowd, R. Beale, "Human-Computer Interaction", 3rd Edition, Prentice-Hall, Inc., 2003.

[24] A. Fernandez, S. Abrah, E. Insfran, "Towards to the validation of a usability evaluation method for model-driven web development", IV International Symposium on Empirical Software Engineering and Measurement, pp. 54, 2010.

[25] J. Nielsen, "Heuristic evaluation", In: Jakob Nielsen, Mack, R. L. (eds), Usability inspection methods, Heurisitic Evaluation, John Wiley & Sons, Inc, 1994.

[26] C. Wöhlin, P. Runeson, M. Höst, M.C. Ohlsson, B. Regnell, A. Wessl, "Experimentation in software engineering: an introduction", Kluwer Academic Publishers, 2000.

[27] C. Denger, R. Kolb, "Testing and Inspecting Reusable Product Line Components: First Empirical Results", V International Software Metrics Symposium, 2006.

Identification Guidelines for the Design of Interfaces in the context of ECAs and ADHD

Sandra Rodrigues Sarro Boarati
FACEN
Methodist University of Piracicaba
Piracicaba, SP
srsarro@unimep.br

Cecília Sosa Arias Peixoto
FACEN
Methodist University of Piracicaba
Piracicaba, SP
cspeixot @unimep.br

***Abstract*— The HCI (Human-Computer Interaction) aims to seek a friendly interaction between man and computer. To demonstrate it, it was studied in this paper the Cognitive Styles of Learning and users suffering from attention deficit disorder aiming at improvements in software development. The user population is diverse, there is a mixture of multiple profiles of users who need to somehow get their needs met and so this is why the system interface should be created in accordance with the diversity of users. Realizing this gap, this study examined the learning styles and attention deficits, allowing to generate a series of recommendations, guidelines, which are best suited to specific characteristics of the profiles of users. These recommendations could be applied in the construction of new interfaces that will be adjusted to different user profiles, and the use of these recommendations will contribute to greater user satisfaction respecting the different characteristics between them.**

Keywords - Human-Computer interaction; Cognitive Users Profiles; Guidelines for design interfaces.

I. INTRODUCTION

The HCI (Human-Computer Interaction) is a "discipline that concerns with the design, evaluation and implementation of interactive computing systems for human use and the study of major phenomena surrounding them. The goals of HCI are to produce usable systems, safe and functional "[1]. The interaction between man and machine takes place through the computer interface [2]. In the quest to bring the user interface, cognitive engineering, which is the process by which one acquires knowledge, is used in order to improve software development.

During the design phase of the interface are necessary to incorporate recommendations in order to build highly usable interfaces. Plasaint and Shneiderman [3] reported that a successful interface design should be based on recommendations from the project, called guidelines. For Nielsen [4], guidelines are list of principles to be followed in the development project. Nielsen proposes the use of a phase specifies guidelines (Guidelines and heuristic evaluation) life cycle, focusing on interface design, called Usability Engineering Lifecycle [4], given the importance of the recommendations during the design of an interface.

Authors such as Nielsen [6], Shneiderman and Plaisant [3] reported several design guidelines in their work and make several recommendations on how the interfaces should be for children, elderly, etc. However most of the recommendations touch only isolated aspects of the characteristics of users. Then there is a large gap in this area. There is more than one skill or characteristic to be understood and it stimulate the development of this study of the Cognitive Styles of Learning (CSL) from the perspective of Souto [7] and attention deficit, allowing to study a series of recommendations, guidelines for the design of interfaces and to consider more deeply the users profile or their skills.

To build the user profile is necessary first the diagnosis of the model user needs, as well as their cognitive, cultural and physical situations. This information is useful to provide the best interaction between the various information systems [8].

This research focused on adding guidelines and rules of selecting to the expert system GuideExpert from Cinto and Peixoto [5], just an area where researches are fairly recent and not yet cataloged in a joint manner. The GuideExpert system has the function to suggest and propose guidelines for interface design. This paper is organized into five sections: Section 1 presents the introduction. Section 2 presents the user and their role in the requirements specification; the interface; the diversity of users and the identification of learning styles. Section 3 presents the system GuideExpert, which is a tool that has the function to suggest and propose guidelines for interface design in its previous version and new recommendations suggested to improve the tool. Section 4 presents the analysis and implementation of some recommendations, guidelines of this work, sites for children and elderly users. Section 5 presents the conclusions.

II. THE USERS AND THEIR DIVERSITY OF PROFILES

The user population is not a system composed of only one type of user. In general, there is a mixture of multiple profiles of users who need to somehow get their needs met [2].

Speaking of users interacting with computers, we refer to the user's knowledge that should be taken into account in the design of a n HCI. Below are some features that must be observed during the interface design [6], [3]:

- The presence of an internationalized system or used in more than one country or region. Each country or region has its own peculiarities. Dialects, cultures, ethnicities, races, etc. All these elements end up generating needs that to be satisfied;

- These characteristics are considered common in a traditional interface, may not correspond to those made for children. They have unique needs for their age. Beyond these specific needs for the children users the designers need to deal with the dangers that are usually present in a web environment, such as pornography and violent or racist content;
- The existence of elderly among the users should be checked for these and needs met;
- People with special needs are another installment of the user community of the system and need adjustments in the system to operate in the environment without difficulty.

For the CSLs we used the basis of the research article: Project Tapejara from Souto [7]. The CSLs refer the subject's characteristic way of learning new concepts or even to generate elaborations of prior knowledge. According to Madeira, Wainer, Verdin, Alchieri, Diehl [11], the CSLs are: Analogue-Analytical (AA), Concrete-Generic (CG), Deductive-Evaluative (DA), Relational-Synthetic (RS) and Synthetic-Evaluative (SA).

Users rated AA-style may require more time for learning because when they were confronted with new information they tend to get a considerable depth on the subject through intense reflection. Users with CG-style tend to be pragmatic and careful in their learning situation. The learning objectives, evaluation criteria and feedback must be clear to this style because then he can work towards the goals. The DA-style can get to disregard a lot of concrete examples when they believe they have already understood the logical pattern underlying the new information. For the RS-style users are easy mental work with pictures and they appreciate the use of charts, diagrams and demonstrations. Efficient reading of charts and mind maps [7].

We relate the CSLs with users with attention deficit disorder and hyperactivity (ADD and ADHD). The people with ADD and ADHD do not develop the scholar knowledge as expected for their ages. The diagnostic in the scholar age is common because in this period can be found the difficulties of attention and remain silent as the studies of Siqueira and Gusgel Giannetti [9], Poeta and Rosa Neto [10].

III. The Project´s Guidelines

The GuideExpert system was initially developed containing in its knowledge base three hundred and twenty six classified guidelines and 10 meta-guidelines, which is the result of gr ouping guidelines, this allows you to search for more targeted features of i nterface design that is being modeled [5].

The system consists of four elements are: user interface, expert system (inference engine and working memory), knowledge base and database. Figure 1 shows the architecture of the expert system which demonstrates the elements through their organization in layers and modules.

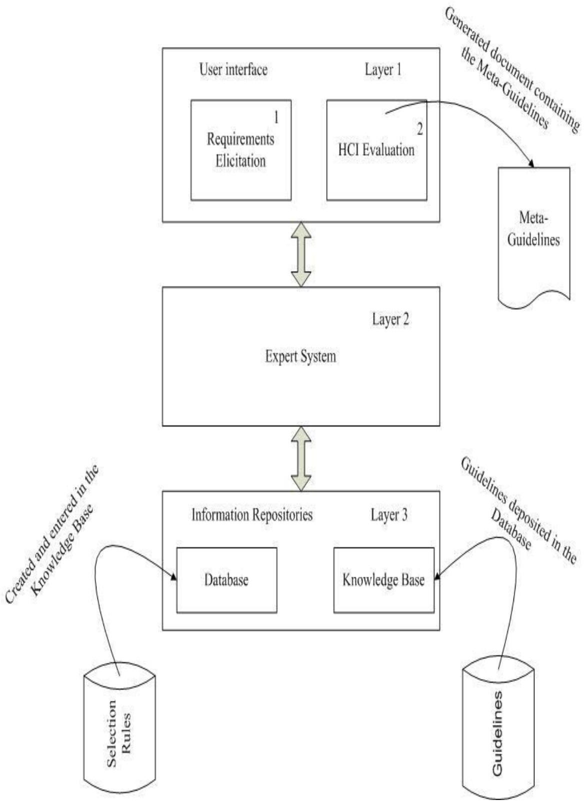


Figure 1 – System Architecture Specialist GuideExpert

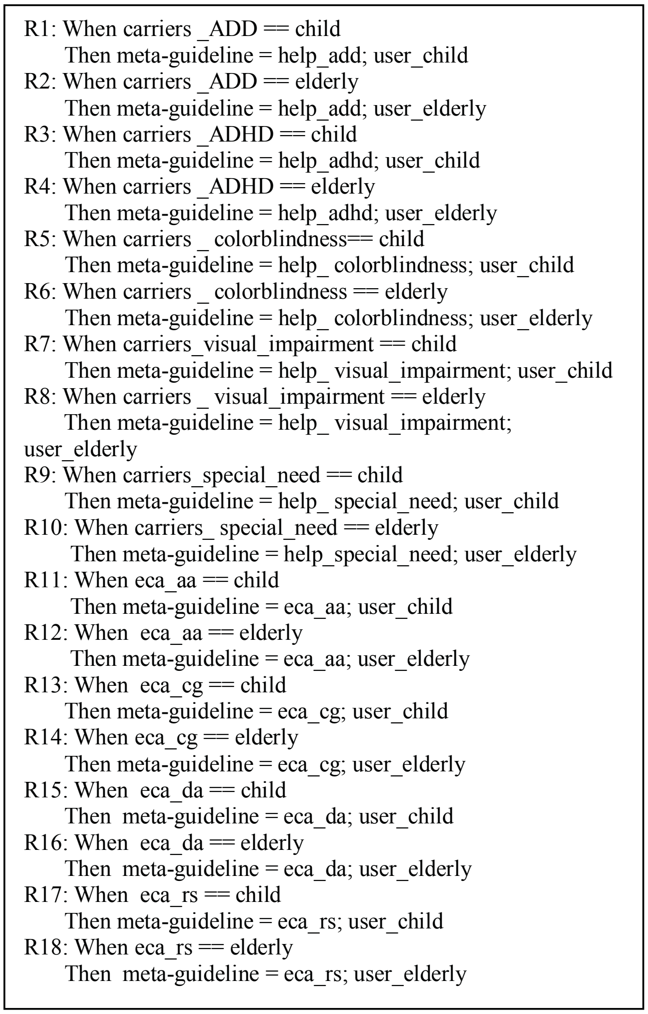
Layer 2 is the expert system, which, when executed, accesses the layer 3 containing the knowledge base, and thus carries the knowledge rules. The first module of the user interface has been assessing the requirements of the HCI with the designer, if executed. After obtaining the information, is provided to the expert system and soon after is analyzed. The database is accessed at layer 3, for the meta-guidelines requested. The second module of user interface aids in heuristic evaluation [5].

This research contributed to the tool GuideExpert incorporating new knowledge items to it, enabling the groups of users with different learning styles (CSLs) and users suffering from attention deficit disorder (ADD and ADHD) obtain special guidelines.

The metaguidelines were increased by more than 18 categories and 18 new rules are created according to the surveys [9], [10], [11] and [12]. The selection rules help the designer to automatically select the most appropriate metaguidelines. For this project we raised through the literature one hundred thirty-six new guidelines.

A. *Representation of Guidelines*

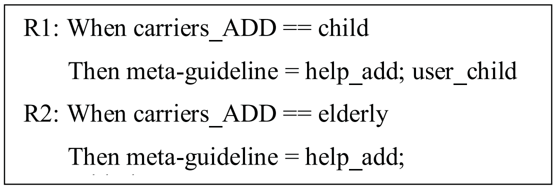
The knowledge basis of G uideExpert consists in the "WHEN-THEN" rules, who help to select the appropriate metaguidelines for the interface being designed. This study adds to the base already built the 18 selected rules that were created according to the preview research and follow the same syntax, is shown in the Figure 2.



```
R1: When carriers _ADD == child
      Then meta-guideline = help_add; user_child
R2: When carriers _ADD == elderly
      Then meta-guideline = help_add; user_elderly
R3: When carriers _ADHD == child
      Then meta-guideline = help_adhd; user_child
R4: When carriers _ADHD == elderly
      Then meta-guideline = help_adhd; user_elderly
R5: When carriers _ colorblindness== child
      Then meta-guideline = help_ colorblindness; user_child
R6: When carriers _ colorblindness == elderly
      Then meta-guideline = help_ colorblindness; user_elderly
R7: When carriers_visual_impairment == child
      Then meta-guideline = help_ visual_impairment; user_child
R8: When carriers _ visual_impairment == elderly
      Then meta-guideline = help_ visual_impairment;
user_elderly
R9: When carriers_special_need == child
      Then meta-guideline = help_ special_need; user_child
R10: When carriers_ special_need == elderly
       Then meta-guideline = help_special_need; user_elderly
R11: When eca_aa == child
       Then meta-guideline = eca_aa; user_child
R12: When  eca_aa == elderly
       Then meta-guideline = eca_aa; user_elderly
R13: When  eca_cg == child
      Then meta-guideline = eca_cg; user_child
R14: When eca_cg == elderly
      Then meta-guideline = eca_cg; user_elderly
R15: When  eca_da == child
      Then  meta-guideline = eca_da; user_child
R16: When  eca_da == elderly
      Then  meta-guideline = eca_da; user_elderly
R17: When  eca_rs == child
      Then meta-guideline = eca_rs; user_child
R18: When eca_rs == elderly
      Then  meta-guideline = eca_rs; user_elderly
```

Figure 2 – Rule Selection

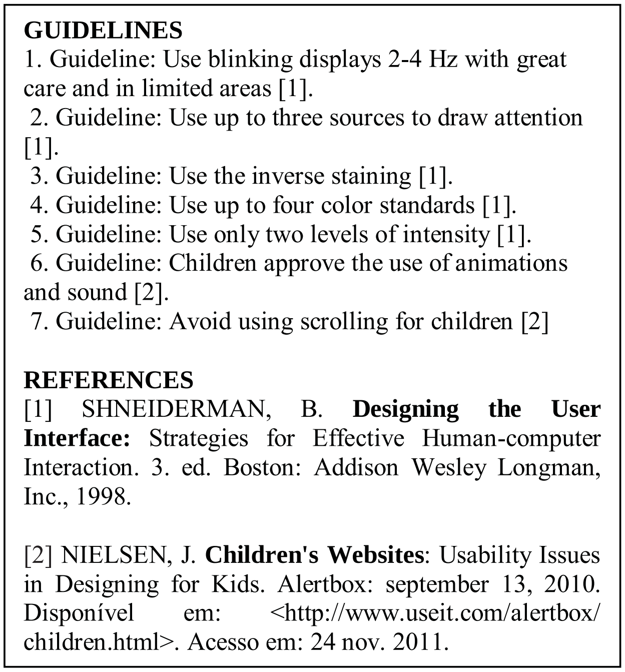
As example is shown in Figure 3, the rule knowledge base for people with ADD related to children and elderly people. It was increased to the knowledge base tool.



```
R1: When carriers_ADD == child
      Then meta-guideline = help_add; user_child
R2: When carriers_ADD == elderly
      Then meta-guideline = help_add;
```

Figure 3 – Rule Knowledge Base - People with ADD - Children – Elderly

For the construction of the selection rules we cross information of users of ECAs with ADD and ADHD disorders and other characteristics, we used the parameter age (child and adult). The resulting guidelines for the rule R1, for example, is shown in the Figure 4, which was selected set of guidelines for users with ADD and set of guidelines for user-child.



GUIDELINES

1. Guideline: Use blinking displays 2-4 Hz with great care and in limited areas [1].
2. Guideline: Use up to three sources to draw attention [1].
3. Guideline: Use the inverse staining [1].
4. Guideline: Use up to four color standards [1].
5. Guideline: Use only two levels of intensity [1].
6. Guideline: Children approve the use of animations and sound [2].
7. Guideline: Avoid using scrolling for children [2]

REFERENCES

[1] SHNEIDERMAN, B. **Designing the User Interface:** Strategies for Effective Human-computer Interaction. 3. ed. Boston: Addison Wesley Longman, Inc., 1998.

[2] NIELSEN, J. **Children's Websites**: Usability Issues in Designing for Kids. Alertbox: september 13, 2010. Disponível em: <http://www.useit.com/alertbox/children.html>. Acesso em: 24 nov. 2011.

Figure 4 – Guidelines - People with ADD - Children

B. *New Interfaces for User´s Profile Selection*

We recommend changes to the GuideExpert interfaces because of the addition of new metaguidelines. The changes were suggested in the items: task analysis (because it not allow user to choose the user "child" and also the to choose the needs of the users); context analysis (new items of graphical user interface were added); evaluation of interface design (new items of choices were added for the visual deficient, special needs, ADD, ADHD and others).

IV. Analysis and Application Guidelines

In this section we analyze two websites, one designed for child-user "Kids Channel" and one for adult-user "Government Portal of the State of São Paulo". The aim was to determine if the websites could be used for the project of interaction. After this analysis we propose improvements to the interfaces using the GuideExpert that recommends appropriated guidelines for their users. The analysis evaluates the interface in accordance with the guidelines identified from the literature. Based on this evaluation, you choose the desired criteria in the tool. Indicates the criteria of a design tool to formulate, develop appropriate interfaces for user profiles.

A. *Analysis of the Website Canal Kids for Children*

Looking at the children´s website “Kids Channel” [1], we can saw it has several children learning and fun items, as shown in Figure 5.

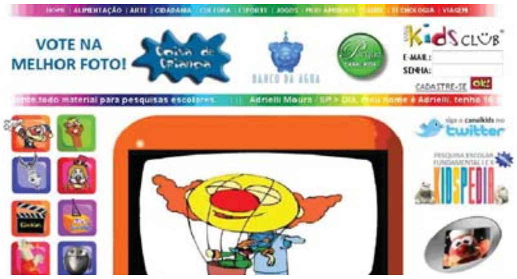


Figure 5 – Site Kids Channel / Home

The initial homepage “Home” use many pictures and colors to draw the attention of the child user. The website makes use of colors, sounds and color changing on the top banner and it draws the attention of children with ADD, as recommended below:

- Children approve the use of animations and sound [13];
- Use different types of sounds and visual effects [14].

After examination of the website home page, we found it is not developed appropriately for the target audience. Thus, if the user is a child and has the attention deficit disorder (ADD) the selection rule R1 will be started by the GuideExpert system resulting sets of guidelines for the appropriate user – child-user, ADD-user.

In the games item, as shown in the Figure 6, the content is very extensive and it should be avoided. Children find it difficult to read large blocks of text, especially when the text was written above their reading level. And the kids do not have the option on that page to read the entire contents using the arrow keys to move down.

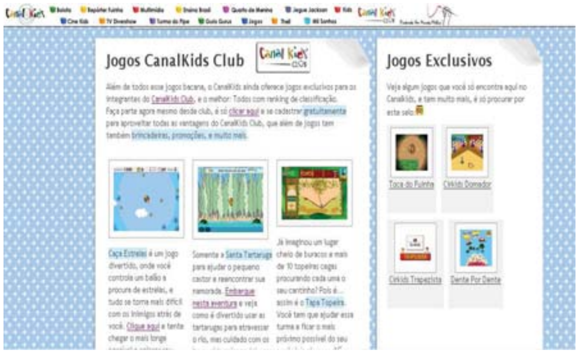


Figura 6 – Site Kids Channel / Games

The Games page at the Kids Channel was not developed properly. Thus, if the user is a child with the cognitive profile learning Deductive-Evaluative (DA), the selection rule R15 will be selected by the GuideExpert system resulting sets of guidelines suitable for child-user with AD profile as recommended: presentation of the concept using figures schema [7].

Analyzing the Kids Channel website, we selected seventy-three guidelines.

B. *Analysis of the Website Government Portal of the State of São Paulo*

During the browsing on the Kids Channel website, we can choose the option “Brazil Teaches”, where appears a link to the “Government Portal of the State of São Paulo”. On this site we analyzed the use of guidelines for the aged user, and if they are in agreement with the recommendations of the HCI. When we select the image we are directed to the “Government Portal of the State of São Paulo” [2] as shown in Figure 7.

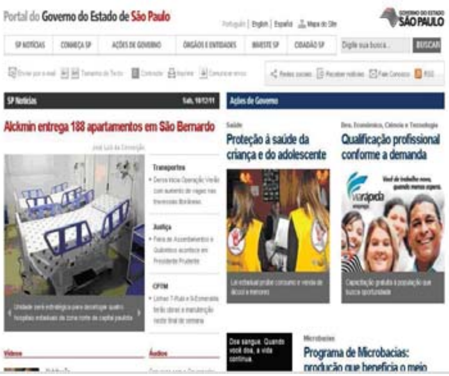

Portal do Governo do Estado de São Paulo
SÃO PAULO
SP Notícias
Ações de Governo
Alckmin entrega 188 apartamentos em São Bernardo
Proteção à saúde da criança e do adolescente
Qualificação profissional conforme a demanda
Programa de Microbacias:

Figure 7 - Site Government Portal of the State of São Paulo

This page of the website is consistent with suggested recommendations for aged-user. In the top menu there are options to increase the font size, change the contrast and not use commercial banners. According to the recommendations: watch the size of fonts, screen contrast and audio volume [2]; and the font size used in the texts should be at least 12 points using buttons to increase and decrease the types, for example, AA-A+, not to use banner advertising or auxiliary window or animations. If these resources were used they should be in full screen and the user must be able to disable them easily. Don’t use icons, buttons, links and small characters.

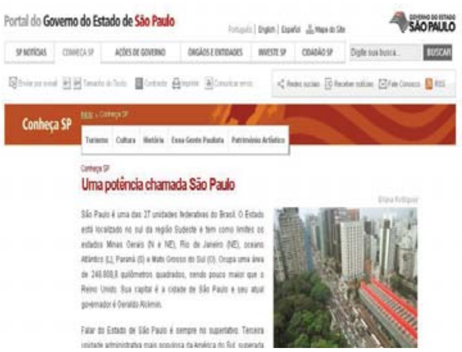
Despite the good website presentation, the links at the top menu are fixed and have too small fonts for elderly users. After all many aged persons have some type of visual impairment.

[1] Site available in Kids Channel: http://www.canalkids.com.br/portal/index.php.

[2] Site available in Government Portal of the State of São Paulo: http://www.saopaulo.sp.gov.br.

In our analysis, the "Government Portal of the State of São Paulo" needs to properly develop some tools for aged-user and the visually impaired. Thus, if the user is old and visually impaired, the GuideExpert system select the R8 rule resulting in sets of guidelines for these kind of users.

At tab "Meet São Paulo", as shown in the Figure 8, we find an extensive report. Again many users can not read the text because they lost interest. According to the cognitive style AA (Analogue-Analytical), the use de concepts and examples with text comparative schemes, schemes with comparative figures, i.e., combining text and image facilitates the analytical process and analogous relations [11].



Uma potência chamada São Paulo

São Paulo é uma das 27 unidades federativas do Brasil. O Estado está localizado no sul da região Sudeste e tem como limites os estados Minas Gerais (N e NE), Rio de Janeiro (NE), oceano Atlântico (L), Paraná (S) e Mato Grosso do Sul (O). Ocupa uma área de 248.808,8 quilômetros quadrados, sendo pouco maior que o Reino Unido. Sua capital é a cidade de São Paulo e seu atual governador é Geraldo Alckmin.

Falar do Estado de São Paulo é sempre no superlativo. Terceira unidade administrativa mais populosa da América do Sul, superada

Figure 8 – Site Government Portal of the State of São Paulo – Meet São Paulo

Looking at the page tab "Meet São Paulo", we found that were not properly developed for aged-user. Thus if the user is old with cognitive style Analogue-Analytical (AA), GuideExpert system select the R12 rule resulting in sets of guidelines for aged-user and the Analog-Analytic cognitive style (AA).

In the analysis of the "Government Portal of the State of São Paulo" were selected a total of sixty-seven guidelines.

By the extension of the GuideExpert it will be possible to specialize more and more recommendations; it will help the designer to automate a way of selection of guidelines that will guide the design or evaluation of interfaces.

V. Conclusion

We note that throughout this work, searching for references related to the proposed theme, we found renowned authors as Nielsen and Shneiderman, who make several recommendations for design of interfaces for children, old people, etc.

However most of the recommendations are about isolated or general aspects of the characteristics or skills of the users. We noticed a large gap in this area in order to relate more than one feature. Considering these problems we examined the learning styles and attention deficits, which allowed us to unveil a series of recommendations and guidelines who suit the specific characteristics of the users profiles.

It was possible to create sets of selection rules according to the different user profiles and their learning styles, resulting in several sets of recommendations and guidelines according to each profile. These recommendations were incorporated into the GuideExpert tool and subsequently applied in the analysis of sites for children and seniors.

References

[1] H. V. da Rocha, M. C. C. Baranauskas. **Design e Avaliação de interfaces Humano-Computador**. Campinas: Unicamp, 2003, pp. 14, 17.

[2] A. A. de Oliveira Netto. **IHC**: Modelagem e Gerência de Interfaces com o Usuário. Florianópolis: Visual Books, 2004.

[3] B. Schneiderman, C. Plaisant. **Designing the user interface**. Strategies for effective human-computer interaction. 5th Ed. Boston: Addison-Wesley Longman, Inc, 2010.

[4] J. Nielsen. **The Usability Engineering: life Cycle. Computer**, vol.25, march. 1992, pp. 12-22.

[5] T. Cinto, C. S. A. Peixoto. **Guidelines de Projeto de Interfaces Homem-Computador: Estudo, Proposta de Seleção e Aplicação em Desenvolvimentos Ágeis de Software**. Relatório Científico PIBIC/FAPIC, UNIMEP, Piracicaba, Brazil, 2010.

[6] J. Nielsen. **Usability Engineering**. Boston: Academic Press, 1993.

[7] M. A. M. Souto. **Diagnóstico on-line do estilo cognitivo de aprendizagem do aluno em um ambiente adaptativo de ensino e aprendizagem na web: uma abordagem empírica baseada na sua trajetória de aprendizagem**. 2003. 147f. Tese (Doutorado em Ciência da Computação) – Universidade Federal do Rio Grande do Sul, Porto Alegre, 2003.

[8] L. F. da Costa. **Usabilidade do portal de periódicos da capes**. 2008. 236f. Dissertação (Mestrado em Ciência da Informação) - Universidade Federal da Paraíba, João Pessoa, 2008.

[9] C. M. Siqueira, J. Gurgel-Giannetti. Mau desempenho escolar: uma visão atual. **Revista da Associação Medica Brasileira**. 2011, vol.57, n.1, pp. 78-87. ISSN 0104-4230.

[10] L. S. Poeta, F. Rosa Neto. Prevalência de escolares com indivíduos de transtorno de déficit de atenção/hiperatividade (TDA/H). **Temas sobre desenvolvimento**, 2005-6; 14(83-84) dez-jan/2005 e jan-fev/2006, pp. 57-62.

[11] F. Bica, M. A. M. Souto, R. M. Vicari, J. P. M de Oliveira, R. Zanella, G. Vier, K. B. Souza, A. A. Sonntag, R. Verdin, M. J. P. Madeira, S. B. Charczuk, M. Barbosa. Metodologia de construção do material instrucional em um ambiente de ensino inteligente na web. **XII Simpósio Brasileiro de Informática na Educação SBIE – UFES**, Vitória, ES, Brazil, 21-23 November, 2001, vol. 1, pp. 374-383.

[12] M. J. P. Madeira, R. Wainer, R. Verdin, J. C. Alchieri, E. K. Diehl. Geração de estilos cognitivos de aprendizagem de negociadores empresariais para adaptação de ensino tutorializado na web. **Paidéia**, 2002, vol. 12, issue 23, pp. 133-147.

[13] J. Nielsen. **Children's Websites: Usability Issues in Designing for Kids.** Alertbox: september 13, 2010. <http://www.useit.com /alertbox/children.html>.

[14] A. P. da Silva, H. Martucci Neto, T. A. Scardovelli, H. A. D. de Oliveira, A. F. Frerè. **Auxilio ao letramento de crianças com hiperatividade via internet**. International Association for Development of the Information Society, Conference IADIS, Ibero-Americana WWW INTERNET, Madrid, Spain, October, 2004, pp 258-260.

Measuring the effect of usability mechanisms on user efficiency, effectiveness and satisfaction

Marianella Aveledo M., Diego M. Curtino, Agustín De la Rosa H., Ana M. Moreno S.
Facultad de Informática
Universidad Politécnica de Madrid
Madrid, Spain

Abstract—**In this paper we present the results of two experiments designed to evaluate the impact of particular usability mechanisms on software system usability attributes. It focuses on mechanisms that human computer interaction (HCI) and software engineering (SE) researchers regard as having a major impact on software development, like undo/ cancel or provide feedback. The experiments were run using two different software applications that we developed specially for the purpose. From the results, we conclude that the inclusion of different usability mechanisms has a positive impact on the usability attributes of efficiency, effectiveness and satisfaction.**

Keywords: usability, usability testig, usability attributes

I. INTRODUCTION

Usability is a software quality attribute listed in most classifications [1][2]. According to ISO/EIC[3], usability is an attribute of software system quality of use and is defined as "The extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use". Usability means how well a product meets stakeholders' needs and achieves specified effectiveness, efficiency and satisfaction goals in a specified context of use. It is not surprising then that usability is increasingly recognized as one of the critical factors for software system acceptance [4].

It has been demonstrated that usability has implications that go beyond the user interface and affect the system as a whole [5][6]. In this context, Juristo, Moreno and Sánchez-Segura [7] identify particular usability mechanism with a major impact on the software functionality. They are: progress feedback, system status feedback, warning, global undo, abort, cancel, structure text entry and help.

Our aim is to study the effect of incorporating such mechanisms in particular usability attributes like efficiency, effectiveness and satisfaction. For that aim we have conducted a literature review of the field of usability evaluation and have found usability testing approaches proposed by several authors [8][9][10][11] [12]. We have studied the different approaches and proposals and found that they all focus on the use of the continuous evaluation of the designed interfaces in usability testing throughout the iterative development of a software system. The goal appears to be to reach the implementation stage with a fairly mature interface model specifying the different usability features established through the iterative evaluation process. The different proposals evaluate a set of ergonomic software criteria and/or usability features that a well-designed user interface should have, such as, visual clarity, consistency, compatibility and appropriate functionality. Note that post-delivery evaluation proposals focus on improving upcoming product releases. [9].

Several tools designed and developed for usability testing have been reported in the literature since the 1990s [13][14][15][16][17][18][19]. Notably, none introduce methodological and/or tool proposals designed to evaluate the software system interface from the viewpoint of how mechanisms for improving different usability attributes can benefit the system, considering effectiveness, efficiency and satisfaction as usability attributes [2][3].

Recently we have designed two experimental applications called EasyTheatre and EasyFlight. These applications have been developed to capture data that facilitate the measurement of the above impact [20]. These applications are used to take quantitative and qualitative measurements that provide sufficient data for calculating the impact of the mechanisms on the attributes. The quantitative measures taken are time to complete the task and number of clicks or equivalent navigational actions. The time to complete tasks is useful for measuring the efficiency attribute, whereas the number of clicks and/or equivalent actions is useful for measuring the impact on relative efficiency [21][22]. The qualitative data that we gather provide information on subjective user issues that are used to calculate the impact on the satisfaction attribute [21][23][24] through the mechanism/question/attribute relationship [20]. So, the test performed can be defined as summative [25], that is, a test whose goal is to perform a competitive analysis to learn how much better (if any) the application is perceived to be by users when the usability mechanisms are implemented. Finally, the impact on the effectiveness attribute is measured through information supplied by users on whether or not they completed the task successfully [2].

In the following, we present the results of the experiments run using the tools that we developed.

II. EXPERIMENTS

A.Pilot test using the EasyTheatre tool

First we ran a pilot test using the EasyTheatre application [26]. EasyTheatre is an e-commerce application for booking theatre tickets. One of the key characteristics of the application is that it includes a usability dashboard. Users can use the dashboard to enable and disable functions corresponding to specific usability mechanisms.

Apart from correctly capturing data, the goal of this pilot study was to establish the soundness of the application, as well as the type of data to be collected and the reliability of the experimentation process.

We ran two experiments in which we tried to measure the impact of global undo, help, system status feedback and warning mechanisms on the efficiency and satisfaction attributes.

The experiments were run with 2011 Universidad Politécnica de Madrid Master in Software Engineering students and 2010 Universidad Simón Bolívar Master in Systems Engineering students. A total of 24 subjects participated in the experiment. They were divided into two groups: a control group (CG) that did not use usability mechanisms and a test group (TG) that used built-in usability mechanisms. Users were given use scenarios and administered questionnaires used to collect the data in writing. Users had to use the application to execute the respective scenarios and then answer the survey questions about the execution. The only difference between the scenarios given to students belonging to the control and test groups was that the control group members were not asked to enable the mechanisms to be measured (global undo, help, system status feedback and warning mechanisms), whereas the test group members were. The questionnaires were the same for both groups. Divided into quantitative data: task time in minutes reported by users in writing and qualitative data: responses on a 1-to-5 Likert scale for each mechanism graded from Never to Always [10][24][27].

Note that we use the difference (increase) in the data for the test group compared with the control group in order to measure the impact.

Data analysis

The datasets were analyzed depending on the type of data they contained. The quantitative data were analyzed using the Microsoft Excel· and SPSS 16 (Statistical Package for the Social Sciences) [28] software tools, whereas the qualitative data were analyzed using SPSS 16 only. SPSS was used to perform the non-parametric Mann-Whitney U test [29] to check whether the differences between the variables are statistically significant, at 95% confidence level.

The aim of the experiment was to determine whether the mechanisms have an impact on the already mentioned usability attributes. The Mann-Whitney U test determines whether there is a statistically significant difference between the data reported by the CG and the TG. Not until this statistical significance has been determined can we infer the effect of each mechanism on the usability attributes.

Quantitative data analysis

Table I lists the mean times used by the control and the test group for each mechanism.

The results of the Mann-Whitney U test applied to the gathered quantitative data show that the difference between the mean times to complete a task in the test and control groups was not statistically significant in any case. The reasons for this could be any, all or a combination of the following:

- Homogeneous group: all users were studying for a computing-related master's degree and had a computing-related background.
- There were fewer than 20 subjects per group.
- Users reported time manually in minutes.

Although the Mann-Whitney U test results are inconclusive, note that, in all cases except for the global undo mechanism, there was a positive difference between the means of the time indicator in minutes in the test and control groups.

There is an explanation for result with respect to the global undo mechanism, namely, users cannot undo actions if the mechanism is not enabled. Control group users cannot find a button to undo the action, so they immediately give up. Accordingly, when we analyzed user responses to the control group survey associated with this mechanism, we found that all the users of this group responded that it is never possible to undo actions.

TABLE I. MEAN TEST DURATION (MINUTES)

MECHANISM	CG	TG	Difference	Increase (%)
Global Undo	3.36	3.88	-0.52	-15,4
Help	4.0	2.91	1.09	27,25
Warning	2.41	1.7	0.71	29,46
System Status Feedback	4.3	2.3	2.0	46,51

Qualitative data analysis

The Mann-Whitney U test applied over the questionnaire responses, revealed that the differences in the satisfaction attribute of both groups were statistically significant at 95% confidence level.

The results show that the differences between the mean responses to all the questions concerning the warning mechanism are statistically significant. This applies to the system status feedback indicators as well. But, the data for the other mechanisms were not always significant. The help mechanism indicators show that three out of the five survey questions were statistically significant, whereas only two out of the four questions for global undo were statistically significant.

We analyzed the survey questions for these mechanisms and concluded that this was a foreseeable result because some of the questions were worded in such a manner as not to signal the functionality that the mechanisms actually provide.

Removing the incorrectly worded questions, we can calculate the mean increase in user satisfaction [24] with respect to the inclusion of each mechanism. *Table II* shows the results. These results show that system status feedback is the mechanism that has the great impact on the satisfaction attribute

B. Experimental test using EasyFlight tool

As a result of the lessons learned in the pilot study, we took corrective measures in order to gather more, and more accurate and reliable data. So, we designed another scenario-driven software application with automated quantitative data capture to prevent any errors caused by human task performance timing. We also implemented the automatic capture of number of clicks or equivalent navigational actions functionality. We revised and corrected the survey questions associated with the different mechanisms and automated survey-taking. Users of this new application do not choose which mechanism to enable, that is, the application automatically enables the mechanism when they select which scenario to execute.

TABLEII. MEAN INCREASE IN SATISFACTION

MECHANISM	Increase (%)
Warning*	161.43
Global Undo*	213.99
Help*	223.01
System Status Feedback*	256.58

* Statistically significant

Thanks to the features added to the new application, the experiment can now be run without the physical presence of the participant users. This means that we can test many more subjects from a heterogeneous group.

The new application developed was the EasyFlight [30]. It had the following functionalities: perform airport administration (i.e., add/edit airports), list and optionally purge the list of expired bookings, check available flights for an airline route or book a flight.

Recruiting process

The people that participated in the experiment were recruited by email. Around 75 requests were sent, and 46 acceptances were received. Of the 46 participants, 24 were categorized as general users (GU) and 22 as computing-related users (CRU). This categorization is necessary because computer literate users might perform better than general users, as they are likely to have frequent exposure to controls and procedures that software applications usually provide such as calendars, dropdown lists, warnings and abort mechanisms and/or go back controls. The categorized users were assigned to either the control or test group. Each one of these two groups was composed of 23 people: 12 GU and 11 CRU. We assumed that response time was random and considered it unnecessary to set up any other randomization process to select the elements of each group. To populate the control group, we went down the list and picked the first 12 GU and the first 11 CRU. We populated the test group similarly.

Task assignments

All 46 participants were given the same tasks related to Global Undo, Progress Feedback, Structured Text Entry and Go Back mechanisms. The tasks are “Undo airport addition”, “Get list of expired bookings”, “Get flight information” and “Book flight tickets”.

All participants were set the same set of four tasks. Details on how to login into the system were provided along with these tasks. *Figure 1*shows an example of some of the steps of the scenario for the task “Book Flight Tickets” and *Figure 2* illustrates the corresponding questionnaire.

Data analysis

As with EasyTheatre, the datasets were analyzed depending on the type of data they contained. The detailed results of these analysis can be found in [31].

USE SCENARIO

The application VuelosFácil is an e-commerce web application developed for the acquisition of low cost airlines flight tickets. Task description: Assume that you are a person visiting the airline's website because **you are interested into book flight ticket as long as they fit on your vacations plans.**

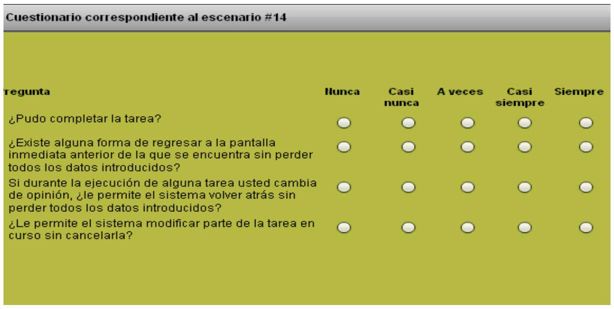
In this particular case, from the available flights you are interested into booking **2 tickets from Barcelona to Granada on the date November, 11th 2011 at 10:30 AM**. However, at choosing the flight you make a mistake and select the one at 6:00 a.m. Thus, you want to modify the flying time before submitting the booking.

You shall:

1. Once logged in, select scenario **"Escenario #14"** from the scenarios' list that is available at the upper left corner of the screen.
2. Click on "Iniciar".
3. Click on "Usuarios".
4. Choose the flight at 6:00 horas from Barcelona to Granada on November, 11th 2011. To do so:
    a. Input "BCN-00" as source airport.
    b. Input "GRX-92" as destination airport.
    c. Input "2011-11-11" as flight date
    d. Input 2 as the number of passengers.
5. Click on "Mostrar vuelos" button.

…….

Figure 1. Scenario for the task "Book flight tickets"



Cuestionario correspondiente al escenario #14

regunta	Nunca	Casi nunca	A veces	Casi siempre	Siempre
¿Pudo completar la tarea?	○	○	○	○	○
¿Existe alguna forma de regresar a la pantalla inmediata anterior de la que se encuentra sin perder todos los datos introducidos?	○	○	○	○	○
Si durante la ejecución de alguna tarea usted cambia de opinión, ¿le permite el sistema volver atrás sin perder todos los datos introducidos?	○	○	○	○	○
¿Le permite el sistema modificar parte de la tarea en curso sin cancelarla?	○	○	○	○	○

Figure 2. Questionnaire for task "Book flight tickets"

Collected data

Both quantitative and qualitative data were collected on each participant. Quantitative data are composed of the time taken to perform the task, and the number of click for performing the task. These data were collected automatically without participant intervention. Qualitative data are participant responses to each task survey. There was no means of collecting these data without participant involvement. The data were collected by the developed web application and stored in a database.

Quantitative data analysis

We calculated the following indicators for each quantitative dataset:1) the mean number of click equivalent actions and 2) The mean time to complete the task in milliseconds. The Mann-Whitney U test [29] was used to check whether the differences between the mean variables for the control, and test group are statistically significant.

The corresponding usability attributes were measured as following: 1) Efficiency: difference between the means time in milliseconds to complete the task calculated as a CG/TG ratio value expressed as a percentage and 2) Relative Efficiency: Difference between the number of clicks per group with respect to the minimum number of clicks required to complete the task.

Table *III* below shows the results for the increase in the efficiency attribute due to each mechanism. Notice that for all mechanisms but go back, the efficiency in the control and test group is statistically significant.

A similar procedure was used to calculate relative efficiency, where 100% task efficiency was equivalent to the minimum number of clicks required to complete the task. Table *IV* lists the relative efficiencies of each group. As Table *IV* shows, we cannot assess the impact of the go back mechanism, as it is not statistically significant.

We find, on the other hand, that the progress feedback and structured text entry mechanisms have a positive impact on the efficiency attribute, where structured test entry is the mechanism with the biggest impact. Regarding the relative efficiency attribute for the global undo mechanism, we can only calculate the relative efficiency of the test group, as it is impossible to undo operations if the mechanism is disabled, and once again the structure text entry mechanism is the one with the biggest impact.

Qualitative data analysis

We also calculate the mean values for each question in the questionnaire and group. We can calculate the mean response to each question, as the survey questions were rated on a 1-to-5 Likert scale [27]. We used these values to calculate the means.

Question #1, *Were you able to complete the task?,* was the same in all surveys. This question was designed to measure the impact of each mechanism on the effectiveness attribute. It is justified by the fact that, according to [20], effectiveness can be measured according to user perceptions, and one such measure is "percentage of users successfully completing a task".

Table V shows the results for Question #1 of the survey for each evaluated mechanism. The impact of the progress feedback mechanism is inconclusive, as it is not statistically significant. We find that the global undo, structured text entry and go back mechanisms have a positive impact on effectiveness, where structured text entry has the biggest impact.

The Mann-Whitney U test of the qualitative data (survey responses) revealed that the mean difference in the responses to all the questions, for all mechanism, were statistically significant. *Table VI* shows the mean increases for all the mechanisms.

We infer from these data that most mechanisms more than double, and progress feedback almost triples, user satisfaction. This means that users are on average twice as satisfied with the applications that provide usability mechanisms.

TABLE III.RESULTS FOR THE IMPACT OF EACH MECHANISM ON THE EFFICIENCY ATTRIBUTE.

Mechanism	Increase (%)
Global Undo *	46.98
Progress Feedback *	34.15
Structure Text Entry *	62.7
Go Back	**14**

*Statistically significant.

TABLE IV.RESULTS FOR THE IMPACT OF EACH MECHANISM ON THE RELATIVE EFFICIENCY ATTRIBUTE.

Mechanism	TG relative efficiency	CG relative efficiency	Increase (%)
Global Undo	87.3%	#	
Progres Feed.*	100%	40%	60
Structured TE *	86%	25%	61
Go Back	72%	68%	**4**

* Statistically significant
# This system implementation is unable to undo an operation unless the mechanism has been enabled

TABLEV. RESULTS FOR IMPACT ON EFFECTIVENESS ATTRIBUTE

Mechanism	CG users that complete the task	TG users that complete the task	Increase (%)
Globla Undo *	60.9 %	95.7 %	35.7
Progress Feedback	95.7 %	100 %	4.3
Structure Text Entry*	86.96 %	4.4 %	82.56
Go Back *	70 %	18 %	52

*Statistically significant.

TABLE VI. RESULTS FOR THE IMPACT OF EACH MECHANISM ON THE SATISFACTION ATTRIBUTE

MECHANISM	MEAN INCREASE (%)
Global Undo*	82.28
Progress Feedback*	294.08
Structured Text Entry*	251.50
Go Back*	205.75

*Statistically significant.

III. CONCLUSIONS

The conclusions of this paper refer to the results of the experiment on the EasyFlight application, as they are much more reliable than the results for the EasyTheatre application. The findings from the results analysis were:

- The structured text entry mechanism was statistically significant in all cases. It also had a high impact, over 60%, on all usability attributes.
- The progress feedback mechanism did not have a statistically significant impact on the effectiveness mechanism. The reason is that users always manage to complete the tasks, irrespective how much information they have on the progress of the task.
- Note that control group users cannot complete the task if global undo is disabled. On this ground, we cannot calculate that relative efficiency of this mechanism.
- The go back mechanism was not statistically significant for calculating the efficiency and relative efficiency attributes. A possible reason is that the browser go back functionality was enabled. This will be corrected in future experiments.

We intend to continue testing the EasyFlight tool for other mechanisms: system status feedback, abort, cancel, warning and help.

Note that the two software applications used in this experiment have similar features. Both are e-commerce transaction systems. Obviously, we have to check the results of this research on other software application types in order to generalize the results. To do this, we have now developed an event-driven home automation application which we will use to run a similar experiment in the very near future.

REFERENCES

[1] "ISO/IEC Std. 9126-1: Software engineering – Product quality," 2001.

[2] "ISO/IEC Std. 9241-11: Ergonomic requirements for office work with visual display terminals. Part11: "Guidance on Usability"," 1998.

[3] "ISO/IEC Std. 25010-3: Systems and software engineering: Software product quality and system quality in use models," 2009.

[4] Cysneros, L.; Wemeck, V.; Kushniruk, A., "Reusable Knowledge for Satisficing Usability Requirements". Proceedings of the 2005 13th IEEE International Conference on Requirements Engineering.

[5] Juristo, N.; Moreno, A.M.; Sanchez-Segura,M., "Analysing the impact of usability on Software Design" Journal of Systems and Software, Vol. 80,Issue 9,2007,pp.1506-1516.

[6] Bass, L.; Bonnie, J.; Kates, J., "Achieving Usability Through Software Architecture". Technical Report MU/SEI-2001-TR-005 ESC-TR-2001-005, (2001)

[7] Juristo, N.; Moreno, A.M.; Sanchez-Segura M., "Guidelines for eliciting usability Functionalities". IEEE Transactions on Software Engineering, Vol. 33, N°. 11, November 2007, pp. 744-758.

[8] Mayhew, D., “The Usability Engineering Lifecycle” Morgan Kaufman Publishers Inc. 1999.

[9] Nielsen, J., ”Usability Engineering” Morgan Kaufman Publishers Inc. 1993.

[10] Radven, S.;Graham, J., “Evaluating Usability of Human Computer Interfaces: A Practical Method” Ellis Horwood Limited 1989.

[11] Rosson, M; Carroll, J., ”Usability Engineering” Morgan Kaufman Publishers Inc. 2002.

[12] Florian B., Solarte O.; Reyes,J., “Propuestsa para Incorporar Evaluación y Pruebas de Usabilidad dentro de un Proceso de Desarrollo de Software” Revista EIA,ISSN 1794-1237 Número13,p.123-141, Julio 2010.

[13] Macleod M.; Rengger R., “The Development of DRUM : A Software Tool for Video-assisted Usability Evaluation,” Methods, People and Computers VIII: proceedings of HCI 93, Loughborough, p. 293, 1994.

[14] Ivory,M.; Hearst M., “The state of the art in automating usability evaluation of user interfaces,” ACM Computing Surveys, vol. 33, pp. 470–516, Dec. 2001.

[15] National Institute of Standards and Technology, “NIST Web Metrics Testbed.” Online. Available from: http://zing.ncsl.nist.gov/WebTools/. [Accessed: February 2, 2012], 2005.

[16] UsableNet Inc., “Lift Machine.” Online. Available from: http://lfd.usablenet.com/usablenet_liftmachine.html. [Accessed: February 2, 2012].

[17] Clearleft Ltd “Silverback 2.0 - Guerrilla usability testing software for designers and developers.” Online. Available from: http://silverbackapp.com/. [Accessed: January 6, 2012], 2011.

[18] Microsoft Corporation, “Expression Encoder 4 Pro.” Online. Available from:
http://www.microsoft.com/expression/products/EncoderPro_Overview.aspx.
[Accessed: 06 January 2012], 2010.

[19] Techsmith Corporation, “Morae. Usability testing and market research software”. Online. Available from:http://www.techsmith.com/morae.asp.
[Accessed: 06 January 2012]., 2011.

[20] Aveledo, M., Curtino, D., De la Rosa, A.; Moreno,A., “Avoiding laboratories to collect usability data: two software applications”. Accepted in CISTI 2012.

[21] Frøkjær, E., Hertzum, M.; Hornbæk, K, “Measuring usability: are effectiveness, efficiency, and satisfaction really correlated?,” in Proceedings of the SIGCHI conference on Human factors in computing systems, CHI ’00, (New York, NY, USA), pp. 345–352, ACM, 2000

[22] Seffah,A.,Donyaee, M., Kline,R.; Padda, H., “Usability measurement and metrics: A consolidated model,” Software Quality Journal, vol. 14, pp. 159–178, June 2006.

[23] Sauro,J.; Kindlund, E., “A Method to Standardize Usability Metrics Into a Single Score” CHI 05 Proceedings of the SIGHI. Available from: www.measuring**usability**.com/.../p482-sauro.pdf
[Accessed : February 27, 2012] , 2005.

[24] Bevan, N.; Macleod, M., “Usability measurement in context”. Behaviour and Information Technology 13,132-145, 1994.

[25] Scriven M., “Evaluation Thesaurus” 4th ed. Newbury Park, C.A.. Sage Publications Inc. ,1991.

[26] De la Rosa, A., “TeatroFácil.” Online. Available from: http://www.grise.upm.es/tf
[Accessed: February, 28 2012], 2009

[27] Brooke, J., “A quick and dirty usability scale” Available from www.itu.dk/courses/U/E2005/litteratur/sus.pdf
[Accessed: February , 28 2012],1996

[28] Pérez, C., “Técnica de análisis de datos con SPSS” Pearson Prentice Hall, 2009

[29] Nadim,N., “The Mann-Whitney U: A test for assessing whether two independent samples come from de same distribution” Tutorials in Quantitative methods for psychology 2008, vol. 4(1) pp. 13-20.

[30] Curtino, D., “VuelosFácil”, Online Available from: http://www.fingflights.com
[Accessed : March 6, 2012], 2011.

[31] Curtino, D., “Usability Mechanisms and Their Impact on Usability Attributes: A First Advance ”, Online Avaible from: http://www.fingflights.com/MasterThesis.pdf
[Accesed : March 11, 2012], 2011.

Automatic Generation of Web Interfaces From User Interaction Diagrams

Filipe Bianchi Damiani
Departament of Informatics and Statistics (INE)
Federal University of Santa Catarina
Florianópolis, Brazil
fbd.sk8@gmail.com

Patrícia Vilain
Departament of Informatics and Statistics (INE)
Federal University of Santa Catarina
Florianópolis, Brazil
vilain@inf.ufsc.br

Abstract—**This paper presents the development and validation of a tool for mapping UIDs (User Interaction Diagrams) to JSF (Java Server Faces) web pages. Based on the characteristics of UID elements and JSF components, we designed a tool with a set of rules that automates the process of mapping UIDs to a web based user interface. This tool was developed in Java and its validation was performed by having automatically generated pages compared against pages that were created by designers of an e-commerce website.**

Keywords-User interaction modeling; User interface design; Automatic software generation

I. INTRODUCTION

At the early stages of a system development, one of the software engineering tasks is to gather the functional requirements. Modeling the interaction between the user and the system aims to facilitate this task by pointing out the information that is exchanged across these entities. That interaction can be described in a textual form, using techniques like scenarios and use cases [1], or aided by diagrammatic representations, such as UIDs (User Interaction Diagrams) [2] [3].

UIDs have the purpose of graphically represent the interaction between a user and a system where the exchange of information is intense. They can also help with identifying the necessary navigation in web-centered methodologies such as OOHDM (Object-Oriented Hypermedia Design Method) [4].

UIDs also facilitate the design of user interfaces for web systems. Generally speaking, UID elements that represent user inputs can be mapped to input components of web pages, while UID elements representing the system outputs can be mapped to output components of web pages.

In this work, we present a set of rules that map functional requirements represented in UIDs to a set of dynamic web pages. We also describe the development of a tool that automatically generates these web pages by applying such rules to a set of input files with the description of the given UIDs. The code generated for the web pages utilizes Java Server Faces (JSF), a framework for developing web applications using Java technology [5].

The paper is organized as follows. Section 2 revisits the diagrammatic notation of UIDs. The JSF framework is presented in section 3. Section 4 presents the rules that map UIDs to JSF pages. Section 5 describes the tool developed to automate the generation of web pages from UIDs based on those mapping rules. Section 6 demonstrates the generation of web pages from UIDs that model the user interaction with an existing website. It also compares the actual webpages against the generated ones. Finally, section 7 concludes this work.

II. USER INTERACTION DIAGRAMS

A User Interaction Diagram (UID) is a diagrammatic notation that represents the interaction between users and a system [2] [3].

While gathering functional requirements, one should avoid influencing the description provided by the user. UIDs serve this purpose by providing a simple notation easily understood by both the software engineer and the user that is only concerned about representing the exchange of information between the user and the system, without considering specific aspects of user interface design or content navigation. UIDs can also assist the definition of the conceptual modeling [3].

As defined in [6], a UID is composed by a set of interaction states that are connected by transitions. Interaction states represent the information that is exchanged between the user and the system, while transitions are responsible for changing the focus of the interaction from one state to another. An interaction state is considered the focus of the interaction when it contains the information that is being exchanged between the user and the system at any given time. The exchange of information is represented within the interaction states, although user selections and options may be associated to transitions. Transitions are usually triggered when the user enters or selects some information.

The UID in Fig. 1 represents the interaction between a user and a system during the task *Buy a CD based on a name*, where the user must provide the name for a given CD and the system returns a set of all CDs that match the name provided. Users can then include in the basket those CDs they wish to buy later. For clarity, the name of each notation element utilized in Fig. 1 is included in highlighted rectangles.

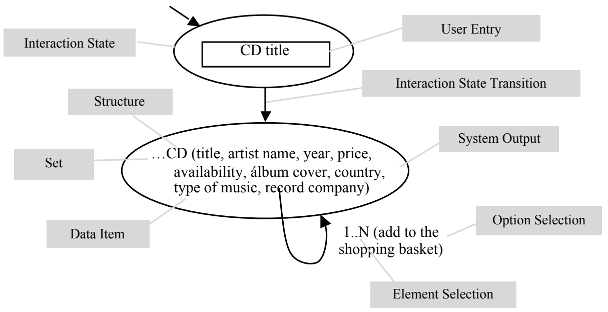

Interaction State
CD title
User Entry
Interaction State Transition
Structure
System Output
Set
…CD (title, artist name, year, price, availability, álbum cover, country, type of music, record company)
Data Item
Option Selection
1..N (add to the shopping basket)
Element Selection

Figure 1. UID Example [6]

III. JAVA SERVER FACES (JSF)

Java Server Faces (JSF) is a framework for developing web applications using the Java technology. It follows the design pattern MVC (Model-View-Control) and its most important differential is the separation between the business model and the visualization [5].

The FacesServelt class is the JSF controller and every request submitted to the system must to be sent to it. For each HTML page instantiated, its components are stored in a tree called View ID. Theses trees, in their turn, are stored in a FacesContext object, which maintains all the information the framework needs in order to manage the components of a page.

Each JSF page usually contains a Java object representing its state that is called Managed Bean. The Managed Bean stores the values of each page field and is responsible for binding the model to the view.

As each JSF visual component has a direct representation in a HTML component, the framework supports the direct rendering of HTML pages from JSF components. Moreover, the framework also supports the implementation of renderers that create interfaces in other languages, increasing even more the flexibility of the application.

The JSF components relevant to this work are:

- form: represents a form for sending data through the JSF servlet;
- panelGrid: produces a table to provide an organized arrangement of the elements grouped in it;
- panelGroup: groups a set of JSF elements. When it is converted to HTML it is mapped to a SPAN or a DIV element;
- column: represents a column of a table;
- dataTable: shows a collection of objects organized as a table;
- outputText: represents a simple text returned by the system;
- outputLabel: represents a label corresponding to a field of a user input;
- inputText: represents a field of a text entry;
- outputLink: produces a hyperlink that takes the user to another page, or another part of the current page, without producing an action event;
- commandLink: it also produces a hyperlink, however it produces an action event and/or the calling of a method of the ManagedBean object;
- commandButton: has the same functionality as the commandLink, but it has the appearance of a button;
- selectManyCheckbox: represents a set of checkboxes from which the user can select a subset;
- selectOneRadio: represents a set of radio buttons from which the user can select only one.

Other components defined in the JSF HTML Tablib (such as outputFormat, message, messages, graficImage, inputTextArea inputSecret, inputHidden, selectManyMenu, selectBooleanCheckbox, selectManyListbox, selectOneMenu, and selectOneListbox) were not utilized here either because they have similar functionality to the previous components or because they do not have a direct relation to UID elements.

IV. MAPPING RULES

During the definition of rules for mapping UID elements to JSF components, our initial idea was to divide the process in two steps: first map the UID elements to abstract widgets, as defined in [7], and then to map abstract widgets to JSF components. The abstract widgets that would be used in this mapping belong to the Abstract Widget Ontology defined in [8]. This ontology is used to specify abstract interfaces that show the information exchange between the user and system, with no reference to technologies neither to the appearance of navigational objects. However, we realized that a lot of the relevant information available in UIDs had been lost after mapping UID elements to abstract widgets. Such loss of information compromised the automatic mapping of UID elements to JSF components mainly because a particular abstract widget could end up being represented by different JSF components.

We realized we needed to define rules that map UID elements directly to JSF components. These rules are presented in Table 1. They were based on the mapping of UIDs elements to abstract widgets, the possible representations of JSF components as abstract widgets, and the relevant information available in UIDs.

TABLE I. MAPPING FROM UIDS ELEMENTS TO JSF COMPONENTS

UID Elements	JSF Components
text	outputText;
data item (system output)	if the data item is the source of any transition, it is mapped to a commandLink; otherwise, it is mapped to an outputText;
Structure (system output)	if the structure is not the source of any transition and does not contain elements, it is mapped to an outputText;

UID Elements	JSF Components
	if the structure is the source of a transition and does not contain elements, it is mapped to a commandLink, ; if the structure contains elements, it is mapped to a panelGrid with an outputLabel. Each element within the structure is mapped according to its type;
set of data items (system output), set of structures (system output)	if the set is not the source of any transition, it is mapped to a dataTable; otherwise, its mapping is done as in the mapping of a transition;
data item (user entry)	inputText;
structure (user entry)	panelGrid with an outputLabel. Each element within the structure is mapped according to its type and the resulting JSF components are included into the panelGrid (mapping rules are recursively applied to nested elements); if the structure does not have elements , it is mapped to an inputText with an outputLabel;
set of data items (user entry)	if the set has an upper limit (0..*), it is mapped to an inputText with an outputLabel; otherwise, inputText components must be replicated up to the number of required entries;
set of structures (user entry)	if the set has an upper limit and the structure does not contain elements, its mapping is done as in a set of data items; if the set has a relatively small upper limit, it is mapped to a dataTable and each element to a column; if the set has no upper limit, it is mapped as a single structure;
enumerated user entry	if only one item can be chosen, it is mapped to a selectOneRadio; if more than one item can be chosen, it is mapped to a selectManyCheckbox;
selection between two data items (or)	selectManyCheckbox;
selection of a data item (xor)	selectOneRadio;
interaction state	it is mapped to a form . The JSF components that correspond to the elements of the interaction state are added to the form. Note that a single form component can implement more than one interaction state;
Sub-State	form;
Transition with the Selection of an Option	if the source is a data item, ignore it (its mapping was done as part of mapping the data item); if the source is a structure, it is mapped to a commandLink added to the panelGrid obtained from mapping the structure; if the source is an interaction state or sub-state, it is mapped to a commandLink added to the form that was obtained from mapping the input elements in that state or sub-state;
transition with selection of elements	if the source is a set of data items and just one element can be selected, the set is mapped to a dataTable with a column containing commandLinks around the items; if the source is a set of data items and more than one element can be selected, the set is mapped to selectmanyCheckbox and the transition is mapped to a commandLink; if the source is a set of structures and just one structure is selected, the set is mapped to a dataTable with an additional column containing commandLinks around the selected option; if the source is a set of structures and more than one structure is selected, the set is mapped to a dataTable and a column containing a selectManyCheckbox to represent the selection and a commandLink added to the form where the dataTable appears.
Call of another UID, Call from another UID, pre-conditions, post-conditions, parameters and notes	ignored as there are no matching JSF components

V. JSF Page Generation Tool

We developed a tool that automates the application of the UID-to-JSF mapping rules shown in the previous section. This tool reads UID information stored in one or more XML (Extensible Markup Language) documents[1] and generates JSF pages. The tool was developed for JSF version 2.0.

A. *Requirements List*

We identified 32 requirements for our tool: 18 requirements related to the implementation of the mapping rules, 13 requirements related to the implementation of a method that returns the string declaration of each of the target JSF components, 1 requirement related to the module responsible for reading UID files, and 1 requirement related to the development of an interface between UID files and the generation of JSF pages . The details related to these requirements are out of the scope of this paper.

[1] UID diagrams are stored as XML to facilitate its interchange across different applications.

B. Component Model

Fig. 2 shows the component model utilized in JSF pages that are automatically generated by the tool. According to the mapping rules defined, the generated JSF page can include any of the JSF components explained in Section 3, i.e., Form, OutputText, OutputLabel, OutputLink, CommandLink, InputText, PanelGrid, DataTable, Column, SelectManyCheckbox, SelectOneRadio, SelectItem, and SelectItems.

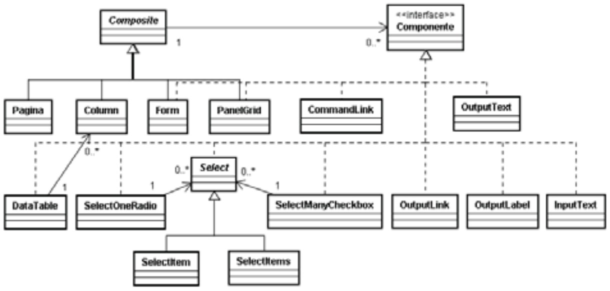


Figure 2. Component model for automatically generated JSF Pages

C. Iterations

Our mapping tool was developed in three iterations. During the first iteration, we developed the requirements *Integration of the Module for Reading the UID Files* followed by the requirements *Mapping the Interaction State and Initial Interaction State* and *Mapping the Sub-State of an Interaction State*. In the second iteration, we developed the mapping of all UID elements. Finally, in the third iteration we developed the remaining requirements.

VI. Running Example

In order to validate the rules that map UIDs to JSF pages, a set of JSF pages automatically generated by our mapping tool was compared against actual web pages available at www.amazon.com for the common task of buying a CD from the results of an advanced search. The corresponding use case for this task is shown below.

Use Case: Buying a CD from an Advanced Search
Description:
1. The user enters some keywords, artist's name, CD title or record label, and chooses 'CD' as the media format.
2. The system returns a set of CDs that match the entries. For each CD listed, the following data are given: CD title, artist's name, year, recommendation, stock price, stock quantity, price of a new one (from a reseller), and price of a used one (from a reseller).
3. The user selects one CD from the set and the system shows the specific information about that CD, including the CD title, artist's name, recommendation, stock price, stock quantity, price of a new one, price of a used one and artist's biography.
5. If the user wants to buy that CD or has interest on it, he or she can instantly proceed with the purchase (one-click ordering), add it to the shopping cart to buy it later or instead simply add it to a wishing list.

Fig. 3 shows the UID corresponding to the use case above. This UID was used as the input to our mapping tool. Next, we analyze and compare the actual web pages against the ones automatically generated by our tool. It is important to note that the UIDs utilized as input must be complete in their notation in order to guarantee a proper generation of the target JSF pages. More specifically, one needs to make sure that the state transition options are given (i.e. the options users must select that will cause a transition to fire), structure and data item cardinalities are specified and also that the names given to data items are consistent throughout all UIDs.

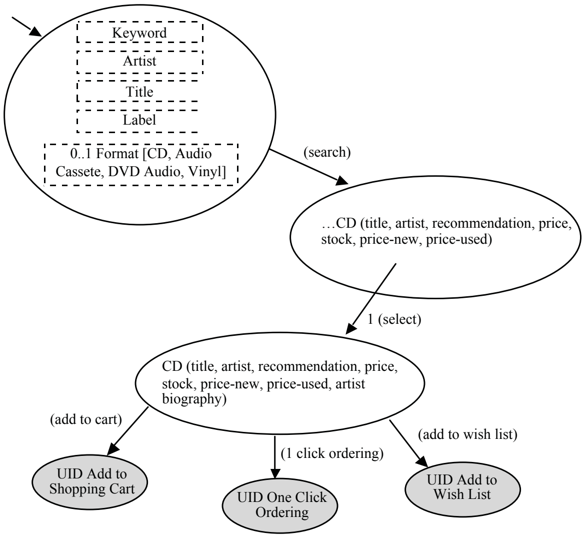
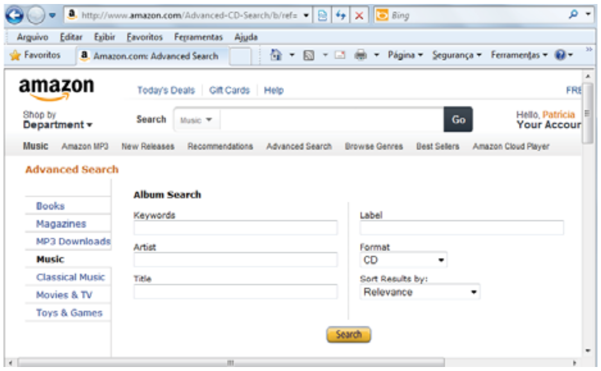

Keyword
Artist
Title
Label
0..1 Format [CD, Audio Cassete, DVD Audio, Vinyl]
(search)
…CD (title, artist, recommendation, price, stock, price-new, price-used)
1 (select)
CD (title, artist, recommendation, price, stock, price-new, price-used, artist biography)
(add to cart)
(1 click ordering)
(add to wish list)
UID Add to Shopping Cart
UID One Click Ordering
UID Add to Wish List

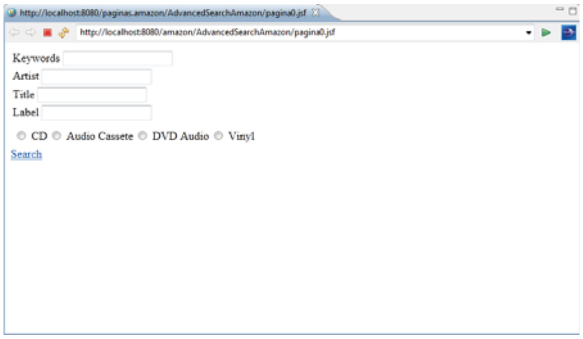
Figure 3. UID: Buying a CD from an Advanced Search

The actual web page corresponding to the initial interaction state of the UID from Fig. 3 is shown in Fig. 4, while the page generated by the mapping tool is presented in Fig. 5. Both pages are similar in that they provide four text entry fields for entering keywords, artist name, CD title and record label. However, the selection of a media format in the actual page (Fig. 4) is accomplished by selecting an item from a list, while in the automatically generated page (Fig. 5) the available media formats appear mapped as a set of radio buttons. Another difference is that the *Search* button in the actual page was mapped to a link named *Search* in the generated page.



Format: CD
Sort Results by: Relevance

Figure 4. Advanced search web page



http://localhost:8080/paginas.amazon/AdvancedSearchAmazon/pagina0.jsf

http://localhost:8080/amazon/AdvancedSearchAmazon/pagina0.jsf

Keywords

Artist

Title

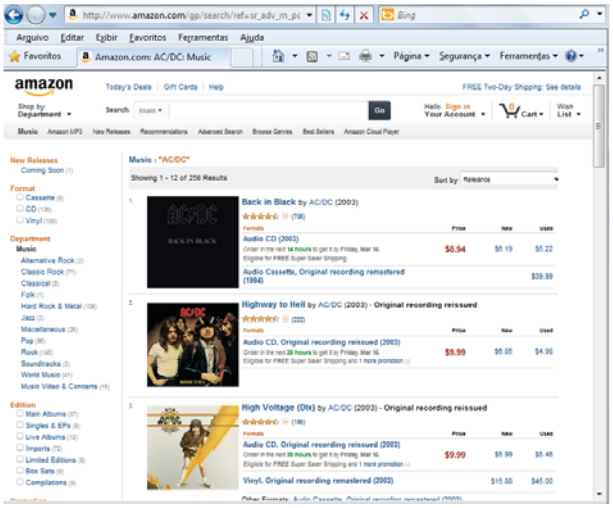
Label

○ CD ○ Audio Cassete ○ DVD Audio ○ Vinyl

Search

Figure 5. Page automatically generated for the initial interaction state of the UID 'Buying a CD from an Advanced Search'

Fig. 6 shows the original web page containing the results of an advanced search for CDs where 'AC/DC' was given as the artist's name. The media format had to be set to 'CD', while remaining fields in the form were left blank. T he same search was carried out through the automatically generated page and the resulting page is shown in Fig. 7.



1. Back in Black by AC/DC (2003)

Formats	Price	New	Used
Audio CD (2003)	$8.94	$6.19	$5.22
Audio Cassette, Original recording remastered (1994)			$39.99

2. Highway to Hell by AC/DC (2003) - Original recording reissued

Formats	Price	New	Used
Audio CD, Original recording reissued (2003)	$9.99	$6.05	$4.98

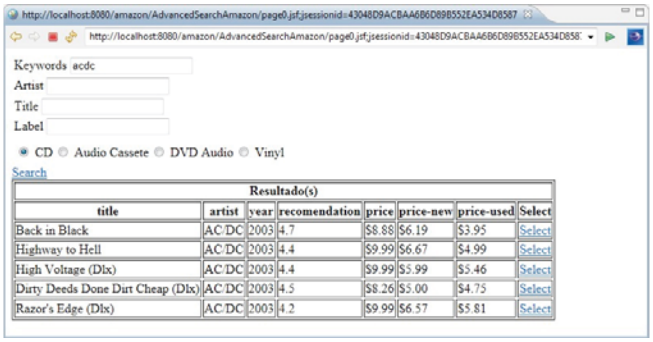
3. High Voltage (Dlx) by AC/DC (2003) - Original recording reissued

Formats	Price	New	Used
Audio CD, Original recording reissued (2003)	$9.99	$5.99	$5.46
Vinyl, Original recording remastered (2003)		$15.88	$45.00

Figure 6. Advanced search results web page

As observed in Fig. 7, the initial interaction state and the second interaction state were mapped into the same JSF page, thus keeping both the advanced search input form and the advanced search results in the sam e page. That differs from the navigation implemented in the original w eb site, where search results will appear in a separate page . Regarding the presentation of the results, our tool mapped the resulting list of CDs (in the second interaction state of the given UID) to a table where each result appears in a different row having its data separated in columns (Fig. 7), following the rule that maps a set of structures of a system output (*if the set is not the source of any transition, it is mapped to a dataTable*). The results shown by the actual page (Fig. 6) are also presented in a table that, despite some visual design differe nces, is sim ilar in content to the generated page.

Both the generated and actual results page s allow the user to select a C D in order to see the data associated to it by clicking on a link. In the actual page, clicking on the CD cover or title activates that link, w hile in the autom atically generated page the user must click on *Select* in order to follow the corresponding link. Navigating that link corresponds to transitioning to the third interaction state of the U ID showed earlier in Fig. 3.



http://localhost:8080/amazon/AdvancedSearchAmazon/page0.jsf;jsessionid=43048D9ACBAA6B6D89B552EA534D8587

http://localhost:8080/amazon/AdvancedSearchAmazon/page0.jsf;jsessionid=43048D9ACBAA6B6D89B552EA534D858

Keywords acdc

Artist

Title

Label

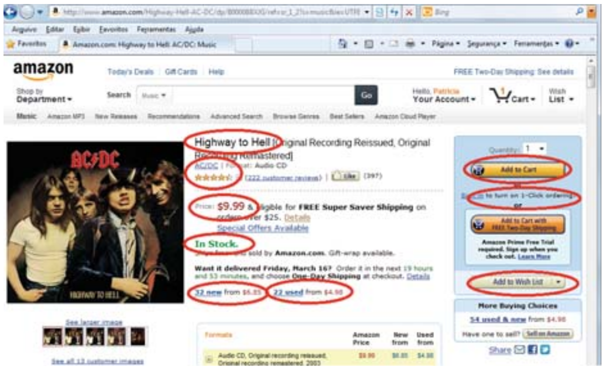
● CD ○ Audio Cassete ○ DVD Audio ○ Vinyl

Search

Resultado(s)							
title	artist	year	recomendation	price	price-new	price-used	Select
Back in Black	AC/DC	2003	4.7	$8.88	$6.19	$3.95	Select
Highway to Hell	AC/DC	2003	4.4	$9.99	$6.67	$4.99	Select
High Voltage (Dlx)	AC/DC	2003	4.4	$9.99	$5.99	$5.46	Select
Dirty Deeds Done Dirt Cheap (Dlx)	AC/DC	2003	4.5	$8.26	$5.00	$4.75	Select
Razor's Edge (Dlx)	AC/DC	2003	4.2	$9.99	$6.57	$5.81	Select

Figure 7. Page automatically generated from the second interaction state of the UID 'Buying a CD from an Advanced Search'

Fig. 8 shows the actual page that corresponds to the third interaction state of the given UID. In this page, the user can add the selected CD to the shopping cart, buy the CD with one click ordering or add the CD to the wish list.



Arquivo Editar Exibir Favoritos Ferramentas Ajuda
Favoritos
Amazon.com: Highway to Hell: AC/DC: Music
Página Segurança Ferramentas
amazon
Today's Deals | Gift Cards | Help
FREE Two-Day Shipping: See details
Shop by Department
Search
Go
Hello, Patricia Your Account
Cart
Wish List
Music Amazon MP3 New Releases Recommendations Advanced Search Browse Genres Best Sellers Amazon Cloud Player
Quantity: 1
Add to Cart
to turn on 1-Click ordering
Add to Cart with FREE Two-Day Shipping
Add to Wish List
More Buying Choices
Sell on Amazon
Share
See larger image
See all 13 customer images

Highway to Hell [Original Recording Reissued, Original Recording Remastered]

AC/DC | Format: Audio CD

(222 customer reviews) | Like (397)

Price: $9.99 & eligible for FREE Super Saver Shipping on orders over $25. Details

Special Offers Available

In Stock.

Ships from and sold by Amazon.com. Gift-wrap available.

Want it delivered Friday, March 16? Order it in the next 19 hours and 53 minutes, and choose One-Day Shipping at checkout. Details

32 new from $6.35 22 used from $4.98

Amazon Prime Free Trial required. Sign up when you check out. Learn More

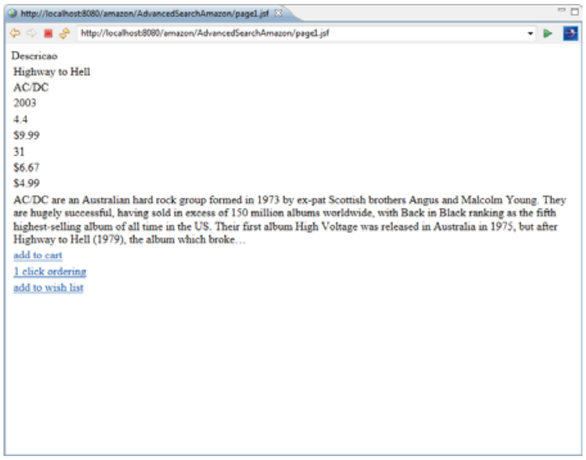
54 used & new from $4.98

Have one to sell?

Formats	Amazon Price	New from	Used from
Audio CD, Original recording reissued, Original recording remastered, 2003	$9.99	$6.35	$4.98

Figure 8. CD detail web page

The automatically generated page that corresponds to the third interaction state of the U ID is shown in Fig. 9. The main difference here is that the actual page (Fig. 8) contains labels that identity the details presented about the CD that w as selected (for exam ple, there is the label *Price* with the value *$9.99*) whereas such labels are not included in the generated page.


http://localhost:8080/amazon/AdvancedSearchAmazon/page1.jsf
http://localhost:8080/amazon/AdvancedSearchAmazon/page1.jsf

Descricao
Highway to Hell
AC/DC
2003
4.4
$9.99
31
$6.67
$4.99
AC/DC are an Australian hard rock group formed in 1973 by ex-pat Scottish brothers Angus and Malcolm Young. They are hugely successful, having sold in excess of 150 million albums worldwide, with Back in Black ranking as the fifth highest-selling album of all time in the US. Their first album High Voltage was released in Australia in 1975, but after Highway to Hell (1979), the album which broke…
add to cart
1 click ordering
add to wish list

Figure 9. Page automatically generated from the third interaction state of the UID 'Buying a CD from an Advanced Search'

VII. Conclusions

This paper presented a set of mapping rules for generating web pages from user interaction diagrams, a technique that models the exchange of information between users and systems. It also presented the results of a prototypical tool that automates the generation of JSF pages by applying those rules.

As a way to validate our tool, the resulting pages were compared to the ones available in an actual website. This comparison showed that the mapping rules can generate JSF pages consistent with the original user task modeled through UIDs.

The first important contribution of this work is that generated pages can be utilized as a good starting point for mapping user functional requirements to their implementation. More specifically it can help user interface designers by providing an initial prototype with all the components needed so that users can exchange information with the target application.

The second contribution of this work is that the generated pages can be utilized during requirements gathering to more easily engage end users in the process of validating those requirements. A number of users may feel more comfortable with validating requirements through a functional prototypical user interface than by looking at diagrams or abstract models.

The UIDs we utilize as input are created by a graphical editor available separately as an IDE plug-in (available at http://www.uid.inf.ufsc.br/). This editor stores XML files that are then passed in to the JSF generation tool. As a future improvement, we intend to integrate the JSF generation functionality with the editor so that a given UID can be directly run as JSF page from within the graphical editor pane.

As a continuation of our research work, we intend to use the MDD approach as a basis for mapping UIDs into concrete user interfaces, similarly to what is done in [9, 10, 11]. Existing work [12, 13] provides a good analysis about MDD tools applicable to user interface generation. Our idea is, instead of using a set of fixed mapping rules, to choose a transformation language that can be utilized for that purpose.

References

[1] I. Jacobson, "The Use-Case Construct in Object-Oriented Software Engineering", Scenario-Based Design: Envisioning Work and Technology in System Development, John Wiley & Sons, 1995, pp. 309-337.

[2] P. Vilain, D. Schwabe, and C.S. de Souza, "A Diagrammatic Tool for Representing User Interaction in UML", UML 2000 Conference, 2000, pp. 133-147.

[3] P. Vilain, *User Interaction Modeling in Hypermedia Applications*, PhD Thesis, PUC-Rio, 2002. (in Portuguese)

[4] N. Güell, D. Schwabe, and P. Vilain, "Modeling Interactions and Navigation in Web Applications", Second International Workshop on the World Wide Web and Conceptual Modeling (WCM2000), 2000, pp. 115-127.

[5] JavaServer Pages Technology, http://www.oracle.com/technetwork/java/javaee/jsp, December 2011.

[6] P. Vilain, *Implementation of a Framework to Support the Representation of Functional Requirements in the Software Process,* Final Project Report, Federal University of Santa Catarina, 2003. (in Portuguese)

[7] L.P. Remáculo, *Customization of User Interaction Diagrams and Mapping to Abstract Widgets Ontology*, BSc. Thesis, Federal University of Santa Catarina, 2005. (in Portuguese)

[8] S.S. Moura, Development of Ontology Driven Interfaces for Semantic Web Applications, Master Thesis, PUC-Rio, 2004. (in Portuguese)

[9] J. Vanderdonckt, "A MDA-Compliant Environment for Developing User Interfaces of Information Systems", 17th international conference on Advanced Information Systems Engineering (2005), 2005, pp. 16-31.

[10] J.-S. Sottet, G. Calvary, J. Coutaz, J.-M. Favre, "A Model-Driven Engineering Approach for the Usability of Plastic User Interfaces", Proc. of Engineering Interactive Systems, 2007, pp. 22-24.

[11] J.I. Panach, S. España, A. Moreno, Ó. Pastor, "Dealing with Usability in Model Transformation Technologies", 27th International Conference on Conceptual Modeling (ER 2008) (ER), 2008, pp. 498-511.

[12] R.Schaefer, "A Survey on Transformation Tools for Model Based User Interface Development", 12th International Conference on Human-Computer Interaction: Interaction Design and Usability (HCI) 2007, pp. 1178-1187.

[13] J.M.G. Calleros, A. Stanciulescu, J. Vanderdonckt, J.P. Delacre, and M. Winckler, "A Comparative Analysis of Graph Transformation Engines for User Interface Development", 4th International Workshop on Model-Driven Web Engineering (MDWE), 2008, pp. 16-30.

Semantic Technology Recommendation Based on the Analytic Network Process

Filip Radulovic
Ontology Engineering Group
Facultad de Informática, Universidad Politécnica de Madrid
Madrid, Spain
fradulovic@fi.upm.es

Raúl García-Castro
Ontology Engineering Group
Facultad de Informática, Universidad Politécnica de Madrid
Madrid, Spain
rgarcia@fi.upm.es

***Abstract*—Semantic technologies have become widely adopted in recent years, and choosing the right technologies for the problems that users face is often a difficult task. This paper presents an application of the Analytic Network Process for the recommendation of semantic technologies, which is based on a quality model for semantic technologies. Instead of relying on expert-based comparisons of alternatives, the comparisons in our framework depend on real evaluation results. Furthermore, the recommendations in our framework derive from user quality requirements, which leads to better recommendations tailored to users' needs. This paper also presents an algorithm for pairwise comparisons, which is based on user quality requirements and evaluation results.**

I. Introduction

Semantic technologies provide new ways to express in machine processable formats knowledge and data that can be exploited by software, and we have seen an exponential growth of these technologies in recent years.

One of the characteristics of semantic technologies is the existence of several different types of technologies. It is often the case that when solving certain problems, users have to use various semantic technologies that belong to different types. In some cases, especially for less experienced users, selecting the right technologies for solving a problem can be a difficult task.

Multiple criteria decision making (MCDM) methods are widely accepted and have been used across various fields, including Software Engineering. These methods have also been successfully applied in software selection problems, which is regarded as an important and rather difficult problem, such as in the selection of ERP systems [1].

Different problems often require different system functionalities and one functionality might not be relevant for every problem. In MCDM recommendation frameworks, usually all functionalities are considered and, therefore, some functionalities that are not important for a problem are taken into account, which might lead to complexity and poor recommendations.

Furthermore, the comparison of alternatives is usually performed manually by a group of experts. In some cases, expert-based comparisons can be difficult because there are no experts that are familiar with every available alternative. Besides, the addition of new alternatives would require experts to perform additional comparisons.

Furthermore, expert-based comparisons are highly subjective and there are cases when we have objective evaluation results in which we can ground recommendations.

This paper presents an application of the Analytic Network Process (ANP) for the recommendation of semantic technologies. The recommendation framework is based on a quality model for semantic technologies, and the recommendations are based on user quality requirements.

The comparison of alternatives in our framework depends on real semantic technology evaluation results provided by the SEALS European project[1]. In this paper, we also present an algorithm for the comparison of alternatives, which uses those results together with user quality requirements.

The reminder of this paper is organized as follows. Section II presents the best-known MCDM methods. Section III gives an overview of the proposed recommendation framework, while Section IV describes the semantic technology quality model. Section V describes the ANP and, afterwards, an algorithm for pairwise comparisons based on quality requirements and evaluation results is presented in Section VI. Section VII presents in detail the ANP framework for the semantic technologies, while Section VIII gives an illustrative example. Finally, Section IX draws some conclusions and includes ideas for future work.

II. Related Work

When facing the complex decision of selecting the best solution between a group of alternatives that can be compared according to different conflicting criteria, decision makers use MCDM methods that help them to better structure the problem and make better decisions. In MCDM problems, alternatives represent concrete products, services or actions that will help in achieving a goal, while criteria represent the characteristics of the alternatives that are important for making a decision.

A large number of MCDM methods have been defined to date. However, no method is considered to be the best to be applied in every decision making problem [2]. Next, we describe the most relevant MCDM methods in the literature, and give examples of their use in the Software Engineering and in the semantic technology fields.

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

PROMETHEE methods [3] belong to a family of outranking methods which are based on preference analysis, and different PROMETHEE methods can be used depending on the goal to be achieved. Alternatives are compared using one of six types of preference functions for each criterion, and the results are synthesized into positive and negative outranking flows. The positive outranking flow of an alternative determines how much it dominates the others, while the negative outranking flow shows how much an alternative is dominated by the others; these positive and negative outranking flows can be synthesized into one final indicator.

One of the drawbacks of the PROMETHEE methods is that they do not include any particular procedure for the calculation of the importance (weights) of criteria [4], which is a key information needed for obtaining the outranking flows.

The Analytic Hierarchy Process (AHP) [5] is a well-known method developed by Thomas L. Saaty. It requires the formulation of the decision problem into a hierarchical structure of goal, criteria, and alternatives.

The key concept in the AHP is a pairwise comparison, which is used to determine the importance of the criteria, as well as to compare the alternatives according to each criterion. Saaty also provides a scale for pairwise comparisons, which consists of natural numbers ranging from 1 (equal importance) to 9 (extreme importance). If number x is assigned when comparing alternative a to b, then a reciprocal value ($1/x$) is assigned when comparing alternative b to a. Furthermore, Saaty developed a method for verifying the consistency of pairwise comparisons, which is regarded as the main advantage of the AHP [6].

The Analytic Network Process (ANP) [7] is another method developed by Saaty, which is a generalization of the AHP where the decision problem is formulated as a network of criteria and alternatives. The main difference between the ANP and the AHP is that the ANP is designed for those problems in which the criteria in the decision process depend on each other.

In recent years, we have seen applications of the PROMETHEE methods in Software Engineering, for example, in the selection of web services [8], [9]. The AHP has been adopted in many different fields because of its simplicity and ease of use, and it is described in the literature as one of the most widely used MCDM methods [10]. In the Software Engineering field, the AHP has been frequently used for software selection problems [11]. The ANP has also been applied successfully in various problems, including Software Engineering ones, such as the selection of ERP systems [12] and of web services [13].

In the semantic technology field, we have only found one example of applying MCDM methods. In her work, Mochól developed an AHP-based framework for manual and (semi-) automatic selection of ontology matching approaches [14].

Mochól's work is focused only on one specific type of semantic technologies, i.e., ontology matching tools, while in our case multiple types of technologies are taken into account simultaneously.

III. Overview of the Recommendation Framework

This section presents the overview of the software recommendation framework. Following a typical MCDM framework, alternatives would be a set of software products to be compared according to different software quality characteristics (i.e., criteria). Then, the output would be a ranking of alternatives.

Next, we present the differences of our framework (depicted in Fig. 1) compared to such typical approach.

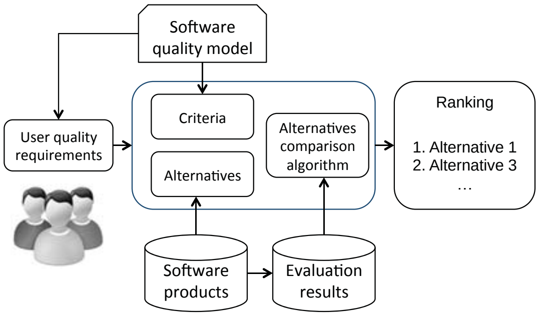


Fig. 1: Overview of the recommendation framework.

- *Software quality model.* When using a MCDM method in a software recommendation process, the criteria usually are software quality characteristics. Therefore, software quality models are a good starting point for the recommendation problem.
  In those cases where there are many dependencies among quality characteristics, which is usual in Software Engineering and in our case, it is recommended to adopt the ANP, to take advantage of these dependencies.
- *User quality requirements.* Usually, criteria that are taken into account in MCDM problems cover all the quality characteristics defined. In our case, solving a problem does not require every characteristic and, therefore, the criteria to take into account consist only of those specified by the user.
- *Alternatives.* In our framework, recommendation covers not one type of software product, but different types of products. User requirements can be satisfied either by a single product or by a combination of them. Therefore, an alternative consists of a combination of software products that together cover a set of common functionalities.
- *Comparison algorithm.* The comparison of alternatives is in most cases performed manually based on subjective opinions made by experts. In our case, the task of comparing the alternatives by experts is difficult because there are no experts with expertise in every software product type. Therefore, in order to overcome this problem and to enable the automatic comparisons, we propose an automated comparison algorithm that is based on evaluation results and user quality requirements.
- *Evaluation results.* For the previously mentioned algorithm a set of evaluation results for the different types of software products is needed. In our case, we use a

corpus of semantic technology evaluation results that have been produced in the SEALS project. These results cover five types of semantic technologies (ontology engineering tools, ontology matching tools, reasoning systems, semantic web services, and semantic search tools), which have been evaluated according to different characteristics (scalability, conformance, interoperability, accuracy, etc.).

IV. Quality Model For Semantic Technologies

In the Software Engineering field, software quality models provide a common framework for software quality specification and evaluation by specifying a consistent terminology for software quality and by providing guidance for its measurement.

Quality models consist of a hierarchy of quality characteristics, which are further decomposed into sub-characteristics. For every quality sub-characteristic, a quality measure or a set of quality measures is defined, which are used for measuring and provide insight of the particular sub-characteristic.

In the case of the AHP, which requires a hierarchical structure in the model, hierarchical quality models (e.g., ISO 9126 [15] or SQuaRE [16]) are very convenient, and different authors have used quality models based on the ISO 9126 together with the AHP [17], [18], [19].

In the semantic technology domain, a quality model for semantic technologies has been proposed [20], which extends the ISO 9126 quality model. The quality model describes 14 quality characteristics and sub-characteristics, and 55 quality measures. Furthermore, for every quality measure, a formula for its calculation is defined [21]; these formulas formally specify the dependencies between measures.

V. Analytic Network Process

The inputs in the ANP are the different alternatives and the set of criteria used to compare them, and the output is a ranking of the alternatives with respect to the criteria.

The ANP consists of several consecutive steps [7]:

1) The first step of a decision process is to define a model of a problem, and it is often referred as the most important step [22]. In the ANP, the model consists of a network of elements (criteria and alternatives) and of the dependencies between them. Elements are organized into clusters, and dependencies between clusters are also defined; these dependencies are deduced based on the existing dependencies between elements.
2) For the defined network, a supermatrix is formulated. The rows and columns of the supermatrix are related to the elements in the network, and are grouped into the corresponding clusters. This way, a supermatrix consists of several sub-matrices, each related to two clusters in the network. The entries of the supermatrix represent the influence priorities of one element over another, e.g., the entry in the *i*-th row and the *j*-th column represents the importance of the *i*-th element over the *j*-th element.
3) The influence priorities are calculated with pairwise comparisons, similarly as in the AHP. For every column in the supermatrix, a pairwise comparison is performed for every cluster in a row separately, and it includes only the elements that influence the one related to the observed column. The standard Saaty's scale for the pairwise comparisons [5] is used, and the eigenvector of the comparison is calculated. The results from the eigenvector are then inserted into the corresponding positions of a column in the supermatrix. If two elements are not connected, a zero is entered.
   In the ANP, criteria are also compared with respect to each alternative. In the pairwise comparisons, every criteria that contributes to a certain alternative is compared to determine the level of contribution to that alternative. The results are then entered as the corresponding elements in the supermatrix. This step is particularly significant when observing the influence of criteria on a single alternative.
4) As the supermatrix has to be stochastic (i.e., the sum in every column has to be one), it has to be weighted. This is done by determining the importance of each block of clusters in the supermatrix in a set of pairwise comparisons performed similarly to the previous step. Then, each entry in the supermatrix is multiplied with the importance of the block the entry belongs to.
5) The next step is the convergence of the weighted supermatrix. The weighted supermatrix is put to a power of an increasing number, until the limit supermatrix is obtained, i.e., that in which the values in every column are equal.
6) The ranking of the alternatives is obtained from the limit supermatrix. The value in every row that corresponds to an alternative represents the result for that alternative in the decision process, which is used to determine the order of alternatives. A higher value denotes a better result, and is used for sorting the alternatives from best to worst.

VI. Alternatives Comparison Algorithm

As presented in Section V, in the third step of the ANP alternatives are compared with respect to each criterion. In this section we present an algorithm for the automatic comparison of alternatives, which is based on the standard 1-9 Saaty's comparison scale.

The inputs of the algorithm are a threshold value t, extracted from the user quality requirements, and evaluation results for the two alternatives, a_1 with the result v_1, and a_2 with the result v_2. The output is a natural number on Saaty's scale, which tells to which degree one alternative is preferable over the other.

There are several cases, with respect to the four types of scale [23] for a quality measure:

- *Nominal scale.* Nominal scale is a type of scale in which results are descriptive labels with no significance of order. We distinguish two possible cases, depending on whether the evaluation result meets the threshold:
  - If only one result is equal to the threshold, e.g., v_1, when comparing a_1 to a_2 a value of 9 (extreme importance) is assigned and, according to the pairwise

comparison rule, a value of 1/9 is assigned when comparing a_2 to a_1.
  - If both results meet the threshold or none of them does, both alternatives are of equal importance. Therefore, a value of 1 is assigned in both comparisons.
- *Ordinal, interval or ratio scale.* Ordinal scale is a type of scale in which results are also descriptive labels, but with significance of order. In interval and ratio scales the results are numerical values and the difference between two results can be calculated. This leads to the following possible cases:
  - If v_1 is equal or better than the threshold, while v_2 is worse, a value of 9 is assigned when comparing a_1 to a_2, and a value of 1/9 when comparing a_2 to a_1.
  - If both alternatives are worse or better than the threshold, they are of equal importance with respect to the requirement. However, they are still compared, and a value of 5 (strong importance) is assigned when comparing the better alternative to the worst. Similarly as in previous cases, a value of 1/5 is assigned when comparing the worse alternative to the better.
  - If both results are equal, a value of 1 is assigned in both comparisons.

  In the ordinal, interval, and ratio scales, when comparing two values, the nature of the criterion determines which result is better. Two possible cases exist: higher-best scale, in which the higher value denotes a better result, and lower-best scale, in which the lower value denotes a better result.

VII. The ANP for Semantic Technologies

In this chapter we describe the particularities of the ANP with respect to the semantic technology domain.

A. ANP Network for Semantic Technologies

The quality model for semantic technologies provides a good starting point in defining the ANP network. In several consecutive steps, we transformed the quality model into the network:

1) Every quality measure from the quality model becomes an element of the network.
2) As every quality measure is used for measuring a sub-characteristic, the network elements are grouped into clusters, each containing those measures that are related to a certain sub-characteristic.
3) Based on the formulas for obtaining the quality measures, defined in the quality model, the dependencies between the measures are deduced. Every two dependent elements are then connected with an arc; the element where the arc begins depends on the element where the arc ends.
4) Based on the dependencies between elements, dependencies between clusters are defined in such a way that two dependent elements imply a dependence between their clusters.

Due to space reasons, we cannot present the whole network. Therefore, on Fig. 2 we present only one part of the network where seven quality measures are grouped into four clusters; dependencies between measures are represented with arcs.

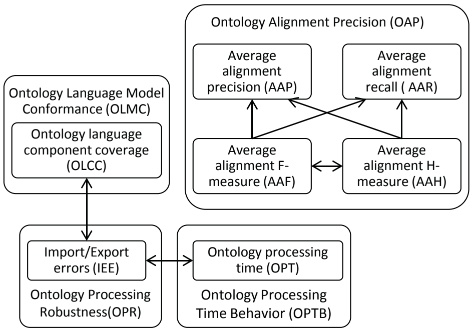

Ontology Alignment Precision (OAP)
Average alignment precision (AAP)
Average alignment recall (AAR)
Average alignment F-measure (AAF)
Average alignment H-measure (AAH)
Ontology Language Model Conformance (OLMC)
Ontology language component coverage (OLCC)
Import/Export errors (IEE)
Ontology Processing Robustness(OPR)
Ontology processing time (OPT)
Ontology Processing Time Behavior (OPTB)

Fig. 2: Part of the semantic technology ANP network.

The network in this case consists only of quality characteristics (criteria), and alternatives are not included. The reason for this is that recommendations are based on user quality requirements, and alternatives are formed and inserted into the network only after the quality requirements are specified.

B. Supermatrix

Based on the previously defined network, a supermatrix was constructed. It consists of several sub-matrices where every sub-matrix is related to two clusters of the network, one at the left of the matrix and one at the top.

For every column in a supermatrix, influence priorities for the criteria were calculated in pairwise comparisons. This task, unlike the comparison of alternatives, was performed by a team of experts in semantic technologies. Every two elements in two rows within a certain cluster that have influence on an element in a column are compared in a pairwise comparison with the following question: "given an element in the column, which of the two elements in the rows has more influence?".

Table I shows an example of a pairwise comparison in which the priorities of measures with respect to *Average alignment F-measure* are calculated. We can see from the network (Fig. 2) that *Average alignment F-measure* depends on *Average alignment H-measure*, *Average alignment precision*, and *Average alignment recall*. Therefore, those three measures are compared in the pairwise comparison to determine their importance. For example, the *Average alignment precision* has a strong plus over the *Average alignment H-measure*, which implies the value 6 in their comparison.

Column *Importance* gives the overall importance for each measure. This comparison suggests that, e.g., *Average alignment precision* influences *Average alignment F-measure* with 0.462 degree of importance.

TABLE I: Pairwise comparisons of measures with respect to Average alignment F-measure.

AAF	AAP	AAR	AAH	Importance
AAP	1	1	6	0.462
AAR	1	1	6	0.462
AAH	1/6	1/6	1	0.076

Using the method provided by Saaty, we verified the consistency of every pairwise comparison in our supermatrix. The method is based on the calculation of the consistency ratio, whose value is limited to 0.1, and which is satisfied in all the pairwise comparisons performed.

Table II presents the part of the supermatrix that is related to the part of the network presented on Fig. 2. The priorities in the supermatrix were obtained through pairwise comparisons performed by experts, and we can see that the values obtained in Table I are inserted into the appropriate positions as a sub column in the supermatrix (column AAF).

TABLE II: Part of the supermatrix.

	OLCC	**IEE**	**OPT**	**AAP**	**AAR**	**AAF**	**AAH**
OLCC	0	1	0	0	0	0	0
IEE	1	0	1	0	0	0	0
OPT	0	1	0	0	0	0	0
AAP	0	0	0	0	0	0.462	0.462
AAR	0	0	0	0	0	0.462	0.462
AAF	0	0	0	0	0	0	0.076
AAH	0	0	0	0	0	0.076	0

The influence priorities of the clusters (i.e., of each block in a supermatrix) are calculated in an analogue way as the influences of their elements. Table III shows the priorities for the previously-presented part of the network.

TABLE III: Cluster priorities.

	OLMC	**OPR**	**OPTB**	**OAP**
OLMC	0	0.15	0	0
OPR	1	0.204	1	0
OPTB	0	0	0	0
OAP	0	0	0	1

VIII. Illustrative Example

In this section, we describe an example of using the proposed recommendation framework. In it, we assume that a user needs to modify existing ontologies (i.e., semantic models) and then match their concepts to other ontologies. For this task, two types of tools are needed, ontology engineering and ontology matching tools.

Table IV shows the user quality requirements in terms of a quality measure and a threshold, as well as the tools that at least cover one requirement; T1 and T2 are ontology engineering tools and T3 and T4 are ontology matching tools. The set of alternatives will consist of the four combinations of tools that cover every user quality requirement: T1+T3 (A1), T1+T4 (A2), T2+T3 (A3), and T2+T4 (A4).

TABLE IV: User requirements and alternatives.

Requirements		**Scale type**	**Tools**			
Quality measure	**Threshold**	**Higher/Lower best**	**T1**	**T2**	**T3**	**T4**
OLCC	80	Higher	85	70	/	/
IEE	3	Lower	5	2	/	/
AAF	0.75	Higher	/	/	0.8	0.74

The network related to this problem is that presented on Fig. 2, with the addition of one cluster related to all four identified alternatives. The part of the supermatrix related to the criteria is that of Table II, while the supermatrix of the complete problem is shown in Table V.

The values in the alternatives cluster of the supermatrix are obtained from the evaluation results; using the comparison algorithm presented in Section VI alternatives are compared according to each of the criteria from the user requirements.

For example, the comparison of alternatives according to *Ontology language component coverage* is shown in Table VI. A1 satisfies the requirement, while A3 does not and, hence, a value of 9 (extreme importance) is assigned when comparing A1 to A3. The overall importance of the alternatives according to the observed criteria is shown in the *Importance* column, and is entered in the corresponding column of the supermatrix.

TABLE VI: Alternatives comparisons with respect to OLCC.

OLCC	A1	A2	A3	A4	Importance
A1	1	1	9	9	0.45
A2	1	1	9	9	0.45
A3	1/9	1/9	1	1	0.05
A4	1/9	1/9	1	1	0.05

The weighted supermatrix is obtained by multiplying each element in the supermatrix with the importance of the cluster, after which a limit supermatrix is obtained. Every column in the limit supermatrix has the same values, which are shown in the *Limit supermatrix* column in Table V.

From the limit supermatrix, we can observe that the best alternative is A3 (with 0.074 score) and A1 (with 0.064) comes after. Both alternatives satisfy two requirements, and A3 is better with respect to *Import/Export errors (IEE)*, while A1 is better with respect to *Ontology language component coverage (OLCC)*; both are equal with respect to the *Average alignment F-measure*. However, since *Import/Export errors* is a characteristic more important than *Ontology language component coverage* ($0.346 > 0.184$) because of the dependencies in the network, A3 has a higher score.

Alternatives A4 and A2 satisfy only one requirement; therefore they are ranked as third and fourth respectively, where A4 is ranked better because it satisfies a characteristic that is more important (*Import/Export errors*).

IX. Conclusions And Future Work

This paper has presented a semantic technology recommendation framework, which is based on the Analytic Network Process. To apply the ANP to the semantic technology domain, we have defined the ANP network, which is based on a quality model for semantic technologies.

Having a quality model makes the definition of the network a straightforward task. Furthermore, the semantic technology quality model is a basis for the specification of quality requirements, and helps users to tailor the recommendation process to their needs.

This paper also describes an algorithm for the automatic comparison of alternatives in the ANP, and also in the AHP.

TABLE V: Supermatrix for the example.

	OLCC	**IEE**	**OPT**	**AAP**	**AAR**	**AAF**	**AAH**	**A1**	**A2**	**A3**	**A4**	**Limit supermatrix**
OLCC	0	1	0	0	0	0	0	1	1	1	1	0.184
IEE	1	0	1	0	0	0	0	1	1	1	1	0.346
OPT	0	1	0	0	0	0	0	0	0	0	0	0.12
AAP	0	0	0	0	0	0.462	0.462	0	0	0	0	0.017
AAR	0	0	0	0	0	0.462	0.462	0	0	0	0	0.17
AAF	0	0	0	0	0	0	0.076	1	1	1	1	0.064
AAH	0	0	0	0	0	0.076	0	0	0	0	0	0.002
A1	0.45	0.05	0	0	0	0.45	0	0	0	0	0	0.064
A2	0.45	0.05	0	0	0	0.05	0	0	0	0	0	0.05
A3	0.05	0.45	0	0	0	0.45	0	0	0	0	0	0.074
A4	0.05	0.45	0	0	0	0.05	0	0	0	0	0	0.060

This algorithm is domain independent and can be used in other scenarios in which evaluation results are available.

The comparison of alternatives in our framework is based on real evaluation results. New results and alternatives can be easily included in the framework, without the long process of expert-based comparisons required by the ANP.

Evaluation results are currently available only for individual tools. A future line of work is to specify new evaluations and obtain results for combinations of tools, i.e., for whole alternatives.

In the interval and ratio scales, the distance of the evaluation results form a threshold can be precisely calculated. Therefore, the alternatives comparison algorithm can be improved to take into account those distances.

The network and the supermatrix in our framework are made by experts in the semantic technology field. However, we plan to perform a validation with a broader group of experts and, in case of changes, to provide a way of easily updating the network and the supermatrix.

Future work also includes the implementation of the proposed framework in a web application. This will give users an easy access to a system that will help them in choosing the best semantic tools for solving the particular problems they face.

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.

REFERENCES

[1] B. Hecht, "Choose the right ERP software," *Datamation-Highlands Ranch*, vol. 43, no. 3, pp. 56–61, 1997.

[2] A. Guitouni and J. Martel, "Tentative guidelines to help choosing an appropriate MCDA method," *European Journal of Operational Research*, vol. 109, no. 2, pp. 501–521, 1998.

[3] J. Brans and P. Vincke, "A preference ranking organization method," *Management Science*, vol. 31, no. 6, pp. 647–656, 1985.

[4] C. Macharis, J. Springael, K. De Brucker, and A. Verbeke, "PROMETHEE and AHP: The design of operational synergies in multicriteria analysis. Strengthening PROMETHEE with ideas of AHP," *European Journal of Operational Research*, vol. 153, no. 2, pp. 307–317, 2004.

[5] T. Saaty, "Decision making with the Analytic Hierarchy Process," *International Journal of Services Sciences*, vol. 1, no. 1, pp. 83–98, 2008.

[6] W. Ho, "Integrated Analytic Hierarchy Process and its applications-a literature review," *European Journal of operational research*, vol. 186, no. 1, pp. 211–228, 2008.

[7] T. Saaty, "Fundamentals of the Analytic Network Process - Dependence and feedback in decision-making with a single network," *Journal of Systems science and Systems engineering*, vol. 13, no. 2, pp. 129–157, 2004.

[8] C. Herssens, I. Jureta, and S. Faulkner, "Dealing with Quality Tradeoffs during Service Selection," *Proceedings of the International Conference on Autonomic Computing (ICAC2008)*, pp. 77–86, 2008.

[9] R. Karim, C. Ding, and C.-H. Chi, "An Enhanced PROMETHEE Model for QoS-Based Web Service Selection," *2011 IEEE International Conference on Services Computing (SCC2011)*, pp. 536 –543, 2011.

[10] O. Vaidya and S. Kumar, "Analytic Hierarchy Process: An overview of applications," *European Journal of operational research*, vol. 169, no. 1, pp. 1–29, 2006.

[11] A. Jadhav and R. Sonar, "Evaluating and selecting software packages: A review," *Information and software technology*, vol. 51, no. 3, pp. 555–563, 2009.

[12] V. Dimitrova, "Application of the Analytic Network Process (ANP) in a framework of ERP systems implementation success," *The 4th International IEEE Conference on Intelligent Systems (IS2008)*, 2008.

[13] M. Godse, R. Sonar, and S. Mulik, "Web service selection based on Analytical Network Process approach," *Proceedings of the Asia-Pacific Services Computing Conference (APSCC2008)*, pp. 1103–1108, 2008.

[14] M. Mochól, "The methodology for finding suitable ontology matching approaches," Ph.D. dissertation, Freie Universität Berlin, Germany, 2009.

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

[16] ——, "ISO/IEC 25010-CD - JTC1/SC7, Software Engineering - Software product Quality Requirements and Evaluation (SQuaRE)," International Organization for Standardization, Tech. Rep., 2008.

[17] W. Ossadnik and O. Lange, "AHP-based evaluation of AHP-software," *European Journal of Operational Research*, vol. 118, no. 3, pp. 578–588, 1999.

[18] H. Jung and B. Choi, "Optimization models for quality and cost of modular software systems," *European Journal of Operational Research*, vol. 112, no. 3, pp. 613–619, 1999.

[19] E. Colombo and C. Francalanci, "Selecting CRM packages based on architectural, functional, and cost requirements: Empirical validation of a hierarchical ranking model," *Requirements engineering*, vol. 9, no. 3, pp. 186–203, 2004.

[20] F. Radulovic and R. García-Castro, "Extending Software Quality Models – A Sample in The Domain of Semantic Technologies," *Proceedings of the 23rd International Conference on Software Engineering and Knowledge Engineering (SEKE2011), Miami, USA*, pp. 25–30, 2011.

[21] F. Radulovic, "A software quality model for the evaluation of semantic technologies," Master's thesis, Facultad de Informática (Universidad Politécnica de Madrid), 2011.

[22] R. Keeney and H. Raiffa, *Decisions with multiple objectives: Preferences and value tradeoffs*. Cambridge University Press, 1993.

[23] S. Stevens, "On the theory of scales of measurement," *Science*, vol. 103, no. 2684, pp. 677–680, 1946.

P2P-based Publication and Location of Web Ontology for Knowledge Sharing in Virtual Communities

Huayou Si[1,2], Zhong Chen[1,2,*]
[1]Software Institute, School of Electronics Engineering and
Computer Science, Peking University
Beijing 100871, China;
{sihy, chen}@infosec.pku.edu.cn

Yong Deng[1,2]
[2]Key Laboratory of High Confidence Software
Technologies, Ministry of Education
Beijing 100871, China;
dengyong@pku.edu.cn

***Abstract*—In recent years, virtual communities, which focus on the purpose of knowledge sharing, are beginning to use web ontology to formally represent their sharable knowledge. In such a community, each member usually creates one or more local web ontologies for a given domain to be semantically queried by other members. But, it has become a pressing issue that, given a semantic query, how to efficiently locate the ontologies from which some solutions of the query can be reasoned out. To address this issue, we propose a structured P2P-based approach to publish sharable ontologies in each member's computer and automatically locate the useful ontologies to process a given SPARQL query. Therefore, given a SPARQL query, this approach can further send it to nodes, where at least one of ontologies useful to the query is located, to reason out solutions for the query respectively. Moreover, given a SPARQL query, if an ontology published can be reasoned out solutions, our approach is sure to locate the ontology and achieve the solutions. We also implemented this approach and conducted two experiments to evaluate its efficiency. The experimental results demonstrate that it is efficient.**

Keywords-Virtual Community; Web Ontology; Ontology Location; SPARQL Query; Peer-to-peer(P2P)

I. INTRODUCTION

Virtual community is a social network of individuals who interact through specific media, especially Internet, in order to pursue mutual interests or goals [1]. If a virtual community just focuses on the purpose of knowledge sharing, it is also called virtual knowledge community (VKC), which brings together geographically dispersed, like-minded people to form a network for knowledge exchange [2]. In recent years, with the wide application of Semantic Web, web ontology is applied to VKC to formally represent and automatically process sharable knowledge. In such a community [3, 4], each member usually creates one or more web ontologies to represent his/her own knowledge of a given domain. These ontologies possess a large quantity of knowledge to be shared and leveraged by each member in the community for his/her own purposes.

Due to the adoption of web ontology, knowledge sharing in such a community is largely based on structural semantic query. But, it has become a pressing issue that, given a semantic query in such a virtual community, how to efficiently locate the web ontologies, from which some solutions can be reasoned out. In recent years, the issue has been given a great deal of attention in practice as well as in research. Most of current approaches to deal with the issue are based on Client–Server (C/S) structure. In these approaches, all the web ontologies in a VKC are gathered and stored in s ome centralized knowledge servers. Community members can query and utilize knowledge under some sort of centralized control. These approaches have been considered inappropriate and ineffective to share knowledge [5, 6]. They are not suitable for the autonomous and dynamic characteristics of knowledge sharing [7, 8]. So, knowledge sharing in a decentralized network, especially supported by peer-to-peer (P2P) technology, is introduced. These approaches usually organize members' computers with sharable ontologies into an unstructured P2P network. Given a semantic query, these approaches try to route it to the nodes with useful knowledge to process it. However, unstructured P2P network limits their scalability and effectiveness.

To address the issue and overcome the limitations of current approaches, in this paper we propose a structured P2P-based approach to automatically publish and locate sharable web ontologies so as to facilitate processing semantic query. In our approach, community members' computers are organized into a structured P2P network. If a computer as a node has sharable web ontologies, it can directly publish them on P2P network. If a node receives a query of SPARQL (Simple Protocol and RDF Query Language) [9] from a requestor, it can efficiently locate the useful ontologies to process the query and send the query to nodes, where at least one of the useful ontologies is located, to reason solutions for the query respectively. Given a SP ARQL query, our approach makes sure that it can find out all the ontologies published, which can be reasoned out solutions for the query. Our approach provides user with a method to automatically share web ontologies in virtual community to process their SPARQL queries.

II. OVERVIEW OF OUR APPROACH

Peer-to-peer (P2P) systems usually consist of large numbers of autonomous nodes and allow the sharable resources of each node to be accessed by others. Especially structured P2P systems, such as Chord [10], they usually organize nodes in a systematic way and publish every sharable resource to a given node respectively. As a result, they can effectively find out a given resource and provide very good sc alability. Because of structured P2P with these strengths, we apply it to our approach for knowledge sharing. Based on structured P2P protocols, we can design two functions to publish and locate web ontologies on P2P network as follows:

1) **pubOnto(*idx, onto*)**, it is used to publish ontology *onto* based on its index *idx*(i.e., a property value of ontology *onto*). According to a given structured P2P protocol, it locates a given node *N* based on index *idx* so as to save the pair <*idx*, *onto*> on node *N*.

2) **lookupOnto(*idx*)**, it is used to get all the ontologies as a set, where these ontologies are published based on index *idx* (i.e., a property value of these ontologies) by some nodes.

B. Our Approach's Overview

In our approach, members' computers concerned (as nodes) constitute a structured P2P network. When a node with some sharable web ontologies joins, it publishes them as follows:

1) For each ontology *O*, based on each entity which appears in it, creates an index for it, which can describe what knowledge it possesses.

2) For each index *idx* of ontology *O*, using function pubOnto(*idx*, *onto*), publish ontology *O*'s IRI as *onto*.

Once a node *N* receives a SPARQL query, it processes it as follows:

1) Node *N* parses the query and creates the indices for the query. The indices can together describe what knowledge an ontology should possess if it is useful to process the query.

2) Based on the indices, node *N* locates the ontologies by using function lookupOnto(*idx*) with some strategies. the query's solutions can be reasoned out just from the ontologies.

3) For each located ontology *O*, node *N* sends the query to the node *n* where ontology *O* locates.

4) Node *n* reasons its ontology *O* to construct solutions for the query and returns solutions to node *N*.

5) Node *N* collects all the solutions returned and sends to requestor of the query.

The idea to create the indices for sharable ontologies and SPARQL query and the strategies to locate ontology for a query will be discussed in detail in the following section.

III. PUBLICATION AND LOCATION OF ONTOLOGY

In this section, we first introduce SPARQL basics and the basic idea to create indices of SPARQL query to locate useful ontologies. Then, we present our method to publish ontologies and our algorithm to locate ontologies for a query.

A. SPARQL Basics

SPARQL [9] is a query language for RDF diagrams. Since ontologies in compliance with OWL [11] or RDFS [12] are all based on RDF data model, they can be queried by SPARQL. Each SPARQL query has a graph pattern which consists of one or more pattern-clause. A graph pattern can be automatically converted into a semantically equivalent triple pattern. For example, the conversion can be shown from Figure 1 to Figure 2. The triple in a triple pattern is like RDF triple except that each of the subject, predicate and object may be a variable.

```
{   [ a dc:book ] dc:title "Semantic Web";
    dc:creator  ?y.
    ?y  a  dc:corporation.  }
```

Figure 1. A Graph Pattern with Initial Pattern-Clauses

```
{   _:b0 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> dc:book .
   _:b0 dc:title "Semantic Web" .
   _:b0 dc:creator ?y .
   ?y <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> dc:corporation }
```

Figure 2. Triple Pattern of Graph Pattern in Figure. 1

Graph pattern is used to convert into a triple pattern so as to match a sub-graph of the RDF diagram being queried when RDF terms from that sub-graph may be substituted for the variables in the graph pattern. The result, i.e., solution, is RDF graph equivalent to the sub-graph matched. Here, RDF terms mean recourses in RDF triples. In a SPARQL query, the graph pattern consists of the following five different categories: Basic, Group, Optional, Alternative, and Named Graphs Pattern.

Basic Graph Pattern contains a pattern-clause which must be matched. Similarly, Group Graph Pattern consists of a group of pattern-clauses which must be all matched too. So, given a SPARQL query with Basic or Group Graph Pattern, if a RDF graph can be queried out results, RDF terms appearing in the graph pattern are bound to appear in the RDF graph. Moreover, if the query's graph pattern is converted into a triple pattern, for each RDF term and its role (as one of subject, predicate and object) in a given triple in the triple pattern, there is at least one triple specified or implied in the RDF graph, which involves in the RDF term as the same role. So, we can take it as a clue to locate useful ontology for a query.

Optional Graph Patterns provide optional patterns to extend solutions to be reasoned out, while Named Graph Patterns designate the graphs that its patterns must be matched against. Thus, they can not provide useful clues to locate ontologies for a query with such graph patterns. Alternative Graph Pattern has two or more possible patterns to be tried. In fact, given a query *Q*, which graph pattern is an Alternative Graph Pattern, it can be broken into several sub-queries. Each sub-query takes one alternative part of query *O*'s graph pattern as its independent query pattern. Out of question, if one of sub-queries can be reasoned out solutions from an ontology, query *Q* surely can be reasoned out the same solutions from the ontology. Vice versa, if query *Q* can be reasoned out solutions from an ontology, one of its sub-queries surely can be reasoned out the same solutions. Thus, the ontologies located for each sub-query of a query can be united as the ontologies useful to the query.

In fact, a graph pattern of a query is usually formed by the five basic patterns in nested way. The outer-most one in a query is called query pattern. If a query's graph pattern consists of one or more Alternative Graph Patterns, we can repeatedly insert the pattern-clause (or other Graph Patterns) paralleled with an Alternative Graph Pattern *P* into *P*'s alternative parts until Alternative Graph Pattern is the query's outer-most graph pattern and each alternative part does not include any other Alternative Graph Pattern. So, we can get a semantically equivalent graph pattern, based on which we can break such query into several sub-queries without any Alternative Graph Pattern.

B. Indices Creation of SPARQL Query

Based on SPARQL basics as discussed above, we present a method to create indices for a given SPARQL query to locate useful ontology for it. The method just concerns the query without Alternative Graph Pattern because the query with such pattern can be broken into several sub-queries without such pattern to substitute for it. It consists of the following steps:

1) given a SPARQL query, extracts its graph pattern *GP*.

2) from graph pattern *GP*, removes Optional and Named Graph Patterns in it, as a graph pattern *G*.

3) converts graph pattern *G* into a triple pattern *TP*. For example, the conversion from Figure 1 to Figure 2.

4) from each triple *T* in triple pattern *TP*, first takes out each RDF term *tm*, which are not blank nodes, variables, or vocabularies of ontology language, such as OWL. Then, records RDF term *tm* and its role *Ro*(as one of subject, predicate, and object) in the triple *T* as a pair <*tm*, *Ro*>. Thus, given a query's triple pattern, we can obtain a pair set, called *tmRoPairs*.

RDF terms in pairs in *tmRoPairs* are also referred to as entities, which are classes, properties, or individuals in web ontology to express the notions in domain. Here, each pair in *tmRoPairs* is reviewed as an index of its SPARQL query. Based on SPARQL basics as discussed in subsection 3.1, we can draw a conclusion as follows:

Conclusion 1: If a web ontology is likely reasoned out results for a query, then for each pair *P* in the query's *tmRoPairs*, the ontology must specify or can be reasoned out a triple *T*, which contains the RDF term *tm* in pair *P*, and *tm*'s role in triple *T* is identical to the role in pair *P*.

For a pair *P* in a query's *tmRoPairs*, if a web ontology neither specify nor can be reasoned out such a triple *T*, it means that a triple in the query's triple pattern, from which pair *P* is constructed, must not be matched in the ontology. So, the ontology must not be reasoned out solutions for the query.

C. Web Ontology Publication

To locate web ontologies according to conclusion 1, we publish an ontology based on each entity and one of its possible roles which appear in a specified or implied RDF triple in the ontology. The process is listed as follows:

1) given sharable ontology *O*, extracts the entities as a set *enSet*, which appear in ontology *O*.

2) for each entity *E* in set *enSet*, reasons the ontology to determine whether there is at least one specified or implied triple where entity *E*'s role is subject (predicate, or object). If such a triple exists, puts *E* into *subSet* (*preSet*, or *objSet*).

3) takes each element *E* in set *subset* (*objSet*, *proSet*) and its corresponding role subject (predicate, object) as a pair <*E*, subject (or predicate, or object)>. Then, views the pair as an index *idx* of ontology *O*.

4) according to each *idx*, publishes the IRI *onto* of ontology *O* by using the function pubOnto(*idx*, *onto*) as discussed in subsection 2.1.

D. Web Ontology Location

Based on conclusion 1 and ontology publication method in subsection 3.3, we design algorithm *ontoLocating* in Figure 3 to locate ontologies useful to a given SPAQRL query. Its basic idea is that, given a query *Q*, we first discover ontologies based on each pair *P* in its *tmRoPairs* defined in subsection 3.2, and then intersect them as a set of useful ontologies to process query *Q*.

```
Algorithm ontoLocating
Input qry: a given SPARQL query.
Output ontos: a set of IRIs of ontologies useful to query qry.
Begin
sets integer i=0 ;
parses out set tmRoPairs from graph pattern of qry;
For each pair p in set tmRoPairs Do
  i=i+1;
  creates a index idx based on the entity and its role in p;
  retrieves a set ontoSet by using function lookupOnto (idx);
  If i=1 Then ontos = ontoSet Else
    ontos = ontoSet ∩ ontos
  End If
End Do
Return ontos;
End
```

Figure 3. Algorithm: *ontoLocating*

In this algorithm, to process a given query, the number that the algorithms access P2P network is the number of pairs in *tmRoPairs*.

IV. EVALUATION

To evaluate our approach, first we have implemented it. Then, design the following two experiments to evaluate its efficiency:

1) ***Experiment 1*** evaluates our approach's consumption of network resources when ontology is published.

2) ***Experiment 2*** evaluates our approach's consumption of network resources when query is processed.

A. Experiment Set Up

In our experiments, first we downloaded 16 web ontologies from TONES [13], an ontology repository, as experimental data, which are listed in Table I. Then, we design 15 SPARQL queries using the entities, which come from the ontologies in Table I.

In addition, we compare the consumption of network resources of our approach M1 with approach M2 and M3. For a same task, if an approach consumes fewer resources, it will be superior. Here, the consumption of network resources refers to the number of accesses to P2P network and quantity of values publishing on or retrieving from P2P network when a web ontology is published, or a query is processed. As far as approach M2 and M3 are concerned, they are all implemented based on RDFPeers [14], a structured P2P-based RDF repositories. RDFPeers stores each RDF triple at three places by applying hash functions to its subject, predicate, and object. Thus, all nodes know which nodes are responsible for the triples they are looking for if the triples exist in the network. So, we can implement approach M2 and M3 based on RDFPeers to process SPARQL query as follows:

*1) **Approach M2***: given a web ontology, it just publishes the triples specified explicitly in it. Given a SPARQL query *Q*, it retrieves the connected sub-graph of each entity appearing in *Q*'s graph pattern to create an ontology to process *Q*.

*2) **Approach M3***: given a web ontology, it publishes all the triples, which are specified or implied in the ontology. Given a SPARQL query, according to entities appearing in the query's graph pattern, this approach retrieves all the relevant triples to process the query.

Like our approach (M1) in this paper, given a SPARQL query, if an ontology published can be reasoned out solutions, approach M2 and M3 are sure to achieve the solutions. Here we do not discuss it in detail.

B. Results and Analysis

Experiment 1 is conducted to evaluate our approach's consumption of network resources when ontology is published. When we publish a web ontology by using approach M1, M2, and M3 respectively, we count the total numbers of the values to be inserted into P2P network according to each approach. In fact, using the three approaches, when a value is inserted into P2P network, the network must be accessed one time. Thus, the total numbers of the values to be published is the number of accesses to P2P network when ontology is published. The experimental results are recorded in column M1, M2, and M3 in the Table I respectively.

TABLE I. WEB ONTOLOGIES AND ITS NUMBERS OF THE VALUES PUBLISHED BY APPROACH M1, M2, AND M3

IRI of Ontologies from TONES		**M1**	**M2**	**M3**	**mulM2**	**mulM3**
O1.	file:/Users/seanb/Desktop/Cercedilla2005/hands-on/people.owl	291	950	5046	3.26	17.34
O2.	http://keg.cs.tsinghua.edu.cn/ontology/software	174	867	4569	4.98	26.26
O3.	http://www.mindswap.org/ontologies/family.owl	30	108	525	3.6	17.5
O4.	http://www.co-ode.org/ontologies/pizza/pizza.owl	341	3345	30867	9.81	90.52
O5.	http://www.owl-ontologies.com/Movie.owl	135	565	1605	4.19	11.89
O6.	http://www.bpiresearch.com/BPMO/2004/03/03/cdl/Countries	67	498	1920	7.43	28.66
O7.	http://www.semanticweb.org/ontologies/2007/9/AirSystem.owl	856	3398	65607	3.97	76.64
O8.	http://protege.stanford.edu/plugins/owl/owl-library/koala.owl	65	247	1209	3.8	18.6
O9.	http://www.loa-cnr.it/ontologies/DUL.owl	763	2689	38202	3.52	50.07
O10.	http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl	196	439	1560	2.24	7.959
O11.	http://www.mindswap.org/ontologies/debugging/university.owl	81	271	2823	3.35	34.85
O12.	http://www.co-ode.org/ amino-acid/2006/05/18/amino-acid.owl	143	2291	4503	16	31.49
O13.	http://www.mindswap.org/dav/commonsense/food/foodswap.owl	40	210	864	5.25	21.6
O14.	http://www.estrellaproject.org/lkif-core/role.owl	271	1078	4083	3.98	15.07
O15.	http://www.estrellaproject.org/lkif-core/lkif-top.owl	23	25	150	1.09	6.522
O16.	http://www.semanticweb.org/ontolgies/chemical	127	398	14313	3.13	112.7
The total quantity of published data		3603	17379	177846	4.82	49.36

TABLE II. NUMBERS OF ACCESSES TO P2P NETWORK FOR EACH QUERY PROCESS USING APPROACH M1, M2, AND M3

	Q1	**Q2**	**Q3**	**Q4**	**Q5**	**Q6**	**Q7**	**Q8**	**Q9**	**Q10**	**Q11**	**Q12**	**Q13**	**Q14**	**Q15**	**Total**
M1	5	6	6	5	5	5	5	6	5	5	4	5	5 5		4	76
M2	1383	1383	22	56	129	171	124	231	231	171	1383	98	98	129	129	5738
M3	5	6	6	5	5	5	5	6	5	5	4	5	5 5		4	76
mulM2	276.6	230.5	3.67	11.2	25.8	34.2	24.8	38.5	46.2	34.2	345.8	19.6	19.6	25.8	32.25	75.5
mulM3	1	1	1	1	1	1	1	1	1	1	1	1	1 1		1	1

TABLE III. NUMBERS OF VALUES RETRIEVED FROM P2P NETWORK FOR EACH QUERY PROCESS USING APPROACH M1, M2, AND M3

	Q1	**Q2**	**Q3**	**Q4**	**Q5**	**Q6**	**Q7**	**Q8**	**Q9**	**Q10**	**Q11**	**Q12**	**Q13**	**Q14**	**Q15**	**Total**
M1	5	6	6	5	5	8	5	6	5	6	4	5	5	5	4	80
M3	8547	8547	104	208	499	1025	564	946	946	1025	8547	429	429	499	499	32814
M2	2009	1726	403	2614	1708	1639	1692	1774	504	1735	1789	229	1821	2022	1926	23591
mulM2	1709.4	1424.5	17.33	41.6	99.8	128.13	112.8	157.67	189.2	170.83	2136.8	85.8	85.8	99.8	124.75	410.18
mulM3	401.8	287.67	67.17	522.8	341.6	204.88	338.4	295.67	100.8	289.17	447.25	45.8	364.2	404.4	481.5	294.89

In Table I, the column mulM2 (or mulM3) is the multiples from column M2 (or M3) to M1. Table I shows that, for approach M2, the multiples range from 1.09 to 16 and have a weighted average value 4.82; for approach M3, the multiples range from 6.522 to 112.7 and have a weighted average value 49.36. These imply that M1 can save large numbers of accesses to network, because it just publishes a very smaller quantity of values for a same ontology than M2 or M3. The reason is that, M1 publishes an ontology just based on entities and their roles appearing in the ontology, while M2 publishes each triple specified three times and M3 publishes all the specified and implied triples three times. An ontology usually contains large numbers of triples, especially, the triples implied.

Experiment 2 is conducted to evaluate our approach's consumption of network resources when a SPARQL query is processed. In this experiment, we conduct 3 tests. For the first one, we published all ontologies in table I by using approach M1. Then, we process the 15 SPARQL queries we constructed. When a query is processed, we record the number of accesses to P2P network and the quantities of the values retrieved from P2P network. For the second and third test, we do the same thing by using approach M2 and M3 respectively. Then, we record the results in Table II and Table III. In Table II, the row M1, M2, and M3 record the numbers of accesses to P2P network when a query is processed by approach M1, M2, and M3 respectively. In Table III, the row M1, M2, and M3 record the quantities of values retrieved from P2P network when a query is processed by approach M1, M2, and M3 respectively.

In Table II, the row mulM2 (or mulM3) is the multiples from row M2 (or M3) to M1. The row mulM2 shows that the multiples range from 3.67 to 345.8 and have a weighted average value 75.5. This means that approach M1 accesses network very less times than M2 when a query is processed. The reason is that, to obtain relevant connected sub-graphs for a query, M2 needs to access network iteratively based on each entity in the desired connected sub-graphs, while approach M1 just uses the entities appearing in the query to locate ontologies. The row mulM3 shows that all the multiples are 1. It means that the numbers in row M1 are identical to the corresponding numbers in row M3. The reason is that, for a query, approach M1 locates the appropriate ontologies from P2P network, while M3 retrieves relevant triples; but, they are all just based on the entities appearing in graph pattern.

In Table III, the row mulM2 (or mulM3) is also the multiples from row M2 (or M3) to M1. The row mulM2 shows that the multiples range from 17.33 to 2136.8 and have a weighted average value 410.18. The row mulM3 shows that the multiples range from 45.8 to 447.25 and have a weighted average value 294.89. That is to say, for a query, M1 only need retrieve significantly less data from P2P network than M2 or M3. It is because that, to obtain relevant connected sub-graphs for a query, M2 has to retrieve all triples related to each entity in desired connected sub-graphs. For approach M3, because it publishes all the triples specified or implied, usually large numbers of triples will be retrieved based on each entity.

V. RELATED WORK

Along with the development of P2P technique and the technical requirement of knowledge sharing in virtual community, in recent years, some P2P-based approaches for ontology publication and discovery have been proposed.

Chen et al. [7], propose a knowledge sharing approach, which organizes the nodes with sharable knowledge as an unstructured P2P network. For a query, the approach tries to send it to the nodes with related knowledge. Similar approaches are also presented in [4, 5, 8]. In these approaches, unstructured P2P network limits their scalability and effectiveness. Min et al. [14] present a scalable distributed RDF repository (named RDFPeers) that stores each triple at three places by applying globally known hash functions to its subject, predicate, and object. Thus, all nodes know which node is responsible for triples they are looking for. Queries are guaranteed to find out matched triples in the network if the triples exist. However, if RDF triples are directly published on P2P, it is difficult to support semantic retrieval.

VI. CONCLUSION AND FUTURE WORK

In this paper, we propose and implement a structured P2P-based approach to publish and locate sharable ontologies to process SPARQL queries in a virtual knowledge community, where web ontology is used to formally represent knowledge. Given a SPARQL query, our approach makes sure that it can find out all the sharable ontologies in community, which can be reasoned out solutions for the query. We also conducted two experiments to evaluate its efficiency. The experimental results demonstrated that our approach is efficient. In near future, we plan to continue our research work in the following aspects:

1) Conduct further study to investigate graph pattern of SPARQL query to find out other clues to more efficiently locate useful ontologies rather than the appearing entities and their roles. For example, structure of graph pattern.

2) Study the method to map words to existing ontological entities so as to facilitate requestors to construct their SPARQL queries automatically.

REFERENCE

[1] Wikipedia. Virtual Community. http://en.wikipedia.org/wiki/Virtual-community. Retrieved March 20, 2011.

[2] Wellman, B., Gulia, M.: Net-Surfers Don't Ride Alone: Virtual Communities as Communities. In: Wellman, B. (ed.): Networks in The Global Village. Boulder, CO: Westview Press (1999) 331-366

[3] P. Maret, M. Hammond, and J. Calmet. Virtual Knowledge Communities for Corporate knowledge Issues [C]. M.-P. Gleizes, A. Omicini, and F. Zambonelli (Eds.): ESAW 2004, LNAI 3451, pp. 33–44.

[4] Melanie Gnasa, Sascha Alda, Jasmin Grigull et al. Cremers. Towards Virtual Knowledge Communities in Peer-to-Peer Networks [C]. J. Callan et al. (Eds.): SIGIR 2003 Ws Distributed IR, LNCS 2924, pp. 143–155

[5] Zhen, L.; Jiang, Z. & Song, H. Distributed recommender for peer-to-peer knowledge sharing Information Sciences, 2010, 180, 3546 – 3561

[6] J.S.H. Kwok, S. Gao, Knowledge sharing community in P2P network: a study of motivational perspective, Journal of Knowledge Management 8(1) (2004) 94–102.

[7] Chen-Ya Wang, Hsin-Yi Yang, Seng-cho T. Chou. Using peer-to-peer technology for knowledge sharing in communities of practices, Decision Support Systems 45 (2008) 528–540.

[8] L.C. Jain, N.T. Nguyen, Knowledge Processing and Decision Making in Agent-based Systems [M], Springer-Verlag, 2009.

[9] W3C. SPARQL Query Language for RDF. http://www.w3.org/TR/ rdf-sparql-query/. Retrieved February 5, 2012.

[10] I. Stoica, R. Morris, D. Liben-Nowell, D.R. Karger, M.F. Kaashoek, F. Dabek, H. Balakrishnan, Chord: a scalable peer-to-peer lookup protocol for internet applications, IEEE/ACM Transactions on Networking (TON) 11(1) (2003) 17-32.

[11] OWL. http://www.w3.org/TR/owl2-overview/. February 9, 2012.

[12] RDF Schema. http://www.w3.org/TR/rdf-schema/. February 8, 2012.

[13] TONES. http://owl.cs.manchester.ac.uk/repository. February 17, 2012.

[14] Min Cai, Martin Frank. RDFPeers: a scalable distributed RDF repository based on a structured peer-to-peer network. Proceedings of the 13th international conference on Word Wide Web (WWW'04). ACM Press. 2004. pp.650-657.

Empirical Validation of Variability-based Complexity Metrics for 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. The product line architecture is one of the most important product line core assets as it is the abstraction of the products that can be generated, and it represents similarities and variabilities of a product line. Its quality attributes analysis and evaluation can serve as a basis for analyzing the managerial and economical values of a product line. This analysis can be quantitatively supported by metrics. Thus, we proposed metrics for the product line architecture complexity quality attribute. This paper is concerned with the empirical validation of such metrics. As a result of the experimental work we can conclude that the metrics are relevant indicators of complexity of product line architecture by presenting a correlation analysis.**

***Keywords*-Complexity, Correlation Analysis, Emprical Validation, Metrics, Software Product Line Architecture.**

I. Introduction

In the last decades effective methodologies to evaluate software architectures, such as ATAM (Architecture Tradeoff Analysis Method) and SAAM (Software Architecture Analysis Method), were proposed and consolidated by both industrial and academic segments [7]. Such a consolidation is corroborated by the analysis of the number of published research papers and technical reports providing important examples of how to carry out a software architecture evaluation based on quality attributes. Thus, these methodologies are essential for evaluating single-product architectures.

In recent years, the software product line (PL) engineering [10] has emerged as a promising reusability approach, which brings out some important benefits, such as increases the reusability of its core assets, while decreases the time to market. One of the most important assets of a PL is its architecture (PLA). The PLA plays a central role at the development of products from a PL as it is the abstraction of the products that can be generated, and it represents similarities and variabilities of a product line.

The evaluation of a PLA must be supported by a set of metrics [8]. Such metrics must both evidence the quality of PL and serve as a basis to analyze the managerial and economical value of a PL [2]. The PLA must explicit the common (similarities) and variable (variabilities) aspects of a PL. The variability impact analysis on the PL development can determine the aggregated value of a PL for an organization. Metrics for a PLA are applied to a set of assets from which variants can be generated rather than one specific product. Thus, it is necessary to define specific PLA metrics to provide effective indicators with regard to the overall PL development and evolution.

We proposed six metrics for PLA complexity. These metrics were defined to provide an indicator of how complex is a PLA by measuring its derived PL products. Complexity is measured based on McCabe's Cyclomatic Complexity (CC) [11] which measures the number of different paths in a source code. Basically, class complexity is calculated by the Weighted Metrics per Class (WMC) metric [6], which is a composition of the CC metric. Thus, component complexity is the sum of the complexity of all classes that form a component.

Variabilities are related to PLA class and/or components. Each variability is related to variation points and/or variants that realize it. A variation point or variant might be a PLA class or component. The complexity of a variability can be calculated based on the complexity of each variation point or variant.

Both theoretical and empirical validations [5] are necessary to validate a set of metrics. Theoretical validation is concerned with demonstrating that a metric is measuring the concept it is purporting to measure. The first requirement for theoretical validation is that either the analyst has an intuitive understanding of the concept that is being measured and/or that the software engineering community has a consensual intuitive understanding of the concept. Theoretical validation of the complexity metrics have been done in [12].

This paper is concerned with the empirical validation of the proposed metrics for PLA complexity quality attribute. The validation aims at correlating the metrics with subject's complexity rating, respectively, when generating PLA configurations. A PLA configuration represents a derived PL product with variabilities resolved.

This paper is organized as follows: Section II defines the complexity metrics to be validated and illustrates how to collect them; Section III presents how the experimental study was planned and carried out to validate the complexity metrics; Section IV discusses the results obtained in this study; and Section V provides the conclusions and directions for future work.

II. Complexity Metrics for Software Product Line Architectures

The complexity understanding is essential from the PL adoption point as a PL manager is able to analyze the complex of the potential PL products to be produced.

Organizations which have a developed PL core asset for a certain domain can analyze the complexity of the distinct configurations and the PL evolution. Therefore, a PL manager may choose from a set of feasible configurations which are the most interesting to be produced.

The complexity metrics for PLA were composed based on the Cyclomatic Complex (CC) [11] and Weighted Methods per Class (WMC) [6]. The CC metric measures the quantity of decision logic represented by the number of paths to be tested in a source code. The WMC metric is the sum of the CC metric for each concrete method in an object-oriented class. Abstract methods have WMC value 0.0. Each metric measures the complexity of class, interface and component based on one of the following PL variability concepts:

- **Variability**, according to Bosch [4], is "the ability of a software or artifact to be changed, customized or configured for use in a particular context." Although a variability can take place at different levels of abstraction and artifacts, the complexity metrics in this paper address only class and component UML artifacts that result from PL activities [14] and represents the PLA;
- **Variation Point** is the resolution of variabilities in generic artifacts of a PL. According to Jacobson et al. [9], "a variation point identifies one or more locations at which the variation will occur." Thus, a variation point may take place at generic artifacts and at different levels of abstraction. Basically, a variation point answers the question: What varies in a PL? [16]; and
- **Variant** represents the possible elements through which a variation point may be resolved. It may also represent a way to directly resolve a variability. Basically, a variant answers the question: How does a variability or a variation point vary in a PL? [16].

The complexity metrics taken into consideration in this paper are as follows:

***CompInterface*:** measures the complexity of an interface. It always has value 0.0 as an interface has no concrete methods to calculate the WMC metric. This metric is represented by the following formula (1):

$$\textbf{CompInterface(Cls)} = WMC(Itf) = 0.0, \tag{1}$$

where:

- *n = # of concrete methods (Mtd) of an interface (Itf)*

***CompClass*:** measures the complexity of a class. It is the WMC metric value for a class. This metric is represented by the following formula:

$$\textbf{CompClass(Cls)} = WMC(Cls) = \sum_{i=1}^{n} WMC(Mtd_i), \tag{2}$$

where:

- *n = # of concrete methods (Mtd) of a class Cls*

***CompVarPointClass*:** measures the complexity of a variation point. It is the value of the metric *CompClass* (Equation 2), for a class which is a variation point, plus the sum of the *CompClass* (Equation 2) value for each associated variant class. This metric is represented by the following formula:

$$\textbf{CompVarPointClass(Cls)} = CompClass(Cls) + \sum_{i=1}^{n} CompClass(Ass_i), \tag{3}$$

where:

- *n = # of (inclusive + exclusive + optional + mandatory) variant classes and interfaces associated (Ass)*

***CompVariabilityClass*:** measures the complexity of a variability. It is the sum of the metric *CompVarPointClass* (Equation 3), for each variation point. This metric is represented by the following formula:

$$\textbf{CompVariabilityClass(Vbt)} = \sum_{i=1}^{nVP} CompVarPointClass(Cls_i), \tag{4}$$

where:

- *nVP = # of class and interface (Cls) variation points*

***CompVarComponent*:** measures the complexity of a variable PLA component. It is the sum of the metric *CompVariabilityClass* (Equation 4), for each variability in a component. This metric is represented by the following formula:

$$\textbf{CompVarComponent(Cpt)} = \sum_{i=1}^{nVar} CompVariabilityClass(Var_i), \tag{5}$$

where:

- *nVar = # of variabilities (Var) in a component (Cpt)*

***CompPLA*:** measures the complexity of a PLA. It is the sum of the *CompVarComponent* (Equation 5) for each component of a PLA. This metric is represented by the following formula:

$$\textbf{CompPLA(PLA)} = \sum_{i=1}^{nCpt} CompVarComponent(Cpt_i), \tag{6}$$

where:

- *nCpt = # of PLA variable components*
- *Cpt_i is the o i_{th} component of a PLA*

III. Experimental Study

In this section we describe the experiment we have carried out to empirically validate the proposed metrics as indicators of PLA complexity. We have followed the suggestions provided by Wohlin et al. [19] and Perry et al. [15] on how to perform controlled experiments with minor changes.

A. Definition

Based on the Goal-Question-Metric (GQM) template [1], the goal of the experiment is presented as follows:

Analyze *collected metrics from UML models and source code*

For the purpose of *validating*

With respect to *the capability to be used as PLA complexity indicators*

From the point of view of *software product line architects*

In the context of *graduate students of the Software Engineering area at the University of Waterloo (UWaterloo), University of São Paulo (ICMC-USP), and State University of Maringá (UEM).*

B. Planning

1) Context Selection: the experiment was carried out in an academic environment.

2) Selection of Subjects: a group of Software Engineering graduate students from ICMC-USP, UEM, and UWaterloo. They have experience in the design of product lines and variabilities using UML.

3) Variable Selection: the independent variables were the class and component complexity of a PLA. The dependent variables were the complexity of each product generated from the PLA.

4) Instrumentation: the objects were: a document describing the Arcade Game Maker (AGM) PL [17]; AGM UML class and component models, a traceability model from classes to components; and a resolution model containing the variabilities to be resolved at class level. The independent variables were measured by the proposed metrics. The dependent variables were measured according to the subjects ratings of complexity.

5) Hypothesis Formulation: the following hypothesis were tested in this study:

- **Null Hypothesis (H_0):** There is no significant correlation between the PLA complexity metric (*CompPLA*) and the subject's complexity rating for each PLA configuration; and
- **Alternative Hypothesis (H_1):** There is a significant correlation between the PLA complexity metric (*CompPLA*) and the subject's complexity rating for each PLA configuration.

6) Experiment Design: all the tasks had to be solved by each of the subjects.

C. Operation

1) Preparation: when the experiment was carried out, all of the subjects had graduated in the Software Engineering area, in which they have learned how to design at least object-oriented (OO) class diagrams using UML. In addition, all of the subjects had experience in applying PL and variability concepts to OO systems designed using UML. The material prepared to the subjects consisted of:

- the class diagram representing the core asset of the AGM PL;
- the AGM component diagram, representing its logical architecture;
- an AGM traceability model from classes to components;
- the description of the AGM PL;
- the SMartyProfile [13], which is a UML metamodel, thus the subjects can understand how the variabilities are represented in class and component diagrams;
- a variability resolution model, which the subjects could resolve the variabilities to generate one AGM configuration; and
- a test (questionnaire) describing complexity concepts, which the subjects had to rate the associated complexity of each generated AGM configuration based on linguistic labels (Table I).

Table I
Linguistic Labels for Complexity Subjects Rating.

Extremely Low	Low	Neither Low nor High	High	Extremely High

We selected five linguistic labels, based on Bonissone [3], as we considered they are significant to cover the complexity category of our variables and bring out balance to obtain better results.

2) Execution: the subjects were given the material described in *Preparation* (Section III-C1). It was required to each subject to generate one AGM configuration. It was done by following instructions on how to resolve the AGM variability resolution model, and how to rate the complexity associated to the configurations generated from the subjects view point. All the tasks were performed by each subject alone, with no time limit to solve them and neither sequentially nor simultaneously. As the metric CompPLA is a composition of the remaining complexity metrics of this paper, we only take CompPLA into consideration for the validation purpose. In addition, the CompPLA value of each configuration was divided by the CompPLA value of the overall AGM PLA, thus resulting in a value ranging from 0.0 to 1.0.

3) Data Validation: the tasks realized by the subjects were collected. We consider the subjects subjective evaluation reliable.

D. Analysis and Interpretation

We summarized the collected data by calculating the metrics CompPLA for the thirty AGM configurations generated by the subjects, as well as verifying the complexity rating of such configurations. Table II presents the observed values for the CompPLA metric from the generated AGM configurations.

Table II
OBSERVED VALUES FOR THE COMPPLA METRIC FROM THE GENERATED CONFIGURATIONS.

Configuration #	CompPLA	Configuration #	CompPLA
1	0.51	16	0.98
2	0.56	17	0.77
3	0.51	18	0.82
4	0.83	19	0.52
5	0.91	20	0.82
6	0.50	21	0.49
7	0.47	22	1.00
8	0.53	23	0.52
9	0.67	24	0.42
10	0.90	25	0.62
11	0.53	26	0.47
12	0.97	27	0.53
13	0.48	28	0.70
14	0.69	29	0.40
15	0.74	30	0.78

1) Descriptive Statistics: Figure 1 presents the CompPLA observed values (Table II) plotted in a boxplot.

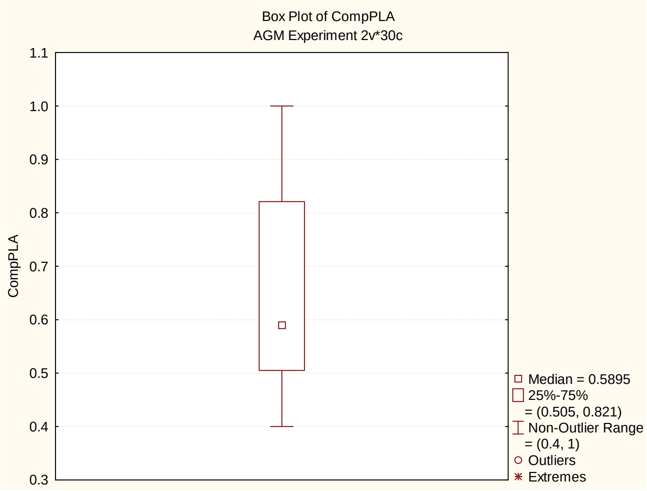


Figure 1. Boxplot for the CompPLA Observed Values.

2) Normality Tests: We can clearly observe that the CompPLA values distribution (Figure 1) is non-normal. In spite of it, Shapiro-Wilk and Kolmogorov-Smirnov normality tests were conducted to make sure of it.

The following hypothesis were proposed for both normality tests with regard to the CompPLA metric:

- **Null Hypothesis (H_0):** the CompPLA observed values distribution is normal, i.e., the significance value (*p*) is greater than 0.05 ($p > 0.05$); and
- **Alternative Hypothesis (H_1):** the CompPLA observed values distribution is non-normal, i.e., the significance value (*p*) is less or equal to 0.05 ($p \leq 0.05$).

Taking into account a sample size (N) of 30, with mean (μ) 0.6545, standard deviation (σ) 0.1842, and median (x̃) 0.5895, the *CompPLA* metric obtained a significance value:

- $p < 0.01$ ($0.01 < 0.05$) for the *Kolmogorov-Smirnov* test;
- $p = 0.0118$ ($0.0118 < 0.05$) for the *Shapiro-Wilk* test.

Thus, there is evidence, for both normality tests, that the null hypothesis (H_0) must be rejected at a significance level of 5%. Then, we cannot consider the CompPLA observed values distribution normal and, consequently, a non-parametric statistic method must be used to analyze the data.

3) Spearman's Rank Correlation: as CompPLA distribution is non-normal, we applied the non-parametric Spearman's Correlation (ρ) [18] to support the interpretation of the data. This method allows to establish whether there is a correlation between two sets of data. Equation (7) presents the Spearman's ρ formula:

$$\left.\begin{array}{l}\rho = 1 - \frac{6}{n(n^2-1)} \sum_{i=1}^{n} d_i^2, \text{ where } n \text{ is the sample} \\ \text{size } (N)\end{array}\right\} \quad (7)$$

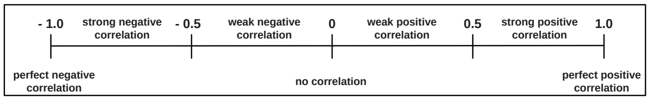

- 1.0
strong negative correlation
- 0.5
weak negative correlation
0
weak positive correlation
0.5
strong positive correlation
1.0
perfect negative correlation
no correlation
perfect positive correlation

Figure 2. Spearman's Rank Correlation Scale.

We performed the following correlation (Corr.1): **CompPLA and the subjects complexity rating**, which shows that the understanding of complexity by the subjects corroborates to the CompPLA metric, establishing how to measure complexity in PLA.

Table III presents the Spearman's ranking correlation for Corr.1. The Spearman ρ coefficient (Equation 7) for Corr.1 is calculated as follows:

$$\rho(_{Corr.1}) = 1 - \frac{6}{30(30^2-30)} \times 293.5 = 1 - \frac{6}{26970} \times 293.5 = 1 - 0.07 = \mathbf{0.93}$$

Thus, according to Figure 2, there is a strong positive correlation ($\rho(_{Corr.1})$ = 0.93) between the metric CompPLA and the subjects complexity rating.

Based on the proposed correlation, we have evidence to reject the null hypothesis H_0 of the study, and accept the alternative hypothesis H_1 (Section III-B5), which states that complexity metrics are significantly correlated to complexity of PLA.

Table III
SPEARMAN'S CORRELATION FOR CORR.1: COMPPLA AND SUBJECTS COMPLEXITY RATES.

Config. #	*CompPLA*	r_a	Subject's Complexity Rating	r_b	d $\| r_a - r_b \|$	d^2
1	0.51	22.5	Neither Low nor High	22.5	0	0
2	0.56	16	Neither Low nor High	22.5	6.5	42.25
2	0.51	22.5	Neither Low nor High	22.5	0	0
4	0.83	6	Extremely High	4.5	1.5	2.25
5	0.91	4	Extremely High	4.5	0.5	0.25
6	0.50	24	Neither Low nor High	22.5	1.5	2.25
7	0.47	27.5	Neither Low nor High	22.5	5	25
8	0.53	18	Neither Low nor High	22.5	4.5	20.25
9	0.67	14	High	12	2	4
10	0.90	5	Extremely High	4.5	0.5	0.25
11	0.53	18	Neither Low nor High	22.5	4.5	20.25
12	0.97	3	Extremely High	4.5	1.5	2.25
13	0.48	26	Neither Low nor High	22.5	3.5	12.25
14	0.69	13	High	12	1	1
15	0.74	11	High	12	1	1
16	0.98	2	Extremely High	4.5	2.5	6.25
17	0.77	10	High	12	2	4
18	0.82	7.5	Extremely High	4.5	3	9
19	0.52	20.5	Neither Low nor High	22.5	2	4
20	0.82	7.5	Extremely High	4.5	3	9
21	0.49	25	Neither Low nor High	22.5	2.5	6.25
22	1.00	1	Extremely High	4.5	3.5	12.25
23	0.52	20.5	Neither Low nor High	22.5	2	4
24	0.42	29	Neither Low nor High	22.5	6.5	42.25
25	0.62	15	High	12	3	9
26	0.47	27.5	Neither Low nor High	22.5	5	25
27	0.53	18	Neither Low nor High	22.5	4.5	20.25
28	0.70	12	High	12	0	0
29	0.40	30	Low	30	0	0
30	0.78	9	High	12	3	9

E. Validity Evaluation

In this section we discuss the empirical study's threats to validity and how we tried to minimize them.

1) Threats to Conclusion Validity: the only issue that we take into account as a risk to affect the statistical validity is the sample size (N=30), which can be increased during prospective replications of this study in order to reach normality of the observed values.

2) Threats to Construct Validity: our dependent variable is complexity. We proposed subjective metrics for it, as linguistic labels, collected based on the subjects rating. As the subjects have experience in modeling OO systems using at least class diagrams, we take their ratings as significant. The construct validity of the metrics used for the independent variables is guaranteed by some insights carried out on a previous study of metrics for PLA [12].

3) Threats to Internal Validity: we dealt with the following issues:

- **Differences among subjects.** As we dealt with a small sample, variations in the subject skills were reduced by applying the within-subject task design. Thus, subjects experiences had approximately the same degree with regard to UML modeling, and PL and variabilities basic concepts.
- **Accuracy of subject responses.** Complexity was rated by each subject. As they have medium experience in UML modeling, and PL and variabilities concepts, we considered their responses valid.
- **Fatigue effects.** On average the experiment lasted for 69 minutes, thus fatigue was considered not very relevant. Also, the variability resolution model contributed to reduce such effects.
- **Measuring PLA and Configurations.** As PLA can be analyzed based on its products (configurations), measuring derived configurations provide a means to analyze PLA quality attributes by allowing the performing of trade-off analysis to prioritize such attributes. Thus, we consider valid the application of the metrics to PLA configurations to rate the overall PLA complexity.
- **Other important factors.** Influence among subjects could not really be controlled. Subjects did the experiment under supervision of a human observer. We believe that this issue did not affect the study validity.

4) Threats to External Validity: Based on the greater the external validity, the more the results of an empirical study can be generalized to actual software engineering practice, two threats of validity have been identified, which are:

- **Instrumentation.** We tried to use representative class and component diagrams of real cases. However, the PL used in the experiment is non-commercial, and some assumptions can be made on this issue. Thus, more empirical studies taking a "real PL" from software organizations must be done.
- **Subjects.** Obtaining well-qualified subjects was difficult, thus we used advanced students from the Software Engineering academia. More experiments with practitioners and professionals must be carried out allowing us to generalize the study results.

IV. DISCUSSION OF RESULTS

Obtained results of the study lead us to conclude that the metric CompPLA is a relevant indicator of PLA complexity based on its correlation to subject's rating.

Several more experiments must be carried out, as well as more PLA configurations must be both derived and incorporated to enhance the conclusions. In addition, we need to apply our metrics to a commercial PL in order to reduce external threats to the study validity and for gathering real evidence that these metrics can be used as complexity indicators.

V. Conclusion

Current literature claims the need of metrics to allow PL architects empirically analyze the potential of a PLA, as well as PL managers analyze the aggregated managerial and economical values of a PL throughout its products.

Performing empirical validation of metrics is essential to demonstrate their practical usefulness. The proposed metrics for the complexity (CompPLA) PLA quality attribute were empirically validated based on their application to a set of 30 products generated by experiment subjects from the *Arcade Game Maker* (AGM) PL. The observed metric values were submitted to normality tests which proved their non-normality. Then, Spearman's rank correlation was used to demonstrate the metrics correlations, which is: CompPLA has a strong positive correlation with the subjects complexity rating.

Although we have used a non-commercial PL to conduct our experiments, we had evidences that our proposed metrics can be used as relevant indicators of complexity of a PLA based on its derived products.

We are currently proposing changes on various issues to improve our experiments with metrics, which are:

- increase the derived configurations sample size, which is important to stay closer to real projects and to generalize the results;
- conduct experiments in a more controlled environment;
- deal with real data from commercial PL obtained from industrial environments; and
- recruit more subjects from the Software Engineering area, both from academic and industrial environments.

References

[1] V. Basili and H. Rombach, "The TAME Project: Towards Improvement-Oriented Software Environments," *IEEE Transactions on Software Engineering*, vol. 14, no. 6, pp. 758–773, 1988.

[2] G. Böckle, P. Clements, J. D. McGregor, D. Muthig, and K. Schmid, "Calculating ROI for Software Product Lines," *IEEE Software*, vol. 21, no. 3, pp. 23–31, May 2004.

[3] P. P. Bonissone, "A Fuzzy Sets Based Linguistic Approach: Theory and Applications," in *Proceedings of Conference on Winter Simulation*. Piscataway, NJ, USA: IEEE Press, 1980, pp. 99–111.

[4] J. Bosch, "Preface," in *Proceedings of the 2nd Groningen Workshop on Software Variability Management: Software Product Families and Populations*. Groningen, The Netherlands: University of Groningen, 2004, pp. 1–2.

[5] L. Briand, K. E. Emam, S. Morasca, K. El, and E. S. Morasca, "Theoretical and Empirical Validation of Software Product Measures," International Software Engineering Research Network, ISERN-95-03, 1995.

[6] S. R. Chidamber and C. F. Kemerer, "A Metrics Suite for Object Oriented Design," *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476–493, 1994.

[7] P. Clements, R. Kazman, and M. Klein, *Evaluating Software Architectures: Methods and Case Studies*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.

[8] E. Dincel, N. Medvidovic, and A. v. d. Hoek, "Measuring Product Line Architectures," in *Proceedings of the International Workshop on Product Family Engineering*. London, UK: Springer-Verlag, October 2001, pp. 346–352.

[9] I. Jacobson, M. L. Griss, and P. Jonsson, *Software Reuse: Architecture, Process, and Organization for Business Success*. Boston, MA, USA: Addison-Wesley Professional, 1997.

[10] 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.

[11] T. J. McCabe, "A Complexity Measure," *IEEE Transactions on Software Engineering*, vol. 2, no. 4, pp. 308–320, 1976.

[12] 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.

[13] ——, "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.

[14] E. A. Oliveira Junior, I. M. S. Gimenes, E. H. M. Huzita, and J. C. Maldonado, "A Variability Management Process for Software Product Lines," in *Proceedings of the Conference of the Centre for Advanced Studies on Collaborative Research*. Toronto, ON, Canada: IBM Press, 2005, pp. 225–241.

[15] D. E. Perry, A. A. Porter, and L. G. Votta, "Empirical Studies of Software Engineering: a Roadmap," in *Proceedings of the International Conference on Software Engineering*. New York, NY, USA: ACM, 2000, pp. 345–355.

[16] K. Pohl, G. Böckle, and F. J. v. d. Linden, *Software Product Line Engineering: Foundations, Principles, and Techniques*. Secaucus, NJ, USA: Springer-Verlag, 2005.

[17] SEI, "Arcade Game Maker Pedagogical Product Line," 2010. [Online]. Available: http://www.sei.cmu.edu/productlines/ppl

[18] C. Spearman, "The Proof and Measurement of Association Between Two Things," *American Journal of Psychology*, vol. 15, no. 1, pp. 72–101, 1904.

[19] C. Wohlin, P. Runeson, M. Hust, M. C. Ohlsson, B. Regnell, and A. Wesslun, *Experimentation in Software Engineering: an Introduction*. Norwell, MA, USA: Kluwer Academic Publishers, 2000.

A Mapping Study on Software Product Lines Testing Tools

Crescencio Rodrigues Lima Neto[1,3], Paulo Anselmo Mota Silveira Neto[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, srlm}@cin.ufpe.br, pamsn@rise.com.br, esa@dcc.ufba.br

Abstract—**The benefits of using a software testing tool in order to achieve significant reductions in cost and time to market, and, at the same time, increasing the quality has encouraged the adoption of testing tools both in single systems and product lines. In this context, this study focuses on the following goals: analyze how the available tools are supporting the Software Product Lines (SPL) Testing Process, investigate the state-of-the-art on single system and SPL testing tools, synthesize available evidence, and identify gaps among the tools, available in the literature. A mapping study was undertaken to analyze important aspects that should be considered when adopting testing tools. A set of four research questions were defined in which 33 studies, dated from 1999 to 2011, were evaluated. From the total of 33 studies considered, 24 of them described single system testing tools and the other 9 described SPL testing tools. However, there is insufficient information about publications describing tools used in the industry. There is no tool suitable to all testing levels of a SPL, researchers need to consider the feasibility of adapting existing tools or constructing new tools.**

Keywords—**Software Testing; Software Product Lines; Software Reuse; Testing Tools; Systematic Literature Reviews**

I. Introduction

In order to achieve the ability to build individualized products, to meet individual customer needs, companies need high investments, which consequently leads to higher prices for final products. In this context, the increasing adoption of Software Product Lines (SPL) practices in industry has decreased implementation costs, reduced time-to-market and improved quality of products. Although the benefits are promising, SPL advantages demand mature software engineering practices, planned and managed reuse [1].

Testing is an essential part of software development to address product quality, since it examines the core assets, the product-specific software, the interaction between them, and finally the completed products [2].

Testing tools are important to increase the quality of testing and at the same time reduce the effort to perform them. However, existing testing tools (see Section IV) are mostly unable to support SPL Testing [3].

This study has been structured combining ideas from [4] with good practices defined in the guidelines proposed by [5], such as the protocol definition. Therefore, we applied a process for a mapping study (MS), including best practices for conducting systematic reviews (SR), making the best use of both methods.

The objective of this study is to investigate ***how do the available tools support the Software Product Lines Testing process?*** This study aims to map out existing testing tools, to synthesize evidence to suggest important implications for practice, as well as to identify research trends, open issues, and areas for improvement. We derived four research questions:

- **RQ1 -** *In which context the proposed tools were proposed and evaluated?* It aims at identifying where the tools were developed (industry or academia) and how they have been validated. Through these descriptions, it is possible to map the current adoption of the tools.
- **RQ2 -** *Is it possible to use the single system testing tools to test software product lines?* It aims to analyze the possibility of using the single system testing tools to test SPL instead of creating tools from scratch.
- **RQ3 -** *Which testing levels are supported by existing tools?* It aims to classify the tools according to the testing levels in order to identify what the tools can offer. It can provide information regarding to further research in a specific area or testing level.
- **RQ4 -** *How are testing tools evolving?* It aims to analyze how the existing testing tools are evolving, regarding to the different testing levels.

Furthermore, we believe that this paper can provide insights for new research in the software testing tools area, serving as a baseline, by analyzing the available tools and presenting the existing gaps.

The remainder of this paper is structured as follows: Section II presents the related work. Section III describes the research method used. Section IV presents the analysis of the results. Section V presents a discussion on the results and the main findings of the study. Section VI presents the threats to validity, and finally, Section VII discusses the main conclusions.

II. Related Work

Several studies [6], [3], [7], [8], emphasize the need for testing tools to support the Testing Process, however, few ones focus on SPL.

Tevanlinna et al. [3] performed a survey in which evaluated the state-of-the-art in SPL Testing highlighting the importance of testing tools for supporting the SPL process. The study investigated the need for product lines specific testing tools that should help to manage the reusable testing assets, automate the test execution activity, and the analysis of their results. Moreover, the authors explored specifically regression testing, testing of partial programs, frameworks and component testing.

Edwin [7] performed a systematic literature review, which aims to find out the primary studies relating to Software Product Lines Testing Tools. It investigated the tools that can be used to support testing in the software product lines context to ensure high quality in SPL and its derived products.

Lamancha et al. [9] presented a systematic literature review, which deals with testing in software product lines. The authors analyzed the existing approaches to SPL testing, discussing the significant issues related to this area. The work discussed also which papers described software product lines testing tools

Neto et al. [10] and Engströn et al. [11] presented a systematic mapping study performed in order to map out the SPL Testing field, through synthesizing evidence to suggest important implications for practice, as well as identifying research trends, open issues, and areas for improvement.

We elaborated a mapping study that investigates single system testing tools and SPL testing tools. We analyzed all the test levels (unit, integration, system and acceptance) of the SPL Testing Process including regression testing. Finally, we identified the possibility to adapt single systems testing tools to test SPL.

III. Research Methodology

According to [12], a MS provides a systematic and objective procedure for identifying the nature and extent of the empirical study data that is available to answer a particular research question. While a SR is a mean of identifying, evaluating and interpreting all available research relevant to a particular question [5].

Figure 1 presents the mapping study process adapted from [4]. The process was divided into three main phases: Research Directives which establishes the protocol and research questions, Data Collection which comprises the execution of the mapping study and the Results responsible for reporting the study outcomes and analysis.

A. Search Strategy

The search strategy was developed by reviewing the data needed to answer each one of the research questions. We divided the strategy in two phases. The first one focused on Single System Testing Tools (SSTT), next, we focused on Software Product Lines Testing Tools (SPLTT). The initial set of keywords was refined after a preliminary search that returned many results with few relevance. We used several combinations of search items until achieving a suitable set of keywords.

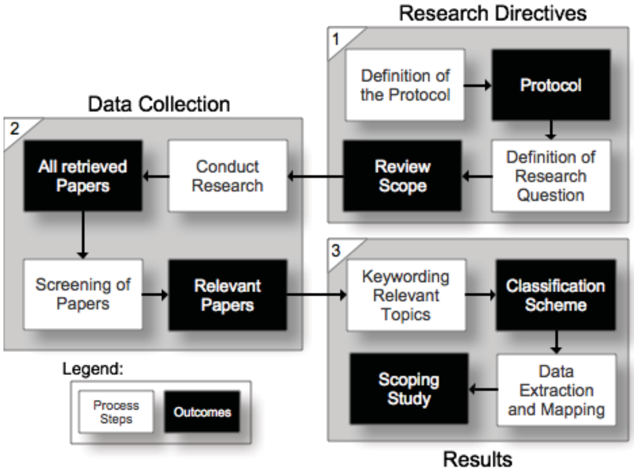


Fig. 1. The Mapping Study Process (adapted from [4])

Furthermore, search strings could then be constructed using boolean AND's and OR's expressions. At the second part of the search, all terms were combined with the term "Product Lines", "Product Family" and "SPL" by using Boolean "AND" operator. All of them were joined by using "OR" operator so that it could improve the completeness of the results. The complete list of search strings is available in Table I.

TABLE I
List of Research Strings

Research Strings
"Testing Tool" **OR** "Test Tool" **OR** "Testing Framework"**OR** "Test Framework" **OR** "Tool for Testing" **OR** "Tool for Test" **OR** "Testing Application" **OR** "Test Application" **OR** "Automatic Testing Tool" **OR** "Automatic Test Tool" **OR** "Automation Testing Tool" **OR** "Automation Test Tool" **OR** "Testing Generation Tool" **OR** "Test Generation Tool" **OR** "Application for Testing"
AND
"product lines" **OR** "product line" **OR** "product family" **OR** "product families" **OR** "SPL"

B. Data Sources

The search was executed using three steps: (i) the search strings in Table I were adjusted and applied in each digital database, and all search strings were systematically checked by more than one author. The list of sources, in alphabetical order, is the following: ACM Digital Library[1], Elsevier[2], IEEE Computer Society Digital Library[3], Science@Direct[4], The DBLP Computer Science Bibliography[5] and Springer Link[6]; (ii) a manual search was performed in the main Journals and Conferences[7]. These libraries were chosen since they are the

[1] http://portal.acm.org/
[2] http://www.elsevier.com/
[3] http://ieeexplore.ieee.org/
[4] http://www.sciencedirect.com/
[5] http://www.informatik.uni-trier.de/ ley/db/
[6] http://springerlink.metapress.com/home/main.mpx
[7] http://wp.me/p157qN-2T

most relevant sources in software engineering [5]; (iii) and finally, the search was also performed using the 'snow-balling' process [12], following up the references in papers and it was extended to include grey literature sources, seeking relevant white papers, industrial (and technical) reports, theses, work-in-progress, and books.

Furthermore, as the described search engines are focused on academic results, and given the fact that the goal was to find the largest number of tools as possible, and these engines would not find commercial tools (as long as they do not have any paper or journal published), these keywords were also used in searches in web search engines, such as Google[8]. In the web engines, the target was tools information and their grey literature.

C. Study Selection

A set of 69 studies involving both single system and SPL testing tools was found. Figure 2 shows each filter applied during the study selection. Moreover, it presents the amount of studies remaining after applying each filter.

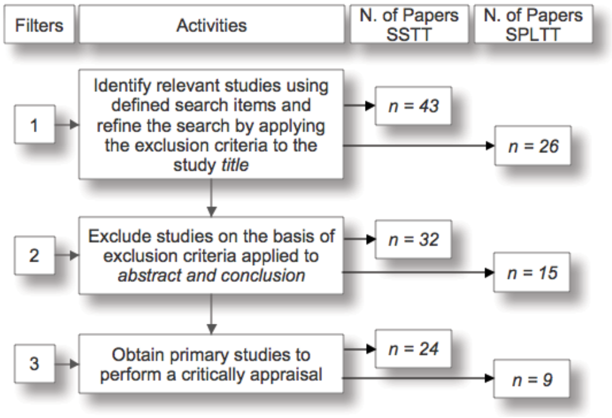

Filters
Activities
N. of Papers SSTT
N. of Papers SPLTT
1
Identify relevant studies using defined search items and refine the search by applying the exclusion criteria to the study *title*
n = 43
n = 26
2
Exclude studies on the basis of exclusion criteria applied to *abstract and conclusion*
n = 32
n = 15
3
Obtain primary studies to perform a critically appraisal
n = 24
n = 9

Fig. 2. Study Selection Summary

The inclusion criteria were used to select all studies during the search step. After that, the exclusion criteria was firstly applied in the studies title and next in the abstracts and conclusions. All excluded studies can be seen by differentiating the results among filters. Regarding the inclusion criteria, the studies were included if they described:

- *Tools that support a testing level.*
- *Tools that support a testing process.*
- *Tools with available executable.*
- *Tools with documentation.*

Studies were excluded if they described:

- *Testing Tools that were not implemented.*
- *Conceptual Testing and conceptual testing frameworks.*
- *Duplicated studies.*

After the first filter, 47 papers were selected (32 from single and 15 from SPL). In the second one, we limited the publication venues to international journals and conferences (no magazines were included). We excluded studies on the basis of exclusion criteria applied to abstract and conclusion.

After this stage, there were 33 papers, which were considered to analysis (24 from single system and 9 from SPL). Figure 3 shows the single system and SPL testing tools regarding the publication years. As it can be seen, there is a peak of SSTT in 2007. Figure 4 shows the amount of publications considering the sources.

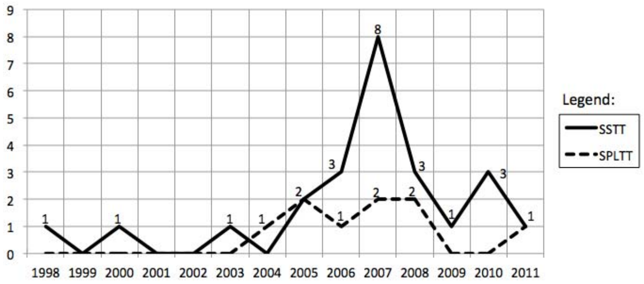


Fig. 3. Distribution of single system and SPL testing tools through their publication years

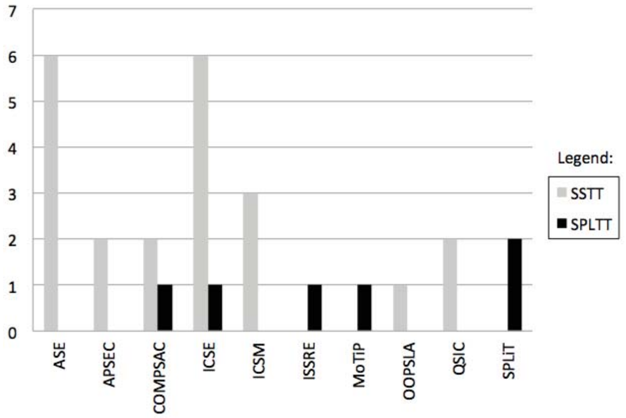


Fig. 4. Amount of Studies vs. sources

1) Reliability of inclusion decisions: The reliability of decisions to include a study is ensured by having multiple researchers to evaluate each study. The study was conducted by two research assistants (the two first authors) who were responsible for performing the searches and summarizing the results of the mapping study, with other members of the team acting as reviewers. A high level agreement existed before the study was included. In case the researches did not agree after the discussion, an expert in the area was contacted to discuss and give the appropriate guidance.

D. Classification Scheme

The studies were categorized based on the key wording process defined by [4]. This process is a way to reduce the time needed in developing the classification scheme and ensuring that the scheme takes the existing studies into account. The abstract, titles and keywords were revisited to

[8] http://www.google.com

TABLE II
STUDIES BY RESEARCH TOPICS

	Validation Research	Evaluation Research	Solution Proposal
SSTT	[13],[14],[15],[16],[17],[18],[19],[20]	[21],[22]	[23],[24],[25],[26],[27],[28],[29],[30],[31],[32],[33],[34],[35],[36]
SPLTT	[37],[38]	[39]	[40],[41],[42],[43],[44],[45]

TABLE III
WHERE THE TOOLS WERE DEVELOPED AND USED

	Academy	Industry	Both
SSTT	[23],[25],[26],[27],[15],[14],[28],[29],[30],[22],[31],[16],[36],[20]	[24],[21],[32],[34],[35],[19]	[13],[17],[33],[18]
SPLTT	[41],[42],[43],[37],[44],[38],[45]	[40]	[39]

identify different facets within the selected studies. More details about the type of research as defined by [4] can be seen in: http://wp.me/p157qN-3i

E. Data Extraction and Synthesis

All the 33 studies were fully read and submitted to a predefined form to accurately record the information obtained by the researchers from the primary studies. The form for data extraction provides some standard information, such as:

- Tool's name;
- Date of data extraction;
- Title, authors, journal, publication details (if available);
- Prototype information (if available);
- Website (if available); and,
- A list of each conclusion and statement found for each question.

Based on the research results, inclusion and exclusion criteria, a set of tools were selected. A brief description presented ordered by the publication year of SSTT tools can be seen in: http://wp.me/p157qN-2Z and SPLTT tools can be seen in: http://wp.me/p157qN-32

IV. OUTCOMES

In this section, each research question is answered by analyzing different point of views, highlighting the evidences gathered from the data extraction process. All of these results populated the classification scheme, which evolved while doing the data extraction.

Initially, we analyzed the studies distribution regarding to the research topics. The classification scheme, detailed in Table II, allowed us to infer that many of the studies are *Validation Research* (31%), *Evaluation Research* (9%) and *Solution Proposal* (60%). On the other hand, no *Philosophical*, *Opinion*, and *Experience Papers* were reported. For this reason, Table II did not show them.

A. Tools Development and Usage

From the selected tools, most of them were developed in the academic environment (14 for single systems and 7 for SPL), while 7 were developed exclusively in industry (6 for single system and 1 for SPL). The remaining 5 tools were developed in both environments, academic and industrial (4 for single systems and 1 for SPL), as detailed in Table III.

[13], [17], and [19], present case studies executed in an academy context. [21], [22], and [39], describe case studies in industry. Finally, [14], [15], [16], [18], [20], [37], and [38], report experiments in order to evaluate the tools. The other studies did not describe empirical evaluation.

B. Software Product Lines Adaptability

According to the results (see Table II), only 33% of the single system testing tools can be used to test SPL. The other ones (*Solution Proposal*) were implemented to specific programming languages, techniques and approaches which cannot be suitable to the SPL context.

Websob [29], Korat [30], CodeGenie [31], JWalk [16], Smart [32], REST [34] and JUnitMX [35] are tools able to be utilized in the SPL testing process.

Testing tools with specific purpose such as: test management, bug reports, security test, can be used in Software Product Lines testing if the methodology of using it is adapted to suit the SPL necessities. Commonalities and variabilities should be considered also.

C. Testing Levels

Many of the analyzed tools have similar functionalities. Moreover, the extracted functionalities have analogous goals, so it was possible to group them. This classification matched the description presented in [6], which defines the following groups:

- **Unit Testing** - Tools that test the smallest unit of software implementation. This unit can be basically a class, or even a module, a function, or a software component [10].
- **Integration Testing** - Tools that test the integration between modules or within the reference in domain-level when the architecture calls for specific domain components to be integrated in multiple systems [10].
- **System Testing** - Tools that ensure that the final product matches the required features [37].
- **Acceptance Testing** - Tools that will be used by customers during the validation of applications [10].
- **Regression Testing** - Even though regression testing is not a test level [6], some tools were developed to work with it. For this reason, we considered regression testing as part of the classification.

This classification can be applied not only for single system testing tools but also for SPL testing tools. The main difference is that SPL divide Testing according to two activities [10]: core asset (grouping Unit and integration testing) and product development (grouping system and acceptance testing). Table IV details the classification of the tools according to the testing level plus regression testing.

D. Testing Tools Evolution

In order to identify the evolution of the tools, we constructed a timeline for SSTT showed in Figure 5. Figure 6 shows the

TABLE IV
TOOL'S CLASSIFICATION ACCORDING WITH TESTING LEVELS PLUS REGRESSION TESTING

	Unit Testing	Integr. Testing	System Testing	Accept. Testing	Regr. Testing
SSTT	[13],[25],[27],[14],[15],[31],[16],[32],[17],[33],[35],[19],[36]	[13],[26],[31],[33]	[23],[28],[29],[30],[21],[22]	[23],[24][18],[20]	[27],[17],[34],[20]
SPLTT	[40],[41],[43],[44]		[39],[37],[45]	[42]	[38]

same idea for SPLTT. Every year, since 2004, at least one tool for Unit Testing Level was published.

There is a clear evolution of tools at the Unit Testing Level. We identified that JTest [14], Jwalk [16], and JUnitMX [35] evolved from the xUnit [41]. A possible reason for this fact was the release of new versions of the Java programming language, which allowed the development of more complex functionalities. Nevertheless, there is no visible evolution in other testing levels of single system and SPL testing tools including regression testing.

V. MAIN FINDINGS

The analysis of the results enables us to identify what have been emphasized in past research and also to identify gaps and possibilities for future research.

Based on the analyzed tools, it was possible to identify that the tools are usually developed to support a specific testing level, under the justification that there are no tools supporting all functionalities of a testing process. None of the selected tools supports the overall SPL life cycle. For example, the prototype tool developed by [39] was created to assist the ScenTED technique, and does not support a general testing process. Moreover, the majority of the testing tools were implemented to solve a specific problem.

The amount of SPL testing tools in academy (78%) is higher than the number of single system testing tools (58%). In industry this percentage is inverted: 25% of the SSTT and 11% of SPLTT. The percentage is equivalent when the tools are applied in both industry and academy (17% for single system and 11% for SPL). In accordance with Table III, the amount of projects applied in industrial context lacks investments.

During the research, we identified the possibility to use single system testing tools such as JUnitMX [35], and CodeGenie [31] to support the SPL testing process, however, it will be necessary to change the methodology of using in order to suit the tool for the SPL context. Another possibility should be implement specific functionalities from SPL such as variability management of the assets, but this will be possible only for open-source applications. Tools such as AsmL [24], and SimC [15] were developed to solve specific problems of a specific environment, thus, the effort to adapt these tools can be impracticable.

Despite the number of SSTT be twice higher than the number of SPLTT, when the tools are organized by testing levels, the percentage of single system and SPL testing tools are equivalent. The only exception was the Integration Testing Level since there was no SPLTT identified for this level.

The amount of SSTT used in industry added with tools used in both academy and industry correspond to 41% of the total. When we focus on SPLTT, this number decreases to 22%. Thus, we believe that SPLTT do not achieve maturity enough to be applied in industrial projects so far. Moreover, the effort required to implement a SPL testing process into organizations hampers the development of SPLTT.

Companies are interested in developing tools that could be used by a large number of consumers. For this reason, they look for attacking problems common to most customers. These tools are implemented to test general functionalities, performance, security, etc. Meanwhile, academy aims to produce tools that solve complex problems, which can explain the large number of Solution Proposal.

Figure 5 shows that 2007 was the year with a higher number of publications describing SSTT (8 papers): 3 in the International Conference on Software Engineering (ICSE'07), 2 in the International Conference on Automated Software Engineering (ASE'07), and 3 in other conferences. These two conferences together published the largest number of papers describing SSTT over the years, 6 papers each one. Coincidentally, 2007 was also the year with more tools focused on System Testing Level. From all the tools of System Testing Level analyzed, only 3 were applied in industry: 2 SSTT and 1 SPLTT.

The lack of SPLTT since 2009 could be explained by the absence of the Software Product Lines Testing Workshop (SPLiT) which is a great responsible by publications of this kind of tools. The SPLiT is part of the Software Product Lines Conference (SPLC) which is the main event of the SPL area.

Finally, we identified a tendency directed to Automatic Generation Tools (50% of the analyzed tools) and according to [46] "The dream would be a powerful integrated test environment which by itself, as a piece of software is completed and deployed, generating the most suitable test cases, executing them and finally issuing a test report".

However, tools for automatic test case, test input, and test data generation, work independently. For this reason, the need for a framework to integrate these tools still exists [3].

VI. THREATS TO VALIDITY

The main threats to validity identified in the review are described next:

- **Tools selection.** A possible threat in such review is to exclude some relevant tool. In order to reduce this possibility, the tools selection was based on the identification of the key research portals in computer science and a wide range of web search engines, besides the main journals and conferences. The defined criteria intended to select tools supporting some functionalities of the SPL Test Process and not just supporting specifics requirements.
- **Data Extraction.** In order to ensure the validity, multiple sources of data were analyzed, i.e. papers, prototypes,

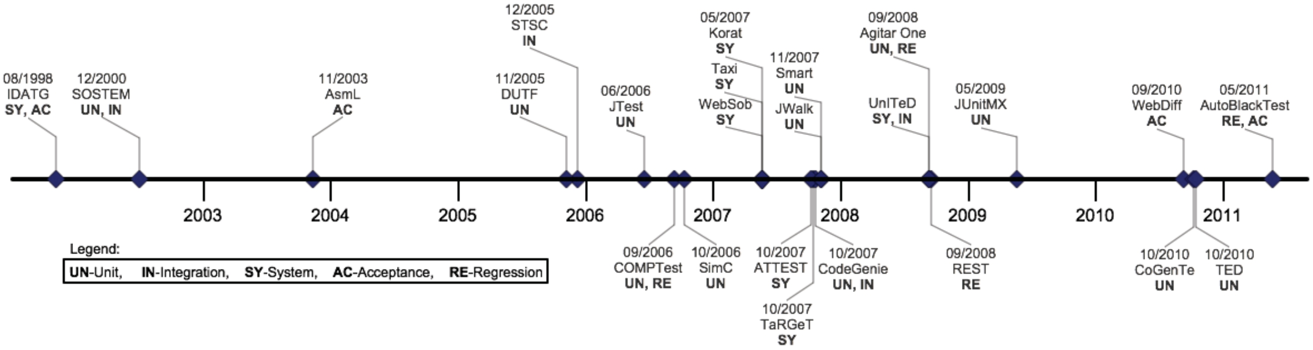


Fig. 5. Single System Testing Tools Timeline

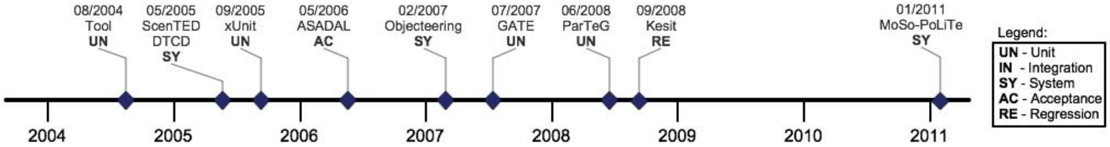


Fig. 6. SPL Testing Tools Timeline

technical reports, white papers and manuals, in addition to the tools executable.

- **Research Questions.** The questions defined could not have covered the whole Testing Tools, which implies that some one cannot find answers to the questions that concern them. To mitigate this feasible threat, we had several discussions with project members and experts in the area, allowing the calibration of the question. Thus, even if we had not selected the most adequate set of questions, we attempted to address the most asked and considered *open issues* in the field.
- **Publication Bias.** We cannot guarantee that all relevant primary studies were selected. It is possible that relevant studies were not chosen during the search process. We mitigate this threat by following references in the primary studies.
- **Research Strings.** The terms used in the research strings can have many synonyms, and it is possible that some work were overlooked. To mitigate this threat, we had discussions with project members and experts in the area.

VII. Conclusion

This paper presented a mapping study on single system and software product lines testing tools, whose main goal was to identify how the available tools are supporting the SPL Testing process. Through the review, it was possible to identify, which current functionalities are being supported by the tools, and which ones should be present in every tool based on their priorities.

The research was conducted using techniques from mapping study, a helpful approach for identifying the areas where there is sufficient information for a systematic review to be effective, as well as those areas where is necessary more research [12].

The great majority of the studies (60% of Solution Proposal) were implemented to solve specific problems that avoid the adaptability of the tools for the SPL context. For this reason, it will be less costly create new tools than adapt the existing ones.

Moreover, we identified an increasing number of automatic testing case generation tools. As a result, one of the next challenges will be learn how to automate this "generated" test cases for the products of the SPL.

Finally, we noticed that publications describing industrial experiences are rare in literature. The existing case studies and experiments report only small projects, containing results obtained from tools that solve specific problems related to testing. Consequently, more experiments involving SPL testing tools are needed.

As future work, we are incorporating the results of this research in the development of a software product lines testing 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] F. J. van der Linden, K. Schmid, and E. Rommes, *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering*. Berlin: Springer, 2007.

[2] P. Clements and L. Northrop, *Software Product Lines: Practices and Patterns*. Boston, MA, USA: Addison-Wesley, 2001.

[3] A. Tevanlinna, J. Taina, and R. Kauppinen, "Product Family Testing: a Survey," *ACM SIGSOFT Software Engineering Notes*, vol. 29, pp. 12–12, 2004.

[4] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson, "Systematic Mapping Studies in Software Engineering," *12th International Conference on Evaluation and Assessment in Software Engineering*, 2008.

[5] B. Kitchenham and S. Charters, "Guidelines for performing Systematic Literature Reviews in Software Engineering," *Software Engineering Group School of*, vol. 2, p. 1051, 2007.

[6] J. D. McGregor, "Testing a Software Product Line," CMU/SEI - Software Engineering Institute, Tech. Rep. CMU/SEI-2001-TR-022, 2001.

[7] O. O. Edwin, "Testing in software product lines," M.Sc. Dissertation, School of Engineering at Blekinge Institute of Technology, 2007.

[8] E. Y. Nakagawa, A. S. Simão, F. C. Ferrari, and J. C. Maldonado, "Towards a Reference Architecture for Software Testing Tools," in *International Conference on Software Engineering & Knowledge Engineering*. Knowledge Systems Institute Graduate School, 2007, pp. 157–162.

[9] B. P. Lamancha, M. P. Usaola, and M. P. Velthius, "Software Product Line Testing - A Systematic Review." in *4th International Conference on Software and Data Technologies*, B. Shishkov, J. Cordeiro, and A. Ranchordas, Eds. INSTICC Press, 2009, pp. 23–30.

[10] P. A. M. S. Neto, I. C. Machado, J. D. McGregor, and S. R. L. M. Eduardo Santana Almeida and, "A systematic mapping study of software product lines testing," *Inf. Softw. Technol.*, vol. 53, pp. 407–423, 2011.

[11] E. Engström and P. Runeson, "Software product line testing - a systematic mapping study," *Information & Software Technology*, vol. 53, no. 1, pp. 2–13, 2011.

[12] D. Budgen, M. Turner, P. Brereton, and B. Kitchenham, "Using mapping studies in software engineering," in *Proceedings of PPIG 2008*. Lancaster University, 2008, pp. 195–204.

[13] S. Liut, T. Fukuzakit, and K. Miyamoto, "A GUI and testing tool for SOFL," *7th Asia-Pacific Software Engineering Conference*, no. 11694173, pp. 421–425, 2000.

[14] T. Xie and D. Notkin, "Tool-assisted unit-test generation and selection based on operational abstractions," *21st IEEE International Conference on Automated Software Engineering*, vol. 13, no. 3, pp. 345–371, 2006.

[15] Z. Xu and J. Zhang, "A Test Data Generation Tool for Unit Testing of C Programs," *6th International Conference on Quality Software*, pp. 107–116, 2006.

[16] A. J. H. Simons, "JWalk: a tool for lazy, systematic testing of java classes by design introspection and user interaction," *22nd IEEE International Conference on Automated Software Engineering*, vol. 14, no. 4, pp. 369–418, 2007.

[17] B. Daniel and M. Boshernitsan, "Predicting Effectiveness of Automatic Testing Tools," *23rd IEEE/ACM International Conference on Automated Software Engineering*, pp. 363–366, 2008.

[18] S. Choudhary, H. Versee, and A. Orso, "A Cross-browser Web Application Testing Tool," in *Software Maintenance, 2010 IEEE International Conference on*, 2010, pp. 1–6.

[19] A. C. Rajeev, P. Sampath, K. C. Shashidhar, and S. Ramesh, "CoGente: A Tool for Code Generator Testing," in *Proceedings of the IEEE/ACM international conference on Automated software engineering*. New York, NY, USA: ACM, 2010, pp. 349–350.

[20] L. Mariani, M. Pezzè, O. Riganelli, and M. Santoro, "AutoBlackTest: a tool for automatic black-box testing," in *Proceeding of the 33rd international conference on Software engineering*. New York, NY, USA: ACM, 2011, pp. 1013–1015.

[21] Y. Ren and F. Chang, "ATTEST: A Testing Toolkit for Validating Software Properties," *23rd IEEE International Conference onSoftware Maintenance*, pp. 469–472, 2007.

[22] S. Nogueira, E. Cartaxo, D. Torres, E. Aranha, and R. Marques, "Model Based Test Generation: An Industrial Experience," in *1st Brazilian Workshop on Systematic and Automated Software Testing*, 2007.

[23] A. Beer, S. Mohacsi, and C. Stary, "IDATG: An Open Tool for Automated Testing of Interactive Software," *22th Annual International Computer Software and Applications Conference*, pp. 6–11, 1998.

[24] M. Barnett, W. Grieskamp, W. Schulte, N. Tillmann, and M. Veanes, "Validating Use-Cases with the AsmL Test Tool," *3rd International Conference on Quality Software*, pp. 238 – 246, 2003.

[25] H. Wu and J. Gray, "Automated Generation of Testing Tools for Domain-Specific Languages," in *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*. ACM, 2005, p. 439.

[26] R. Shukla, P. Strooper, and D. Carrington, "Tool Support for Statistical Testing of Software Components," in *12th Asia-Pacific Software Engineering Conference*. IEEE Computer Society, 2005, p. 8.

[27] J. Gao, D. Gopinathan, Q. Mai, and J. He, "A Systematic Regression Testing Method and Tool For Software Components," *30th Annual International Computer Software and Applications Conference*, pp. 455–466, 2006.

[28] A. Bertolino, J. Gao, E. Marchetti, and A. Polini, "TAXI - A Tool for XML-Based Testing," *29th International Conference on Software Engineering*, pp. 53–54, 2007.

[29] E. Martin, S. Basu, and T. Xie, "WebSob: A Tool for Robustness Testing of Web Services," *29th International Conference on Software Engineering*, pp. 65–66, 2007.

[30] A. Milicevic, S. Misailovic, D. Marinov, and S. Khurshid, "Korat: A Tool for Generating Structurally Complex Test Inputs," in *29th International Conference on Software Engineering*. IEEE Computer Society, 2007, pp. 771–774.

[31] L. Lemos, O. Augusto, S. Bajracharya, and J, "CodeGenie:: a Tool for Test-Driven Source Code Search," *Proceedings of the 22th annual ACM SIGPLAN conference on Object-Oriented Programming, Systems, Languages, and Applications*, pp. 917–918, 2007.

[32] Q. Xie, M. Grechanik, and M. Hellige, "Smart: A Tool for Application Reference Testing," in *22nd IEEE/ACM international conference on Automated software engineering*. ACM, 2007, pp. 537–538.

[33] F. Pinte, N. Oster, and F. Saglietti, "Techniques and tools for the automatic generation of optimal test data at code, model and interface level," *30th international conference on Software engineering*, p. 927, 2008.

[34] Q. Xie, M. Grechanik, and C. Fu, "REST: A Tool for Reducing Effort in Script-based Testing," *24th IEEE International Conference onSoftware Maintenance*, pp. 468–469, 2008.

[35] J. Wloka, B. G. Ryder, and F. Tip, "JUnitMX - A Change-aware Unit Testing Tool," *31st International Conference on Software Engineering*, pp. 567–570, 2009.

[36] E. Mishra and Y. Sonawane, "TED: Tool for Testing and Debugging uDAPL," in *Architectures for Networking and Communications Systems (ANCS), 2010 ACM/IEEE Symposium on*, 2010, pp. 1–2.

[37] C. Nebut, Y. Traon, and J. Jezequel, "System Testing of Product Lines: From Requirements to Test Cases," *Software Product Lines*, pp. 447–477, 2007.

[38] E. Uzuncaova, D. Garcia, S. Khurshid, and D. Batory, "Testing Software Product Lines Using Incremental Test Generation," *19th International Symposium on Software Reliability Engineering*, pp. 249–258, 2008.

[39] 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.

[40] Z. Stephenson, Y. Zhan, J. Clark, and J. McDermid, "Test Data Generation for Product Lines - A Mutation Testing Approach," in *3rd International Workshop on Software Product Line Testing*. Citeseer, 2004, p. 13.

[41] M. Galli, O. Greevy, and O. Nierstrasz, "Composing Unit Tests," in *4th International Workshop on Software Product Line Testing*. Citeseer, 2005, pp. 16–22.

[42] K. Kim, H. Kim, M. Ahn, M. Seo, Y. Chang, and K. C. Kang, "ASADAL: A Tool System for Co-Development of Software and Test Environment based on Product Line Engineering," *28th International Conference on Software Engineering*, pp. 783–786, 2006.

[43] Y. Feng, X. Liu, and J. Kerridge, "A product line based aspect-oriented generative unit testing approach to building quality components," *Proceedings of the 31st Annual International Computer Software and Applications Conference*, pp. 403–408, 2007.

[44] S. Weissleder, D. Sokenou, and B. Schlingloff, "Reusing State Machines for Automatic Test Generation in Product Lines," *1st Workshop on Model-based Testing in Practice*, p. 19, 2008.

[45] S. Oster, I. Zorcic, F. Markert, and M. Lochau, "MoSo-PoLiTe: Tool Support for Pairwise and Model-Based Software Product Line Testing," in *Proceedings of the 5th Workshop on Variability Modeling of Software-Intensive Systems*. New York, NY, USA: ACM, 2011, pp. 79–82.

[46] A. Bertolino, "Software testing research: Achievements, challenges, dreams," in *2007 Future of Software Engineering*, ser. FOSE '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 85–103.

Optimal Variability Selection in Product Line Engineering

Rafael Pinto Medeiros*, Uéverton dos Santos Souza†, Fábio Protti† and Leonardo Gresta Paulino Murta†
*Universidade do Estado do Rio de Janeiro
Rio de Janeiro, Rio de Janeiro, Brazil
Email: rafaelmedeiros@uerj.br
†Instituto de Computação, Universidade Federal Fluminense
Niterói, Rio de Janeiro, Brazil
Email: {usouza,fabio,leomurta}@ic.uff.br

***Abstract*—Software Configuration Management is being adopted with success in the development of individual products, mainly supporting the creation of revisions and sporadically supporting the creation of variants via branches. However, some emerging software engineering methods, such as product line engineering, demand a sound support for variants to generate customized products from product lines. The adoption of branches in this scenario does not scale due to the huge number of interdependent variants. The main goal of this paper is to systematize a way to select the best variants of a product line to generate products according to specific user needs. Moreover, our paper contributes on providing an algorithm for product generation from product lines. Our algorithm can be easily implemented in the existing product line approaches as an alternative for product selection.**

***Keywords*-And/Or Graphs, Software Configuration Management, Software Versioning.**

I. Introduction

Software Configuration Management (SCM) is a discipline applied during the software development process to control the software evolution [9]. As changes can happen anytime during the software development process, SCM activities are developed to identify the changes; to assure the changes are being correctly implemented; and to inform the changes to people who have interests on it [6]. Due to that, it is possible to conclude that the main objective of SCM is not to avoid changes, but provide control and coordination over the changes. Moreover, it is also concerned on providing consistency among interdependent components and allowing the reconstruction of previous states of the software.

As an important system of SCM, the Version Control System (VCS) is responsible for managing different versions of a product. Usually, VCS are developed through models that define the objects to be versioned, version identification and organization, as well as operations to retrieve previous versions and create new ones. However, versions can serve for different purposes [5]. Versions that are used to replace other versions of the same component are called revisions. On the other hand, versions that live together with other versions of the same component, acting as alternatives, are called variants.

SCM is being adopted with success in the development of individual products, mainly supporting the creation of revisions and sporadically supporting the creation of variants via branches. However, some emerging software engineering methods, such as product line engineering, demand a sound support for variants to generate customized products from product lines. The adoption of branches in this scenario does not scale due to the huge number of interdependent variants.

In product line engineering, a software product line is composed to represent the commonalities and variabilities of a software family. According to [4] a product line is usually combined with feature models and configuration knowledge, responsible to identify the possible product features and how these features interplay. The derivation process is fundamental in product line engineering. This process consists in composing specific products from the product line according to some user requirements. According to [8] the existing approaches to model product line architectures are predominantly focused on enumerating the available component versions for each possible product that can be generated from the product line. However, conceptual differences in product features and their interrelationships are not easily expressed in the available modeling constructs. On the other hand, a goal-based approach provides a natural mapping to modeling product line architectures, considering the user needs during the product generation.

The main goal of this paper is to systematize a way to select the best variants of a product line to generate products according to specific user needs. This can result in VCS that are better prepared to support product line engineering and other methodologies that focus on the conception of families of products. Moreover, our approach formalizes this product composition according to the SCM terminology. Product line researchers can build upon our approach to implement their derivation process according to their specific technologies (i.e., features model, architecture description languages, etc.).

This paper is organized into 5 sections besides this introduction. Section 2 presents some background related to software versioning concepts. Section 3 introduces the combinatorial problem that emerges from the product line scenario. Section 4 presents our approach to generate the optimal product from a product line. Section 5 presents some related works on software versioning. Finally, section 6 presents final considerations and future works.

II. Background on Software Versioning

During the development process, software engineers need to build specific versions of the software. A software version is structured by components, and each component also have specific versions. However, the selection of different components or different versions of the same components leads to different versions of the software as a whole. At an exponential rate, different software versions start to become possible of building, even versions that are not aligned to the user desires or requirements [5].

A version model identifies and organizes items that should be versioned and items that should not be versioned. Each SCM system provides its own version model according to the target domain and builds over its own formalism. According to Conradi and Westfechtel [5], there are many ways to represent a version model, such as file-based models, where versioning is applied on files and directories, and data-based models, where versioning systems manage versions of objects stored in a database. Other resources are commonly applied to express versioning rules, such as textual languages.

The version model can be described in terms of a product space and a version space. In order to achieve proper existence of a software, it is necessary to define what composes the software itself. In other words, the software components, their role to the final product, their functions inside the software, and their relationships to each other should be defined. This arrangements of components is defined in [5] as the product space. The product space represents a universe of items and their relationships, without considering their versions.

On the other hand, based on the definition in [5], the version space represents the universe of possible versions of a software, highlighting the common traits and differences of the versions of an item. The transformation of a non versioned item (in the product space) into a single, double, or multi versioned item can be seen as a binding of an specific moment in time of the product space with the version space. Hence, a software version is composed by versions of the software components, and generated by the combination of the product space with a moment of existence of each component in the version space. An item without this moment of existence in the version space is a non versioned item, with its changes implemented through overwriting.

A set of versions can be defined basically in two ways [5]: extensional versioning and intensional versioning. We can differ extensional from intensional versioning due to the reasons that demand the generation of a new version. Extensional versioning is realized through enumerating its components' versions; from this point, the user is able to retrieve a version v_x, apply changes over v_x, and generate a new version v_z. Intensional versioning is capable of generating versions from larger spaces to satisfy a set of goals established by the user.

As a consequence, extensional versioning only allow the retrieval of previously created versions, while intensional versioning allows the retrieval of versions on demand, combining component versions that may never worked together before and that can potentially generate inconsistent software versions in terms of the user needs. This is one of the main reasons why current SCM systems usually adopt extensional versioning.

III. The Optimal Intensional Version Problem

An *object base* is defined as a combination of product space and version space, comprehending all the versions of a software [5]. This base contains all the software components, all their versions, non versioned objects and their relationships.

The arrangement of an intensional version can be seen as a selection of objects inside the base in a way that the selected objects are enough to build the product version. This selection is structured to satisfy the needs that motivated the product development. During the versioning process, these needs are transformed into affirmatives, named versioning rules. Therefore, the selection is directed by a set of versioning rules.

This method leads to a combinatorial problem inside the intensional versioning. From a large number of potential versions, only a few of them sustain the consistency needed to satisfy the set of versioning rules. In summary, the configuration process is based on satisfying restrictions and demands to lead to a functional resulting software version. The versioning rules in this article represent restrictions and demands, and the object base is the universe of all possible versions, including inconsistent ones.

According to Conradi and Westfechtel[5], the most difficult factor when facing the combinatorial problem on intensional versioning is to eliminate inconsistent versions. After that, it relays on the configurator to build the version that matches a certain query.

We present a formalization to the optimal intensional versioning problem as follows:

Problem: OIV – Optimal Intensional Version
Input: An object base and a set of versioning rules
Output: To find in the object base, if possible, an optimized version of the software that satisfies the set of versioning rules.

In this article the software's object base is represented through an And/Or graph, according to the representation introduced in [5]. And/Or graphs [12] provide a general model for integrating product space and version space. An And/Or graph is a directed graph G, such that every vertex $v \in V(G)$ possesses a label $f(v) \in \{And, Or\}$. In this graph, the directed edges represent dependency relationship among the vertices: And-type vertices (represented through an arc between its out-edge) depend strictly on all its out-neighbors; Or-type vertices depend only on one of its out-neighbors.

To represent the object base the source vertex maps to the software as a whole, and the other vertices map to software modules or components and its versions. In this graph, the And out-edge represent composition relationship and the Or out-edge represent possible versions of an item.

According to [5] a distinction is made between And and Or

edges, which emanate from And and Or nodes, respectively. An unversioned product can be represented by an And/Or graph consisting exclusively of And nodes/edges. A versioned product is modeled by introducing Or nodes. Versioned objects and their versions are represented by Or nodes and And nodes, respectively.

For example, Figure 1 illustrates an And/Or graph representing a base of objects. It is important to notice that fine-grained visibilities of an object base can also be represented by And/Or graphs using the same formalisms discussed in this paper. However, we focused our examples on a coarse-grained visibility to allow a better understanding of the problem and our proposed solution. The main difference of coarse and fine granularity is the number of components and their versions, which enforces the necessity of faster algorithms to solve the problem when fine-grained components are in place.

IV. Solution to the OIV Problem

In this section we introduce a transformation of the OIV problem into a combinatorial problem related to And/Or graphs (MIN–AND/OR). Through this transformation we systematize a process where the versioning rules are converted into weights at the edges of the object base's graph. Thus, we present an approach that enables the development of algorithms for solving the problem of intentional versioning. In addition, we present a backtracking algorithm for MIN–AND/OR which in turn solves the OIV problem.

The MIN–AND/OR problem consists on finding a subgraph (*solution subgraph*) that matches a set of restrictions considering a weighted And/Or graph with a source vertex s. We introduce the definition of the MIN–AND/OR Problem[11] as follows:

Problem: MIN–AND/OR

Input: An acyclic And/Or graph G=(V, E) properly connected and weighted with a source vertex $s \in V$, where each vertex v possesses a label $f(v) \in \{And, Or\}$ and each edge possesses a weight $\tau(e) \geq 1$.

Output: A subgraph $H = (V', E')$ of G such as the sum of the weights of its arcs is minimum, and satisfies:

1) $s \in V'$;
2) if $v \in V'$ and $f(v) = And$ then all out-arcs of v must belong to E';
3) if $v \in V'$ and $f(v) = Or$ then exactly one of the out-arcs of v must belong to E'.

To transform the OIV Problem into the MIN–AND/OR Problem, it is necessary to weight the And/Or graph G (representing the object base) according to the query built from the versioning rules.

A. Weighting the And/Or graph G

Considering the And/Or graph G it is possible to add weights to its edges using two types of versioning rules:

- Configuration Rules: a series of affirmative statements originated from questions asked to the stakeholders of the software; from these affirmatives, it is possible to exclude some versions from the space of possible versions. As an example of the formulation of the configuration rules, the configuration manager can ask how the user communicates with the system (local access, remote access). It becomes possible then to formulate the configuration rule: the software must provide local access support. This configuration rule discards the components that provide remote access or portability features, because they are not demanded. These rules consider the *functional* requirements of the product being generated, and can be replaced by an existing feature-based approach if the product line engineering process already has one in place.
- Classification/Qualification rules: a set of items classified/qualified considering their priority to the stakeholders according to the needs and demands of the software. For instance, the stakeholders can ask for a product that allows high-end control capabilities and is efficient. The configuration manager then is able to formulate the Classification/Qualification rules: Control Capability, Efficiency. Versions of components that best excel at those two items are preferred. These rules consider the *non-functional* requirements of the product being generated. They are important to solve situations where there are open alternatives even after imposing the rules related to *functional* requirements.

1) Applying Configuration Rules: The set of configuration rules characterizes the first process of weighting the graph of the base of objects. The possibilities that do not accomplish the defined configuration rules are unable of existing in a solution-subgraph with non-infinite cost. Along the weighting process, the configuration rule analysis dictates if an edge receives an infinite weight or not, sometimes excluding a large amount of versions (by excluding one component, all its versions are automatically excluded).

The configuration rules are therefore defined as affirmative statements, originated from a set of questions answered in cooperation with the stakeholders. The software engineers should create questions with a high capability of constraining the version space.

When applying configuration rules, the questions work as a way to formulate demanding affirmatives - the configuration rule itself. All edges of G pointing to vertices that oppose the affirmative receive infinite weight; all other vertices have their edges set with weight 1. According to this algorithm, a version with non-infinite weight is a version that matches the configuration rule not necessarily by accomplishing the configuration rule, but by not going directly against it.

2) Applying Classification/Qualification Rules: With the application of the classification/qualification rules, edge values bounded to each component will emerge. By doing so, some versions shall protrude, therefore distinguishing themselves from the others. The stakeholder must classify some criteria, according to his/her priority, aiming to reflect what is expected in the final version of the software. In other words, the

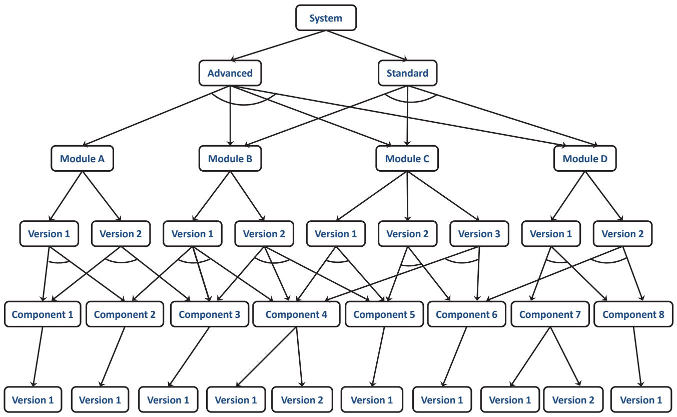


Fig. 1. Example of a object base represented by an And/OR graph.

stakeholder is demanded to prioritize non-functional aspects of the software.

In this paper we named these criteria as Classification/Qualification rules. As an example of classification rules, the stakeholders can be asked to classify the following items according to their priority: Control Capability, Efficiency, and Support. It becomes possible, after this step, to choose the version that best matches the stakeholder's priorities.

The Classification/Qualification rules initiates by considering a set of criteria $\{s_1,\ s_2,\ s_3,...,s_k\}$, in general, some of these criteria can be obtained from the ISO/IEC 9126 for the evaluation of software quality [2], such as: usability, efficiency, maintainability and portability. These criteria are classified by the stakeholder with the tags *high relevancy*, *regular relevancy*, *low relevancy*. Next, the software engineer classifies each component of the object base according to the quality (*bad*, *regular*, *good*, *excellent* or *does not interfere/ not relevant*) of each s_j criterion.

After that, it is proposed a weighting process of the And/Or graph as follows:

To each component c_i do:

1) Add
   - weight 0 to the criterion classified as excellent or not relevant;
   - weight 1 to the criterion classified as good;
   - weight 2 to the criterion classified as regular;
   - weight 3 to the criterion classified as bad.
2) Calculate
   - the sum of the weights of the high relevancy criteria as HR;
   - the sum of the weights of the regular relevancy criteria as RR;
   - the sum of the weights of the low relevancy criteria as LR;
3) Sum W to the weight of the vertex in-edge that represents the component c_i ($W = 3HR + 1.5RR + LR$).

The W value is calculated to ensure that the weight of a high relevancy criterion is the double of the regular relevancy criterion weight and the triple of the low relevancy criterion weight. Consequently, a high relevancy criterion classified/qualified as good corresponds to a regular relevancy criterion classified as regular or to a low relevancy criterion classified as bad. Clearly the higher relevancy criteria weights more, what is justified because of the MIN-AND/OR problem structure. The MIN-AND/OR Problem consists on the weight minimization in which the higher relevance criteria weight is expected to be the smallest. Figure 2 illustrates the object base shown in Figure 1 after applying a possible set of versioning rules.

At this point it is important to highlight that other possible versioning, with different versioning rules, would produce another weighting of the G graph; however, as the object base remains the same, the graph itself is not rebuilt. In addition, even when there are changes on the object base, the G graph is capable of easy adapting and still has not to be rebuilt.

With the And/Or graph properly weighted, it is possible to state that:

1) a version of the software corresponds to a subgraph solution-subgraph) of G, such as:
   - The source vertex (representing the software) belongs to solution-subgraph.
   - If a And-type vertex belongs to the solution-subgraph then all its out-edges do as well.
   - If an Or-type vertex belongs to the solution-subgraph then exactly one of its out-edges does as well.

   This observation is easily verifiable. The out-edges of an And vertex represents composition, that is, if a module belongs to a version z so do all of its components. On the other hand, out-edges of an Or vertex represents version options of a component; as a software uses at most one version of each component, this observation is true.
2) To find the optimal intensional version of the OIV Problem corresponds to find the solution-subgraph with minimum cost of the G graph. In our example, the

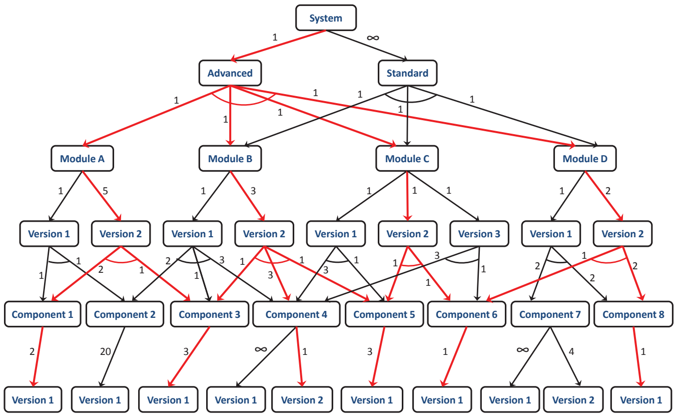


Fig. 2. And/Or graph G representing an object base after applying the versioning rules.

optimal intensional version highlighted in red in Figure 2.

From these statements it becomes possible to adopt the existing results in the literature regarding the MIN–AND/OR Problem to achieve results to the OIV Problem. Among the results of the MIN–AND/OR Problem, it is known that the problem is $NP\text{-}Hard$ in general. However, it becomes polynomial when the G graph is a tree. In addition, it is known that finding a viable solution (not necessarily optimal) is polynomial [11], which means finding a version that satisfies the configuration rules of the OIV Problem is also polynomial.

B. Backtracking algorithm to The MIN–AND/OR *Problem*

At this point, it is known that the And/Or Graph represents the software object base; consequently a solution subgraph of G corresponds to a specific version of the software. We present the Algorithm 1 that, considering an And/Or graph G, returns its optimal solution-subgraph, if there is such. In other words, this algorithm finds the optimal solution to the MIN–AND/OR problem and consequently to the OIV problem.

The algorithm is divided in 3 parts; in the first part it realizes a topological sorting of the G vertices, which is possible because G is an acyclic graph. The sorting assures that a vertex v_j ($1 \leq j \leq n$) is always before its out-neighbors on the vertices arrangement.

Next, the procedure Generate is called to enumerate all possible $solution\text{-}subgraphs$ of G. These subgraphs are stored in V, where:

1) The vertices with the register flag set to 1 belong to the current solution-subgraph.
2) If the position i in V represents an Or-type vertex that belongs to the current subgraph, then exactly one index j is stored in V[i], where v_j is an out-neighbor of v_i.
3) If the position i in V represents an And-type vertex that belongs to the current subgraph, then a set with the index j of each vertex v_j which is an out-neighbor of v_i is stored in V[i].
4) If the position i in V represents a $sink$ then an empty set is stored in V[i].

With the V array properly populated, it becomes possible to go through it building the solution-subgraph and calculating its weight, which is done by the $Cost$ procedure. When the algorithm terminates, the result is the solution-subgraph with the smallest weight.

The complexity of the algorithm is $O(n.K)$ where K is the number of possible $solution\text{-}subgraphs$, and the time to generate each solution-subgraph is $O(n)$.

V. RELATED WORK

In [1], Ghose and Sundararajan presented a work for measuring software quality using pricing and demand data, quantifying the degradation associated with software versioning.

Conradi and Westfechtel [5] introduced a uniform method to represent version models using graphs, providing to configuration managers a more flexible and illustrative way to work with intensional and extensional versioning. In the same direction, in this paper, we show that intensional versioning rules can be represented by weights on the graph's edges, and the main problem on intensional versioning can be seen as a classic combinatorial problem.

According to [7], product derivation is the process of making decisions to select a particular product from a product line and to customize it for a particular purpose. In product derivation, the variability provided by the product line is communicated to the users of the product line and based on customers' requirements, variants are selected from the product line thus resolving the available variability. The most important requirement for tool-supported product derivation is obviously to support resolving variability. Users need tools that present the available variability and let users decide about choices interactively. Many SPLE tools for variability resolution are model-based, e.g., they visualize feature or decision models and allow users resolving variability by selecting features [3] or making decisions [10]. The approach presented in this paper can be adapted for variability resolution merging selecting features and making decisions.

```
Algorithm 1: Backtracking for MIN–And/Or
input  : An $And/Or$ graph $G$; two arrays $V$ and $SS$ of $n$
         positions (inicially empty).
output: $SS$ storing the optimal solution-subgraph of $G$.
begin
    Assume $v_1, v_2, v_3, ..., v_n$ an arrangement of the vertex of
    G, given by an topological sorting;
    for $i$:=1 to $n$ do
        if $v_i$ is an $And$-type vertex then
            V[i].out:=$O_i$;
            ($O_i$ is the set of index of the out-neighbours of $v_i$);
        else
            V[i].out:={};
        if $i = 1$ then
            V[i].flag:=1;
        else
            V[i].flag:=0;
        V[i].in:={}
    smallest:=$\infty$;
    Generate(1,smallest,V,SS);
end
procedure Generate($i$, smallest:integer,$V$, $SS$:array)
    if $i \neq n$ then
        if V[i].flag=0 or $v_i$ is sink then
            Generate(i+1,smallest,V,SS);
        else
            if $v_i$ is an $Or$-type vertex then
                foreach out-neighbor $v_j$ of $v_i$ do
                    V[i].out := {j};
                    V[j].flag:=1;
                    V[j].in := V[j].in $\cup$ {i};
                    Generate(i+1,smallest,V,SS);
                    Clear(i,{j});
            else
                foreach out-neighbor $v_j$ of $v_i$ do
                    V[j].flag:=1;
                    V[j].in := V[j].in $\cup$ {i};
                Generate(i+1,smallest,V,SS);
    else
        if Cost(V,1) $\leq$ smallest then
            SS:=V;
            smallest := Cost(V, 1);
end
procedure Cost($V$: array $i$: integer)
    if V[i].out $\neq$ {} then
        foreach $j \in V[i].out$ do
            value:= weight of edge($v_i, v_j$) + Cost(V,j)
        Cost:=value;
    else
        Cost:=0;
end
procedure Clear($k$: integer $O_k$: set of integer)
    foreach $j \in O_k$ do
        V[j].in := V[j].in $\setminus$ {k};
        if V[j].in = { } then
            if V[j].out $\neq$ { } then
                Clear(j,V[j].out);
            V[j].flag:=0;
end
```

VI. Conclusion

In this paper we built upon the representations of version models introduced by Conradi and Wesfechtel [5] and proposed an approach of graph weighting and search for obtaining the optimal intensional version of a software system according to the preferences of the stakeholder. We showed that a query over the object base can be translated into a weighting of a And/Or graph G, where its edges are weighted according to specific criteria and priorities. In addition, we presented a transformation of the Optimal Intensional Versioning Problem to the combinatorial MIN–And/Or Problem, thus utilizing some of the MIN–And/Or existing results to solve the OIV Problem. Finally, we presented an algorithm that takes the weighted And/Or graph G and finds, if possible, the optimal version, that is, the version with the smallest cost that matches all the existing conditions over the referred object base.

For future work, we would like to apply the proposed algorithm in projects with different characteristics to evaluate how the algorithms behave and if there are situations where intensional versioning should be avoid at all. Moreover, we intend to investigate situations where the stakeholder preferences or even the criteria relevance change dynamically. These situations are common in dynamic software product lines, used in self-adaptive systems at runtime.

References

[1] A. Ghose, A. Sundararajan, Software versioning and quality degradation? An exploratory study of the evidence, Leonard N. Stern School of Business, Working Paper CeDER, New York, NY, USA, pp. 05-20, July 2005.

[2] ISO, ISO/IEC 9126 - Software engineering - Product quality, International Organization for Standardization, 2001.

[3] K. Czarnecki, S. Helson, U.W. Eisenecker, Staged configuration using feature models, Proc. of the 3rd International Software Product Line Conference (SPLC 2004), Springer, Berlin/Heidelberg, Boston, MA, USA, 2004, pp. 266-283.

[4] K.C. Kang, J. Lee, P. Donohoe, Feature-Oriented Product Line Engineering, IEEE Software, v. 19, issue 4, pp. 58-65, July/August 2002.

[5] R. Conradi, B. Westfechtel, Version models for software configuration management. ACM Computing Surveys, v. 30, issue 2, pp. 232-282, June 1998.

[6] R. Pressman, Software Engineering: A Practitioner's Approach. McGraw, 2009.

[7] R. Rabiser, P. Grünbacher, D. Dhungana, Requirements for product derivation support: Results from a systematic literature review and an expert survey, Information and Software Technology, v. 52, pp. 324-346, 2010.

[8] S. A. Hendrickson, A.van der Hoek, Modeling Product Line Architectures through Change Sets and Relationships, 29th International Conference on Software Engineering (ICSE'07), pp. 189-198, 2007.

[9] S. Dart, Concepts in configuration management systems, SCM 91 Proceedings of The 3rd International Workshop on Software Configuration Management, ACM Press, New York, NY, USA, pp. 1-18, June 1991.

[10] T. Asikainen, T. Soininen, T. Männistö, A Koala-based approach for modelling and deploying configurable software product families, Proc. of the 5th International Workshop on Product-Family Engineering (PFE 2003), Siena, Italy, Springer, Berlin/Heidelberg, 2003, pp. 225-249.

[11] U. dos S. Souza, A Parameterized Approach for And/Or Graphs and X-of-Y graphs. Master Thesis, Federal University of Rio de Janeiro, 2010.

[12] W. F. Tichy, A data model for programming support environments and its application, Proc. of the IFIP WG 8.1 Working Conference on Automated Tools for Information System Design and Development, New Orleans, North-Holland, pp. 31-48, Jan 1982.

Synthesizing Evidence on Risk Management: A Narrative Synthesis of two Mapping Studies

Luanna Lopes Lobato[1,2], Ivan do Carmo Machado[3], Paulo Anselmo da Mota Silveira Neto[1],
Eduardo Santana de Almeida[3], Silvio Romero de Lemos Meira[1]
[1]Informatics Center. Federal University of Pernambuco. Recife – PE. Brazil
[2]Computer Science Department. Federal University of Goiás. Catalão – GO. Brazil
[3]Computer Science Department. Federal University of Bahia. Salvador – BA. Brazil
{lll,pamsn,srlm}@cin.ufpe.br, {ivanmachado,esa}@dcc.ufba.br

Abstract—**Software Product Lines (SPL) Engineering is an effective development paradigm for systematic software reuse. It is focused on improving software practices, leading companies to experience a series of benefits, such as: reduction in both time-to-market and overall development effort, as well as improvements in terms of quality for the products delivered to customers. However, establishing an SPL is not a simple task, and may affect several aspects of an organization. Moreover, it indeed involves significant investment if compared to Single System Development (SSD). As a consequence, a greater set of risks is involved with SPL adoption, which can impact the project success if they are not well managed and appropriate RM activities are not applied. In this context, this paper presents an evidence analysis about RM in both fields, SPL and SSD. Outcomes from several published studies were analyzed, by means of systematic mapping studies, and hence compared in order to highlight the common and different findings among them. In addition, the studies were analyzed in order to present their contributions and the methods used to perform the research.**

Software Product Lines; Single System Development; Risk Management; Evidence-Based Software Engineering; Narrative Synthesis.

I. INTRODUCTION

SPL Engineering is based on a set of systems sharing a common, managed suite of features that satisfy the specific needs of a particular market or mission. Products in an SPL are developed from a common set of core assets in a prescribed way [1]. Such a development strategy, based on systematic software reuse, is aimed at achieving large-scale software production and customization, reduced time to market, improved quality and minimized costs [2].

Despite the benefits of the SPL adoption, challenges and problems may be faced, since it demands mature and systematic practices. The SPL development consists of a three-level process, namely Core Assets Development (CAD), Product Development (PD) and Management (M) [3]. In the prior, highly reusable core assets are developed, that will be assembled in PD to build the planned and expected products. Management is an orthogonal activity, performed in order to coordinate the whole SPL, thus encompassing both CAD and PD issues [3].

Establishing an SPL is not a simple task, which might affect several aspects of an organization. In this context, risk management policies should be set down, and communicated throughout the SPL development life cycle, since these can affect the project success in all of its phases [4].

Risk Management (RM) in SPL has gained attention from the research community, but it is still considered as an open field for investigation, especially in terms of moving research to industry practices, unlike traditional software development, i.e., single systems development (SSD), which contains a relevant set of reported evidences.

In this scenario, this investigation is aimed at discussing the RM field, on the basis of a comprehensive analysis of common issues reported for both SPL and SSD. We are interested in verifying likely gaps in the field of RM, in order to provide practitioners with a set of open rooms for improving the RM practices in SPL development.

Findings from two empirical studies [5] serve as a basis for this work. Such studies, performed as literature reviews on software risk approaches, in respectively SSD and SPL, sketched the state-of-the-art in RM in the both fields. They followed the systematic mapping study methodology [6]. An initial discussion on these findings was performed in a previous work [7], a short paper in which a comparison of outcomes is run. In such a study we described and synthesized the risks identified in the both literature reviews, by applying the *narrative synthesis* methodology.

In this work, an extension of [7], the synthesis was performed based on the guidelines provided by Rodgers et al. [8] and some experiences described by Cruzes and Dybå [9]. We also followed the lessons learned presented by Mills et al. [10] about the analysis of different syntheses.

The remainder of this paper is structured as follows. Section II describes the related work. Section III discusses the comparative analysis about the findings identified. In Section IV the limitations and threats are presented. Finally, Section V concludes this work and outlines future work.

II. RELATED WORK

Besides our initial comparative analysis [7], there are no studies that focus the comparison between RM to SPL and SSD. Hence, as far as we know, our research is the initial effort devoted reported towards understanding, based on comparing outcomes from such research fields, and performed using a systematic research strategy.

Cruzes and Dybå [9] proposed a tertiary study where 49 literature reviews were analyzed in order to verify the synthesis developed in these studies. Among the studies that presented synthesis, two thirds performed a narrative or a thematic synthesis. The focus of this study was to understand which are the challenges in synthesizing SE research and which are the implications for the progress of research and practice. They concluded that few researches has been conducted on studies synthesis in SE and highlighted the importance in comparing and contrasting evidence from more than one study during the synthesis.

Despite that in study an evidence analysis was developed, it differs of our work since we focused on analyze the evidence found on RM to SPL and SSD, in order to highlight the common and different findings.

III. COMPARATIVE ANALYSIS

We analyzed extracted data from 30 primary studies in the mapping study on RM in SPL (RM-SPL), and 74 studies in the mapping on RM in SSD (RM-SSD) [5]. Table I presents the number of studies from RM-SPL and RM-SSD, developed in either academic or industrial scenarios.

TABLE I. STUDY ENVIRONMENT

Studies	Academy	Industry	N/A
RM-SPL	2	27	1
RM-SSD	15	45	14

The number of studies reporting on industry practices was higher than academic investigations. In SSD, we identified a relevant number of studies that did not mention where the research was applied. Next we discuss the addressed development phases.

1) Development Phase

RM is a practice to be considered throughout the development life cycle. In addition, it is important to realize in which discipline the risk was leveraged. Both RM-SPL and RM-SSD considered all relevant disciplines [5], as follows (SPL - SSD): *Pre-Scoping* (8 - 0) ; *Scoping* (11 - 1); *Requirements* (8 - 8); *Architecture* (10 - 5); *Implementation* (0 - 1); *Test* (5 - 2); *Orthogonal* to more than one discipline (21 - 36); and *not available* (6 - 23).

It is worth mentioning that SPL and SSD can present different development phases. In SSD, the initial planning and scoping is embedded in the requirements discipline. However, as a way to compare the findings, the studies were classified in *Pre-Scoping* and *Scoping*, respectively. Thus, the basic disciplines were conserved following the SPL definition.

In RM-SPL, the total number of studies is higher than the number of primary studies selected, since one study could address more than one development phase. Such scenario represents the case where a study was not primarily focused on RM issues for SPL, but rather focused on a different SPL aspect and included some RM practices. This scenario was different in RM-SSD, in which the studies presented the specific disciplines where they were implemented. Unlike SPL, in SSD, RM is a common and well-explored activity. Hence it is possible to find several studies addressing specific software development lifecycle disciplines.

In RM-SPL, 11 studies addressed the *Scoping* discipline and 10 studies addressed the *Architecture* discipline. Since such disciplines are maybe the most important ones for SPL, and thus the SPL research community largely addresses them, our initial assumption was that such scenario, i.e., the number of studies we identified in the review on RM in SPL, would be portrayed in this investigation.

On the other hand, in RM-SSD, the most addressed disciplines were *Requirements* and *Architecture*. An important point to consider is that, despite the studies in SSD present more specific issues about RM, they did not present, in a clear way, in which disciplines the research was performed. There are represented by *Not available* values.

By *Orthogonal*, we mean the set of studies that do not fit any specific discipline, but rather gather information from a set of different disciplines without referring to a specific one.

2) Research Questions (RQ).

As a means to understand what has been studied in RM through the available literature about software development projects, we defined a number of questions that guided the data collection:

RQ1. Which risk management activities and practices are adopted by the approaches? This question aims to identify the RM activities and practices to deal with risks. These combine a set of activities performed in order to manage the risks.

RQ2. Which risk management steps are suggested by the approaches? The purpose is to identify the steps used to manage risks. These steps aim to make RM activities and practices easier, allowing managers to solve the risks.

RQ3. Which evaluation methods are presented by studies? The goal of this question is identify how the studies analyzed have been performed, if they are empirical or theoretical studies.

RQ4. Which contribution type has been presented by the studies? This questions aims to present the type of results developed by the primary studies.

Table II presents the number of the studies that answered each research question (RQ).

TABLE II. STUDIES PER RESEARCH QUESTIONS (RQ)

RQ	**RQ1**		**RQ2**		**RQ3**	
Answer	***SPL***	***SSD***	***SPL***	***SSD***	***SPL***	***SSD***
Yes	3	34	2	16	13	39
Partial	21	33	22	31	10	19
No	6	7	6	27	7	16

In RM-SPL, only 3 primary studies were classified with "Yes", regarding the activities and practices that could be applied to RM (RQ1). RM-SSD selected a greater number of studies in this sense. A similar scenario was observed in RQ2.

There are two possibilities that can be drawn from such scenario: *(1)* RM is not a commonly performed activity in an SPL project; or, *(2)* practitioners have not reported their experience about what has been experienced in RM to SPL projects. Such possibilities were confirmed through the studies analyzed during the mapping in SPL and SSD [5].

a) RM activities and steps.

A number of common and different activities were identified during the results analysis. In order to understand them, some steps were also considered, as showed in Table III. The use of steps may ease the activities execution. However, for some identified activities and practices, the steps were not mentioned.

TABLE III. ACTIVITIES AND STEPS

ID	Activities and Practices	SPL	SSD	Steps	SPL	SSD
Ac1	Mature Scope Definition	✓	✗	Identify the drawbacks in Scope	✓	✗
				Isolate stable areas from unstable	✓	✗
				Meetings with the experts	✓	✗
Ac2	Risk Identification	✓	✓	Apply Interview	✗	✓
				Apply Brainstorming	✗	✓
				Analyses Documentation	✗	✓
				Apply Questionnaire	✗	✓
				Continuous Identification	✗	✓
				Early Identification	✗	✓
				Observation and Expertise	✗	✓
				Provide Meetings	✗	✓
				Provide Risk Description	✗	✓
				Risk Scenario	✗	✓
				Use of Tool	✗	✓
Ac3	Risk Prioritization	✓	✓	Checklist	✗	✓
				Define Level of Exposure	✗	✓
				Rank the Risks	✗	✓
				Summarize the Risks	✗	✓
Ac4	Risk Reporting	✓	✗	----	✗	✗
Ac5	Team Commitment	✓	✓	Define Stakeholders Role	✓	✓
				Improve the Communication	✗	✓
				Provide Meetings	✗	✓
				Stakeholder´s Expertise	✗	✓
				Training the Team	✗	✓
Ac6	Artifacts Reuse	✓	✓	Identify reusable assets	✓	✗
				Define Level of Exposure	✗	✓
Ac7	Provide (SPL) Documentation	✓	✓	Define Graphs	✗	✓
				Define Risk Template	✗	✓
				Risks Description	✗	✓
				Risk Historic	✗	✓
				Risk Ranking	✗	✓
				Risk Scenario	✗	✓
Ac8	SPL Management	✓	✗	Software Product Line Mapping	✓	✗
				Artifacts Management	✓	✗
Ac9	Define Interview	✓	✓	Identify risk through interview	✓	✗
				Define the Questionnaire	✗	✓
Ac10	Mature Domain Definition	✓	✗	Preparation, Execution and Analysis	✓	✗
				Establish a domain analyst role	✓	✗
				Assess the subdomains	✓	✗
				Identification of reusable entities	✓	✗
				Make a domain potential analysis	✓	✗
Ac11	SPL Assessment	✓	✗	Assess the architecture	✓	✗
				Identify the strengths and weakness	✓	✗
Ac12	Feature Development	✓	✗	Features Identification	✓	✗
				Features Elicitation	✓	✗
				Feature Model Development	✓	✗
Ac13	SPL Variability	✓	✗	Variability Assessment	✓	✗
Ac14	Requirements Management	✓	✗	Requirements Elicitation	✓	✗
Ac15	Tool for SPL	✓	✗	----	✗	✗

Continued on next page

TABLE IV. CONTINUED FROM PREVIOUS PAGE

Ac16	Architecture Definition	✓	✗	Architecture Assessment	✓	✗
Ac17	SPL Tool	✓	✗	----	✓	✗
Ac18	SPL Testing	✓	✗	----	✓	✗
Ac19	Contingency Plan	✗	✓	Identify Contingency Factors	✗	✓
Ac20	Cooperative RM	✗	✓	----	✗	✗
Ac21	Define RM Plan	✗	✓	Action Plan	✗	✓
Ac22	Define Scenario	✗	✓	----	✗	✗
Ac23	Mitigation Plan	✗	✓	Anticipatory Mitigations	✗	✓
				Avoid or Prevent the Risk	✗	✓
				Define Level of Exposure	✗	✓
				Define Mitigation Plan	✗	✓
				Early Identification	✗	✓
				Risk Transfer	✗	✓
				Risk Reduction	✗	✓
				Risk Acceptance	✗	✓
Ac24	Provide Communication	✗	✓	Provide Meetings	✗	✓
Ac25	Provide Knowledge	✗	✓	----	✗	✗
Ac26	Risk Analysis	✗	✓	Analysis Effects and Causes	✗	✓
				Analysis Entire Project	✗	✓
				Analysis Interview	✗	✓
				Analyse the Documentation	✗	✓
				Define Graphs	✗	✓
				Decision Makers	✗	✓
				Define Level of Exposure	✗	✓
				Risk Scenario	✗	✓
				Rank the Risks	✗	✓
				Stakeholders Influence	✗	✓
Ac27	Risk Assessment	✗	✓	Apply a Questionnaire	✗	✓
				Define Level of Exposure	✗	✓
				Quantify the quality requirements	✗	✓
				Rank the Risks	✗	✓
				Risks Categories	✗	✓
				Training the Team	✗	✓
Ac28	Risk Avoidance	✗	✓	----	✗	✗
Ac29	Risk Classification	✗	✓	Effort Factors	✗	✓
				Managing Changes	✗	✓
				Organization factors	✗	✓
				Project factors	✗	✓
				Risk Archetypes	✗	✓
				Schedule Factors	✗	✓
				Team factors	✗	✓
				Technical factors	✗	✓
Ac30	*RM in Early Stage*	✗	✓	----	✗	✗
Ac31	*Risk Monitoring*	✗	✓	Define Stakeholders Roles	✗	✓
				Early Identification	✗	✓
				Provide Meetings	✗	✓
				Review Risk	✗	✓
				Risk Identification	✗	✓
				Tracking the Risk	✗	✓
Ac32	*Risk Pattern*	✗	✓	*Review Risk*	✗	✓
						Continued on next page

TABLE III. CONTINUED FROM PREVIOUS PAGE

Ac33	*Risk Profile*	✗	✓	Monitoring	✗	✓
				State Probabilities	✗	✓
Ac34	*Risk Resolution*	✗	✓	Define Resolution Plan	✗	✓
				Provide Inspection	✗	✓
				Risk Acceptance	✗	✓
				Risk Avoidance	✗	✓
				Risk Ignoring	✗	✓
				Risk Minimization	✗	✓
				Risk Transfer	✗	✓

TABLE IV. EVALUATION METHODS APPLIED BY THE PRIMARY STUDIES

SS Study	**Empirical Research**						**Theoretical Research**	
	Systematic Review	Case Study	Observational Study	Field Study	Experimental Study	Surveys	Theoretical Study	Expert Opinion
SPL	0	16	0	0	0	0	5	12
SSD	1	8	1	1	8	17	18	22

TABLE V. CONTRIBUTION TYPE OF THE STUDIES

Study	**Contribution Type**								
	Process	Framework	Method	Technique	Model	Approach	Tool	Characterization Scheme	Lessons Learned
RM-SPL	1	0	4	0	1	9	0	0	19
RM-SSD	1	8	2	1	11	9	1	1	40

It was noticed that a number of basic activities and practices were pointed in both studies, such as: *Risk Identification*, *Provide Documentation*, and *Define Interview*. In addition, there are specific activities that were mentioned in RM-SPL, due to their inherent characteristics: *Mature Scope Definition* and *Mature Domain Definition* – since in SPL the need for defining bo undaries is extremely relevant, and the core assets will be developed in compliance with these boundaries; *SPL Management* – management has to be systematic and well defined.

The activities and practices can serve as a guide for the Risk Manager to conduct RM, as well as to provide the research community with valuable insights.

b) *Evaluation Methods*

Regarding RQ3, through the analysis we could identify several research methods applied to conduct the primary studies and which are the contributions that these studies indicated to the area, as presented in the Table IV.

The categorization of methods was based on Lianping and Muhammad [11] and Montesi and Lago [12]. The methods were classified into Empirical and Theoretical Research, in order to facilitate the research type that has been performed to SPL and SSD. The number of primary studies presented in the table can be greater than the number of primary studies selected to each mapping studies, since one study can present more than one method.

Regarding SPL, almost the same number of studies was classified as Empirical Research and Theoretical Research divided in Case Studies, Theoretical Study and Expert Opinion. The same scenario was found in SSD, although other evaluation methods were also addressed. Several studies (in SPL and SSD) were only reports about expert's opinion. Also, as previously mentioned, many of the studies were related to experiences in the industry.

When it comes to the SPL scenario, it is important to highlight that the primary studies were not specific to RM, i.e., they did not present specific approaches or methods to conduct RM during the projects. However, they were included since they reported SPL studies discussing RM, at least superficially.

c) *Contibution type*

Through the RQ4, we could identify the type of contribution of the studies analyzed, which are the results presented, as showed in Table V. Many studies present lessons learned as results and the several studies were classified in Theoretical Research in the previous table.

We could observe that, either in SPL or SSD context, the primary studies do not bring clear evidence about the methods used and their contributions. Thus, the analysis of the contribution of each study was made based on our feelings during the development of the mapping studies, in which the primary studies were analyzed in a systematic way.

When it comes to the SPL scenario, it is important to highlight that the primary studies were not specific to RM, i.e., they did not present specific approaches or methods to conduct RM during the projects. However, they were included since they reported SPL studies discussing RM, at least superficially.

IV. LIMITATIONS AND THREATS

Some limitations and threats were identified during the development of this research, as follows. In order to reduce these problems, three researchers had been actively involved in this work.

The studies analyzed in the mapping studies lack sufficient information regarding ways that RM could be applied, mainly to SPL projects. Hence, to identify the results and report relevant findings the researcher involved in the analysis had to infer on the available results if these results were not explicitly presented as way to manage the risks.

A potential bias involves the fact that one researcher was responsible for synthetizing the evidences in this study, while different researchers were involved with the validation of the analysis.

The same researchers were involved in the both mapping studies execution, consequently, bias from others studies can be presented in this evidence analysis.

The RM-SPL presented studies until 2010 and the RM-SSD until 2011. This can be a threat since some studies could be considered in SSD due to the time in that the research was performed.

V. CONCLUSIONS

Despite the need to apply Risk Management (RM) activities during the development of SPL projects, these practices have not been adequately reported in the literature. Hence, explicit RM in SPL can still be considered as an open question, which may be confirmed by analyzing industrial practices, unlike SSD, which contains a large set of evidence.

In this effect, regarding SPL, RM still has a long way to develop research, since no empirical evidence is available about its effective use in companies that develop software based on product line paradigm. It was verified through the execution of two mapping studies in SPL and SSD to RM, where few studies reported experiences applying RM for SPL projects compared with SSD [5].

Based on this analysis, we identified a set of activities to apply RM in the SPL context. In addition, it was possible to understand the practices adopted by the primary studies for performing RM in software development. In the SPL context, we identified 32 risks, 15 different activities and practices that can be used to reduce them, and 26 steps to optimize the RM activities and practices execution. In the SSD review, we found 56 risks, 22 RM activities and practices and 85 steps.

The narrative synthesis added meaning and value to the results reported in our mapping studies, providing more detailed implications for further research about RM in SPL. The comparison described, through the narrative description, allowed us to investigate the scenario of RM and to collect specific insights to propose a clear RM approach to SPL.

REFERENCES

[1] Clements, P. and Northrop, L. M. Software Product Lines: Practices and Patterns. Boston MA U.S.A.: Addison-Wesley, Aug 2001

[2] Schmid, K. An Assessment Approach To Analyzing Benefits and Risks of Product Lines. Computer Software and Applications Conference, Annual International, pp. 525, 25th Annual International Computer Software and Applications Conference (COMPSAC'01).

[3] Northrop, L. M. SEI's Software Product Line Tenets. IEEE Softw. 19, 4 (July 2002), 32-40.

[4] Lobato, L. L., Silveira Neto, P. A. M., Machado, I. C., Almeida, E. S. and Meira, S. R. L. Risk Management in Software Product Lines: An Industrial Case Study. In: International Conference on Software and Systems Process (ICSSP), 2012, Zurich.

[5] Lobato, L. L. An approach for Risk Management in Software Product Lines. Ph.D. Thesis. Federal University of Pernambuco, Brazil, 2012, pp 382p.

[6] Petersen, K., Feldt, R., Mujtaba, S., Mattsson, M. Systematic mapping studies in software engineering, in: EASE '08: Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering, University of Bari, Italy.

[7] Lobato, L. L., Silveira Neto, P. A. M., Machado, I. C., Almeida, E. S. and Meira, S. R. L. An Study on Risk Management for Software Engineering. In: 16th International Conference on Evaluation & Assessment in Software Engineering (EASE), 2012, Ciudad Real, Spain.

[8] Rodgers, M., Sowden, A., Petticrew, M., Arai, L., Roberts, H., Britten, N., and Popay, J. Testing Methodological Guidance on the Conduct of Narrative Synthesis in Systematic Reviews, Evaluation, 15(1): 49–74.

[9] Cruzes, D. S. and Dybå, T. Research Synthesis in Software Engineering: A Tertiary Study, Information and Software Technology 53, 5 (May 2011), 440-455.

[10] Mills, A. J., Durepos, G. and Wiebe, E. Encyclopedia of case study research. SAGE Publications, Thousand Oaks.

[11] Lianping C. and Muhammad A. B. A systematic review of evaluation of variability management approaches in software product lines. Inf. Softw. Technol. 53, 4 (April 2011), 344-362.

[12] Montesi, M. and Lago, P. Software engineering article types: An analysis of the literature. Journal of Systems and Software. 81, 10 (October 2008), 1694-1714.

PlugSPL: An Automated Environment for Supporting Plugin-based Software Product Lines

Elder M. Rodrigues*, Avelino F. Zorzo*, Edson A. Oliveira Junior†, Itana M. S. Gimenes†,
José C. Maldonado‡ and Anderson R. P. Domingues*
*Faculty of Informatics (FACIN) - PUCRS - Porto Alegre-RS, Brazil
Email: {elder.rodrigues, avelino.zorzo}@pucrs.br
Email: anderson.domingues@acad.pucrs.br
†Informatics Department (DIN) - UEM - Maringá-PR, Brazil
Email: {edson, itana}@din.uem.br
‡Computing Systems Department (ICMC) - USP - São Carlos-SP, Brazil
Email: jcmaldon@icmc.usp.br

***Abstract*—Plugin development techniques and the software product line (SPL) approach have been combined to improve software reuse and effectively generate products. However, there is a lack of tools supporting the overall SPL process. Therefore, this paper presents an automated environment, called PlugSPL, for supporting plugin-based SPLs. Such environment is composed of three modules: SPL Design, Product Configuration, and Product Generation. An example of a PlugSPL application is illustrated by means of the PLeTs SPL for the Model-Based Testing domain. The environment contributions are discussed whereas future work is listed.**

***Keywords*-Software Product Lines, Plugin-based SPL, Model-based Testing.**

I. Introduction

In recent years, software product line (SPL) [1] engineering has emerged as a promising reusability approach, which brings out some important benefits, e.g., it increases the reusability of its core assets, in the meanwhile decreases time to market. The SPL approach focuses mainly on a two-life-cycle model [1]: domain engineering, in which the SPL core assets are developed for reuse; and application engineering, in which the core assets are reused to generate specific products. It is important to highlight that the success of the SPL approach depends on several principles, in particular variability management [2].

Although SPL engineering brings out important benefits, it is clear the lack of an environment aimed at automating the overall SPL life cycle, including: (i) configuration of feature model (FM); (ii) configuration of products; and (iii) generation of products. Literature and industry present several important tools that encompass part of the SPL development life cycle, e.g., SPLOT [3].

The plugin approach has also received an increasing attention in the development of SPLs [4]. In the SPL field, a plugin-based approach enables the development of different applications by selecting/developing different sets of plugins. Although the use of plugins to develop SPL products is a promising approach and several works have been published in recent years, to the best of our knowledge, there is no tool to support plugin-based SPLs. Therefore, this paper presents an automated environment for supporting the overall SPL engineering life cycle, the PlugSPL. Such an environment differs from existing tools as it aims at supporting plugin-based SPLs. Moreover, PlugSPL provides capabilities both to import/export FMs from/to other tools and to effectively generate products.

This paper is organized as follow. Section II discusses some concepts of SPL and plugin-based SPLs; Section III presents the PlugSPL environment and its main characteristics; Section IV illustrates the use of PlugSPL to manage a Model-Based Testing (MBT) SPL; and, Section V presents the conclusion and directions for future work.

II. Background

In recent years, the plugin concept has emerged as an interesting alternative for reusing software artifacts *de facto* [5]. Moreover, plugins are a useful way to develop applications in which functionalities must be extended at runtime.

In order to take advantage of the plugin concept for developing software, it is necessary to design and implement a system as a core application that can be extended with features implemented as software components. A successful example of the plugin approach is the Eclipse platform [6], which is composed of several projects in which plugins are developed and incorporated to improve both the platform and the providing services.

The plugin approach has also received an increasing attention in the development of SPLs [4]. The SPL approach has emerged over the last years due to competitiveness in the software development segment. The economic considerations of software companies, such as cost and time to market, motivate the transition from single-product development to the SPL approach, in which products are developed in a large-scale reuse perspective [1]. Whereas a SPL can be defined as a set of applications that share a common set of features and are developed based on a common set of core assets, the plugin approach can be easily applied to build new applications by plugging different sets of plugins to a core application [7]. Although the use of plugins to develop products is a promising approach and several works have been published in recent years, there is no tool to support plugin-based SPLs.

III. PlugSPL Environment: Supporting Plugin-based Software Product Lines

Although there are many tools focused on SPL modeling, consistence checking [3], and product generation support [8], currently, there is no tool that integrates all SPL development phases. Moreover, there is no tool to support the automated product configuration and product generation from a plugin-based SPL. Therefore, in this section we present the PlugSPL environment that has been developed to support SPL design, product configuration and generation of plugin-based SPLs. Figure 1 presents the PlugSPL modules and activities, as follows:

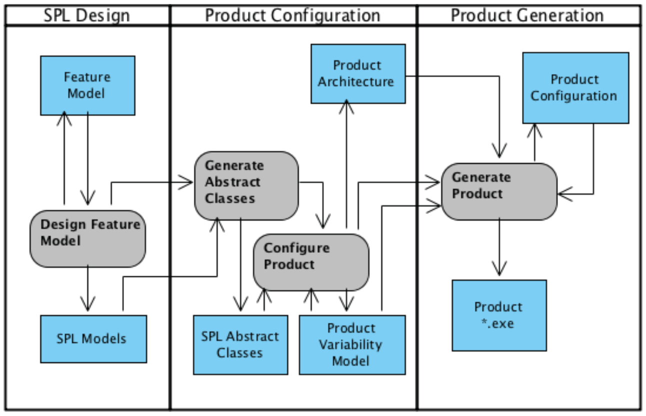


Figure 1: The PlugSPL modules.

a) the SPL Design module aims to design a FM by either creating it from scratch or importing a pre-existing FM from SPL tools. Such tools, usually, do not use a common format to represent FM elements and constraints, therefore, we conceived the PlugSPL SPL Design module to work with a wide FM representations and file formats. Thus, PlugSPL FMs can be seen as a starting point to automate the creation of the SPL architecture and then to generate products. Based on information extracted from a FM, the SPL Design module represents such information as SPL Models (Figure 1). Such models are taken as input to the Product Configuration module for composing the SPL architecture;

b) the Product Configuration module is responsible for automating the SPL architecture. Basically, this module has two activities - Generate Abstract Classes and Configure Product. The former receives the SPL Models provided by the SPL Design module and creates a set of abstract classes, one class for each feature. Each abstract class is a variation point and/or a variant with a specific type. Thus, each plugin may extend only one abstract class. The abstract classes might be used, according to the SPL documentation, by the core assets developer to build each plugin that composes the SPL. After the generation of the abstract classes, the SPL engineer is able to select the desired features for the target system (product configuration). PlugSPL checks the system consistency and generates the target system architecture (abstract classes).

c) the Product Generation module takes the target system architecture as input. This module graphically shows such architecture to the application engineer. Thus, this module retrieves from the plugin repository respective plugins that implement the types of the abstract classes (product architecture). After that, the plugins are linked to each of their respective abstract classes and the consistency between the generated architecture and FM is checked. PlugSPL shows graphically the set of plugins that is able to be selected for resolving the variability in each class and generates the target system. Them, the application engineer: (i) selects one or more plugins (which denote a feature in the FM) to resolve each class variability; (ii) gives a name to the target system; and (iii) clicks a button to generate the system.

IV. PlugSPL Application Example

This section presents an example of how PlugSPL can be used to design, develop and derive MBT products from PLeTs SPL [7]. PLeTs is a SPL aimed at automating the generation, execution and results collection of MBT processes. The MBT process consists in the generation of test cases and/or test scripts based on the application model. The MBT process main activities are [10]: *Build Model*, *Generate Expected Inputs*, *Generate Expected Outputs*, *Run Tests*, *Compare Results*, *Decide Further Actions* and *Stop Testing*. PLeTs goal is the reuse of SPL artifacts to make it easier and faster to develop a new MBT tool. Figure 2 shows the main features of the current PLeTs FM: Parser, TestCaseGenerator,ScriptGenerator and Executor.

Figure 2 shows several dependencies (denoted by propositional logic) between features. For instance, if feature Executor and its child feature LoadRunnerParameters are selected, then feature ScriptGenerator and its child feature LoadRunnerScript must be selected as the generated tool is not able to execute tests without test scripts.

The PLeTs FM can be extended to support new testing techniques or tools by adding new child features to its main features. For instance, if one adds new features for the SilkPerformer testing tool, new child features for the ScriptGenerator and Parameterization features must be included.

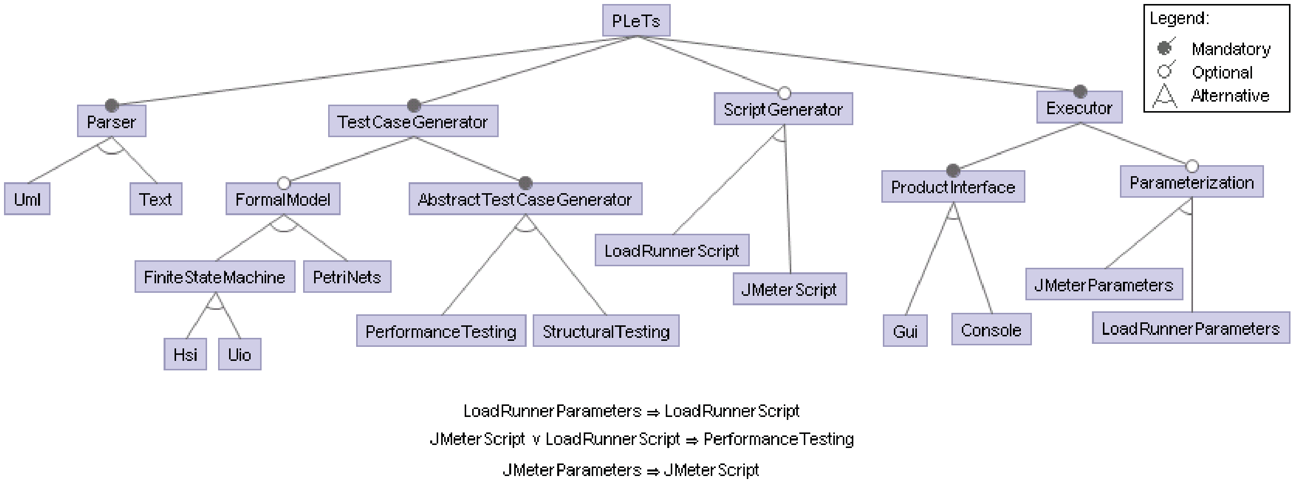


Figure 2: The PLeTs Feature Model [7].

A. *Using PlugSPL for Generating a MBT SPL*

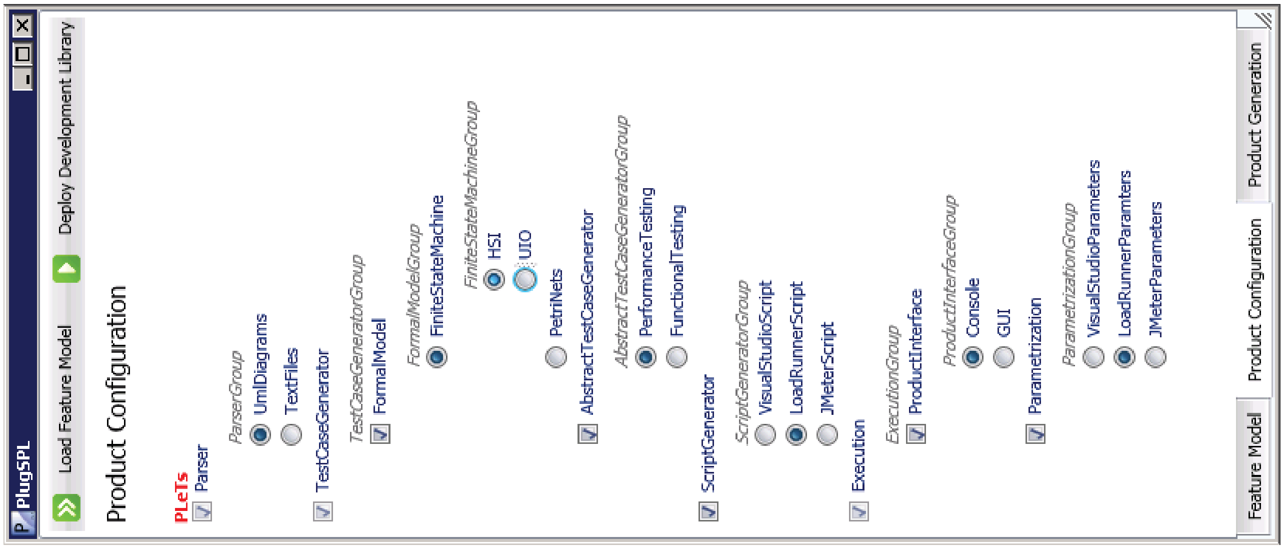
Designing and development of SPLs supported only by FMs editors and SPL documentation itself might be error prone and time consuming activities. Moreover, checking features constraints manually is a hard task. Therefore, this section explains how PlugSPL can be used to automate the overall SPL process by using PLeTs as an example SPL.

PlugSPL imports PLeTs FM (Figure 2) to the `SPL Design` module. Thus, PlugSPL re-constructs and checks the FM and shows the result to the SPL architect using a tree notation. Therefore, the SPL architect can interact with the PLeTs FM to, for instance, add or edit feature relationships. The PLeTs FM is saved as SPL Models to support the Product Configuration phase.

During Product Configuration, PlugSPL generates the PLeTs architecture, formed by its abstract classes, e.g., Parser, UmlDiagrams and ScriptGenerator. Thus, the SPL architect might export such classes, by clicking on the `Deploy Development Library` button, and send it to the plugin development team. Based on the architecture, such team develops a plugin by extending a specific abstract class (e.g. Parser), and sends it back to the SPL architect to store it in the SPL plugin repository. As shown in Figure 3, the product architect might define each product architecture by selecting the abstract classes in the tree structure. A product is a performance MBT tool that realizes the following activities; a) Accepts as input an UML model (Parser, UmlDiagrams); b) Transforms the model in a formal model (TestCaseGenerator, FiniteStateMachine), and applies it a sequence method (HSI) to generate the testing sequence. Based on such a testing sequence, it generates the abstract performance test cases (AbstractTestCaseGenerator, PerformanceTesting); c) Uses the abstract performance test cases to generate scripts to the LoadRunner performance tool (ScriptGenerator, LoadRunnerScript); d) Executes product via command line (Execution, Console) and sets the LoadRunner parameters (Parameterization, LoadRunnerParameters).

PlugSPL allows the product architect to save each product architecture in a repository for reuse. It is important to highlight that in the PLeTs SPL each abstract class is a variation point that is resolved by selecting a variant (plugin).

The Product Generation module presents graphically the abstract classes structure of a product. It also links a plugin to classes by performing a search in the plugin repository to find what plugins are implemented by each product abstract class. Thus, the product architect selects a plugin, or a set of plugins, to resolve each variability. In the Product Generation activity each variability (abstract class) uses only one variant (plugin) to resolve a variability. However, it might be necessary to use two or more variants to resolve a variability. After resolving a variability, the product architect has two options: save the product configuration and/or generate the MBT product. In the first option, the tool asks for a product name and, then, saves its classes and plugins to generate a product later. In the second option, PlugSPL asks for a product name, and then generates the MBT product. In order to generate the product, PlugSPL: (i) selects the product abstract classes and its related plugins; (ii) packages them by using a "glue code"; and (iii) generates an executable product file. Although the "glue code" generation is only invoked and executed by PlugSPL, the piece of code that generates such a code is implemented in the SPL base plugin. This approach simplifies the development and evolution of the product as the complex information necessary to generate products from a wide amount of SPLs is stored as a SPL artifact.



Product Configuration

- PLeTs
  - ☑ Parser
    - ParserGroup
      - ● UmlDiagrams
      - ○ TextFiles
  - ☑ TestCaseGenerator
    - TestCaseGeneratorGroup
      - ☑ FormalModel
        - FormalModelGroup
          - ● FiniteStateMachine
            - FiniteStateMachineGroup
              - ● HSI
              - ○ UIO
          - ○ PetriNets
      - ☑ AbstractTestCaseGenerator
        - AbstractTestCaseGeneratorGroup
          - ● PerformanceTesting
          - ○ FunctionalTesting
  - ☑ ScriptGenerator
    - ScriptGeneratorGroup
      - ○ VisualStudioScript
      - ● LoadRunnerScript
      - ○ JMeterScript
  - ☑ Execution
    - ExecutionGroup
      - ☑ ProductInterface
        - ProductInterfaceGroup
          - ● Console
          - ○ GUI
      - ☑ Parametrization
        - ParametrizationGroup
          - ○ VisualStudioParameters
          - ● LoadRunnerParamters
          - ○ JMeterParameters

Figure 3: PlugSPL Feature Model Editing and Product Configuration.

V. Conclusion and Future Work

This paper presented PlugSPL, which is an automated environment to support the overall plugin-based SPL life cycle. Although there are tools to partially support the SPL life cycle as, for instance, `pure::variants`, there is no tool that supports plugin-based SPLs and the overall SPL life cycle. Furthermore, there are tools to design FMs, but most of them use different notations and file formats. PlugSPL provides capabilities with regard to create or import/export FMs from/to other tools and uses a wide file format. Therefore, there is no need to incorporate other tools/environments into PlugSPL.

Although PlugSPL is a flexible environment for modeling FMs, its most significant benefit is supporting the generation of SPL products based on its FM. Moreover, PlugSPL automatically generates an abstract class structure, which can be used to develop third-party plugins. A PlugSPL application example was presented for deriving MBT tools from the PLeTs SPL. Directions for future work are: (i) plan and conduct experiments for assuring the effectiveness of PlugSPL environment; (ii) extend PlugSPL functionalities to support different plugin-based SPLs; and (iii) include into PlugSPL an overall SPL evaluation module.

References

[1] 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.

[2] E. A. Oliveira Junior, I. M. S. Gimenes, and J. C. Maldonado, “Systematic Management of Variability in UML-based Software Product Lines,” *Journal of Universal Computer Science*, vol. 16, no. 17, pp. 2374–2393, 2010.

[3] M. Mendonça, M. Branco, and D. Cowan, “S.P.L.O.T.: Software Product Lines Online Tools,” in *Proc. Conf. Object Oriented Programming, Systems, Languages, and Applications*. New York, NY, USA: ACM, 2009, pp. 761–762.

[4] R. Wolfinger, S. Reiter, D. Dhungana, P. Grunbacher, and H. Prahofer, “Supporting Runtime System Adaptation through Product Line Engineering and Plug-in Techniques,” *Int. Conf. Commercial-off-the-Shelf (COTS)-Based Software Systems*, pp. 21–30, 2008.

[5] J. Mayer, I. Melzer, and F. Schweiggert, “Lightweight Plug-In-Based Application Development,” in *Proc. Int. Conf. NetObjectDays on Objects, Components, Architectures, Services, and Applications for a Networked World*. London, UK, UK: Springer-Verlag, 2003, pp. 87–102.

[6] M. Kempf, R. Kleeb, M. Klenk, and P. Sommerlad, “Cross Language Refactoring for Eclipse Plug-ins,” in *Proc. Workshop on Refactoring Tools*. New York, NY, USA: ACM, 2008, pp. 1–4.

[7] M. B. Silveira, E. M. Rodrigues, A. F. Zorzo, L. T. Costa, H. V. Vieira, and F. M. de Oliveira, “Model-Based Automatic Generation of Performance Test Scripts,” in *Proc. Software Engineering and Knowledge Engineering Conf.* Miami, USA: IEEE Computer Society, 2011, pp. 258–263.

[8] D. Beuche, “Modeling and Building Software Product Lines with Pure::Variants,” in *Proc. Int. Software Product Line Conf.* New York, NY, USA: ACM, 2011, pp. 358–.

[9] T. Thum, C. Kastner, S. Erdweg, and N. Siegmund, “Abstract Features in Feature Modeling,” in *Int. Conf. Software Product Line*, 2011, pp. 191–200.

[10] I. K. El-Far and J. A. Whittaker, *Model-based Software Testing*. New York: Wiley, 2001.

GS2SPL: Goals and Scenarios to Software Product Lines

Gabriela Guedes, Carla Silva, Jaelson Castro, Monique Soares, Diego Dermeval, Cleice Souza
Centro de Informática
Universidade Federal de Pernambuco/UFPE
Recife, Brazil
(ggs, ctlls, jbc, mcs4, ddmcm, ctns)@cin.ufpe.br

Abstract — **GORE (Goal Oriented Requirements Engineering) approaches can effectively capture both the stakeholders' objectives and the system requirements. In the context of Software Product Lines (SPL), they offer a natural way to capture similarities and the variability of a product family. Goals to Software Product Lines (G2SPL) is an approach that systematically guides the creation of an SPL feature model from i* models with cardinality. However, it is not possible to model behavioral characteristics of an SPL through GORE approaches, such as i*. In order to capture the system behavior, it is common to use a scenario specification technique. This paper defines a Requirements Engineering approach for SPL that integrates the G2SPL approach and a technique to specify use case scenarios with variability. This new approach is named GS2SPL (Goals and Scenarios to Software Product Lines) and also includes a sub-process for configuring specific applications of an SPL based on the priority given to non-functional requirements.**

Keywords - Requirements Engineering; Software Product Line; Goal Modeling; Feature Model; Scenarios

I. INTRODUCTION

Requirements Engineering (RE) is the phase of software development concerned with producing a set of specifications of software characteristics that satisfy the *stakeholders* needs and can be implemented, deployed and maintained [1].

In RE for Software Product Lines (SPL), feature models (FM) are often used to capture similarities and the variability of a product family. However, using only FM, it is difficult to relate the features of a software product and the objectives of the stakeholders [2]. Besides, there wasn't a systematic way to select features for a particular product. In this context, in [3] we proposed a systematic approach to obtain features of SPL from stakeholders' goals and to select the features for a specific product of an SPL, based on non-functional requirements (NFR) analysis.

However, our previous approach didn't use scenario-based descriptions (e.g. use cases) that are easily understood by stakeholders [4] and capable of capturing the dynamic or behavioral aspect of the SPL.

Hence, in this paper, we propose the GS2SPL (Goals and Scenarios in Software Product Lines) approach, which supports the generation of feature models and scenario-based descriptions from goal-based requirements specification. Besides, this approach takes into account the priority given to the NFRs used in its configuration sub-process to select the features of a specific product that best satisfies the stakeholders' needs.

This paper is organized as follows. Section II describes the background required for a better understanding of this work. Section III presents the GS2SPL approach. Section IV discusses related work and Section V su mmarizes our contributions and points out future work.

II. BACKGROUND

A. Software Product Line Engineering

Clements and Northrop [5] define an SPL 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 prescribed way. The term "software product family" is also used as synonym to "software product line" [6].

According to Pohl et al. [6], Software Product Line Engineering (SPLE) is a paradigm to develop software systems using platforms and m ass customization. The same authors proposed an SPLE framework that consists of two separate processes:

- Domain Engineering: responsible for establishing the reusable platform and defining the commonality and the variability of the product line [6]. This process is called Core Asset Development in [5];
- Application Engineering: responsible for deriving product line applications from the platform established in domain engineering [6]. This process is called Product Development in [5].

Our approach covers both processes, but it is limited to the requirements phase only.

B. iStar

In the i* framework [7], stakeholders are represented as actors that depend on each other to achieve their goals, perform tasks and provide resources. Each goal is analyzed from its actor point of view, resulting in a set of dependencies between pairs of actors. The Strategic Dependency (SD) model provides a description of the relationships and external dependencies among organizational actors. The Strategic Rationale (SR) model enables an analysis of how goals can be fulfilled through contributions from the several actors. Fig. 1 presents part of the SR model for MobileMedia [8], an SPL that will be used in this paper as a running example. The main purpose of MobileMedia is to manage media files in mobile devices.

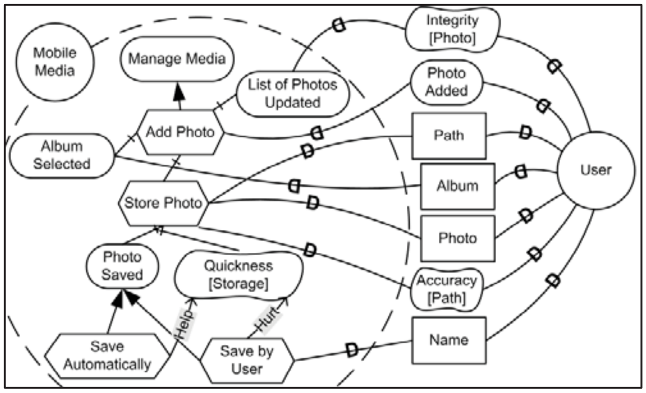


Figure 1. MobileMedia i* SR model

The SR model in Fig 1 depicts two actors ("MobileMedia" and "User") and their dependencies. "User" depends on "MobileMedia" to achieve the "Photo Added" goal and to obtain "Album" resource. It also expects that "MobileMedia" contributes to satisfy the "Integrity [Photo]" softgoal. On the other hand, "MobileMedia" depends on "User" to obtain "Path", "Photo" and "Name" resources and it expects "User" to contribute to the "Accuracy [Path]" softgoal.

The boundary of the "MobileMedia" actor, shown in Fig. 1, provides information on how this actor can fulfill its dependencies and why it depends on other actors. The "Add Photo" task is a way to satisfy the "Manage Media" goal, thus these elements are linked by a means-end relationship. The "MobileMedia" will fulfill the "Photo Added" dependency through "Add Photo" task, which is decomposed in "Album Selected" and "List of Photos Updated" goals and the "Store Photo" task. "Store Photo" is decomposed in the "Quickness [Storage]" softgoal and the "Photo Saved" goal. The "Photo Saved" goal can be satisfied by "Save Automatically" task, which contributes positively ("help" contribution link) to satisfy the "Quickness [Storage]" softgoal, or by "Save by User" task that contributes negatively ("hurt" contribution link) to the satisfaction of the same softgoal.

To capture variability in SR models of SPLs, an extended version of i* has to be used. The i*-c (i* with cardinality) allows the insertion of cardinality in task and resource elements and also in means-end relationships [2]. This i* extension is used by G2SPL [3] approach to derivate feature models. Our approach is an extension of the G2SPL and, hence, it uses i*-c models to derive not only FMs, but use case scenarios as well.

C. *Scenarios for SPL*

PLUSS (Product Line Use case modeling for Systems and Software engineering) [9] is an SPL approach that combines feature models and use case scenarios. It captures both common and variable behavior of the SPL. In PLUSS, both use cases and scenario steps are annotated with the features to which they are related. During product configuration, desired features are selected in the FM, and their corresponding annotations, present on the use case scenarios, are used to configure these scenarios for a specific product. Thus, only use cases and scenario steps annotated with the selected features will be present on the product's use case descriptions.

III. GS2SPL

GS2SPL is an RE approach for SPL in which the feature model and the use case scenarios of an SPL are obtained from i*-c goal models. The GS2SPL process is divided in eight activities, mostly are part of the Domain Engineering process and just the last one is part of Application Engineering process. The first four activities were inherited from G2SPL and, therefore, they will not be explained in details in this paper. Additional information about these activities can be found in [3]. The rest of the process consists on the addition of new activities or adaptations of G2SPL activities. The next sub-sections present all activities of GS2SPL:

A. *Creation of SR Model*

This activity consists of modeling the stakeholders' goals using the i* framework. It is considered an optional activity if the SR model is already available. The output of this activity, for the running example, is depicted in Fig.1.

B. *Identification of Candidate Elements to be Features*

In this activity, the domain engineer identifies the elements of the SR Model that could represent features. Features may be extracted from tasks and resources. Therefore, all internal tasks and resources of the actor that represents the SPL should be highlighted, as well as task and resource dependencies connected to this actor.

C. *Reengineering the SR Model*

In this activity, cardinality is added to the SR model based on some heuristics defined in the G2SPL approach [3]. In summary, cardinality may be added to intentional elements and to means-end relationships in which the root element (end) has more than one sub-element (means). The output of this activity is an SR Model with cardinality, as the one shown in Fig. 2.

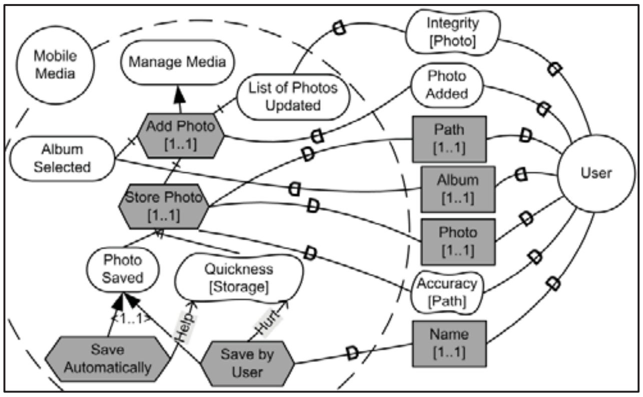


Figure 2. MobileMedia i*-c SR model with feature candidates highlighted in grey

D. *Elaboration of the Feature Model*

This activity is concerned with the derivation of the SPL feature model (activity output). The input artifacts are some heuristics to elaborate the FM and the SR model with cardinality. The heuristics suggests the construction of a table (Table I) that keeps the traceability between features and tasks/resources. This table is used to obtain the feature model.

TABLE I. TABLE OF TRACEABILITY BETWEEN FEATURES AND TASKS/RESOURCES

Element	**Cardinality**		**Parent Element**	**Feature**
	Type	***Value***		
Add Photo	Element	[1..1]	–	Add Photo
Album	Element	[1..1]	Add Photo	Album
Store Photo	Element	[1..1]	Add Photo	Store Photo
Save Autom.	Group	<1..1>	Store Photo	Save Autom.
Save by User	Group	<1..1>	Store Photo	Save by User
Path	Element	[1..1]	Store Photo	Path
Photo	Element	[1..1]	Store Photo	Photo
Name	Element	[1..1]	Save by User	Name

According to the heuristics defined in this activity, optional features are obtained from elements with cardinality [0..1], while mandatory features are obtained from elements with cardinality [1..1]. Elements involved in a means-end relationship with cardinality become alternative features with equivalent cardinality. The FM obtained from the SR model depicted in Fig. 2 is shown in Fig. 3.

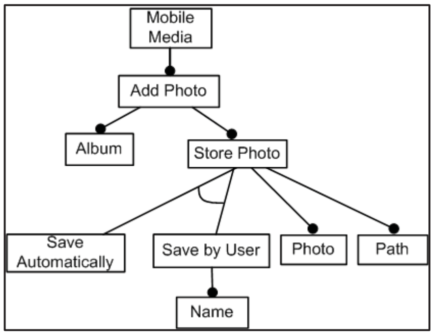


Figure 3. Feature model of MobileMedia

E. Feature Model Refinement

This is an optional activity to be executed if the FM needs to be reorganized or if new features must be add ed, because they were not present in the SR model. Reorganization is required if the FM has repeated features, sub-features with more than one parent or different features with the same meaning. This activity can be performed as many times as the domain engineer believes it is necessary. Our running example is quite simple and did not require the execution of this activity.

F. Elaboration of Use Case Scenarios

The SPL use case scenarios are specified based on an adaptation of the guidelines defined by Castro et al. [10]. This activity obtains the PLUSS scenarios description for an SPL from its SR model and feature model. The guidelines proposed by Castro et al. in [10] are a mapping between i* models and use case scenarios that are not specific for dealing with SPL variability. We propose guidelines to map i*-c models to PLUSS use case scenarios. As a result, some of the 10 previous guidelines were removed; others were split into new ones.

In Step 1, all four guidelines were maintained, but a new one was inserted (current Guideline 4) to deal with actors that have only softgoal dependencies with the SPL actor. In Step 2, the two previous guidelines were merged, since their sub-guidelines were exactly the same. In Step 3, (i) a guideline was inserted (current Guideline 7) to address PLUSS annotations; (ii) the previous Guideline 8 was split into four (current Guidelines 8, 9, 10, 11) because of the necessity to deal with every possible cardinality in i*-c; (iii) the previous Guideline 9 (current Guideline 12) was reformulated to take into account use cases derived from optional steps; and (iv) previous Guideline 10 was removed because it was a recommendation to draw a u se case diagram, but not a mapping guideline. The guidelines we propose in GS2SPL approach are presented as follows:

Step 1 – Discovering actors:

- Guideline 1: Every i* actor is a candidate to be mapped to a use case actor;
- Guideline 2: The candidate i* actor should be external to the intended software system; otherwise, it cannot be mapped to a use case actor;
- Guideline 3: The candidate i* actor should have at least one dependency with the actor representing the SPL; otherwise, it cannot be mapped to a use case actor;
- Guideline 4: Analyze the dependencies between the candidate i* actor and the actor that represents the SPL. If all of them are s oftgoal dependencies, the i* actor cannot be mapped to a use case actor;
- Guideline 5: Actors in i*, related through the ISA relationship, and mapped individually to use case actors (applying guidelines 1, 2, 3 and 4), will be related through the "generalization" relationship in the use case diagram;

In our example, there is only one external actor, "User", and, according to the presented guidelines, it can be mapped to a use case actor.

Step 2 – Discovering use cases for the actors:

- Guideline 6: For each use case actor discovered in Step 1, analyze its dependencies with the SPL actor;
  - Guideline 6.1: Goal dependencies – goals in i* can be mapped to use cases;
  - Guideline 6.2: Task dependencies – it should be investigated if the task needs to be decomposed into sub-tasks. If the task requires many steps to be executed, then it can be mapped to a use case;
  - Guideline 6.3: Resource dependencies – it should be investigated if the resource can only be obtained after many interaction steps between the discovered actor and the system-to-be. If so, the dependency can be mapped to a use case;

  - Guideline 6.4: Softgoal dependencies – typically, the softgoal dependency in i* is an NFR for the intended system. Hence, a softgoal does not represent a use case, but an NFR associated with a specific use case or with the SPL as a whole;

After applying Step 2 to the example, we discovered that only the "Photo Added" goal dependency can be mapped to a use case.

Step 3 – Discovering and describing use case scenarios:

- Guideline 7: After discovering the use cases, the field "Feature" in their description (Table II) should be filled. Also, the scenario steps to be discovered by using the next guidelines should be annotated with the features related to them;
  - Guideline 7.1: If a task or resource dependency originated the use case, then the "Feature" field is filled with the name of the feature related to the task/resource on the table of traceability (Table I);
  - Guideline 7.2: If a goal dependency originated the use case, then it is necessary to analyze the internal element to which the dependency is connected in the SPL actor. If such element is a task or resource, the "Feature" field is filled with the name of the feature related to that task/resource. If the element is a goal, then the "Feature" field will be filled with the name(s) of the feature(s) related to the task(s) in which this goal is refined .
  - Guideline 7.3: The steps obtained from tasks or resources should be annotated with the name of the related feature between brackets – [];
- Guideline 8: Analyze sub-elements of task decompositions in order to map them to mandatory steps of the use case primary scenario;
  - Guideline 8.1: If the decomposed task in analysis satisfies a dependency that was mapped to a use case, then the sub-elements of that decomposition will be mapped to mandatory steps of such use case;
  - Guideline 8.2: Every step obtained from task decomposition sub-elements is identified by a unique number without parenthesis, according to PLUSS notation for mandatory steps;
  - Guideline 8.3: If a sub-element of a task decomposition is a softgoal, then it is associated to the use case as an NFR;
- Guideline 9: When a means-end relationship has only one sub-element, the sub-element's cardinality should be analyzed;
  - Guideline 9.1: If the cardinality is [1..1], the element will be mapped to a mandatory step. Hence, it will be identified by a unique number without parenthesis;
  - Guideline 9.2: If the cardinality is [0..1], the element will be mapped to an optional step. Hence, it will be identified by a unique number between parenthesis, according to the PLUSS notation for optional steps;
- Guideline 10: When a means-end relationship has more than one sub-element, the relationship cardinality should be analyzed;
  - Guideline 10.1: If the cardinality is <1..1>, the sub-elements will be mapped to mutually exclusive alternative steps. Therefore, they will be identified by equal numbers without parenthesis;
  - Guideline 10.2: If the cardinality is <0..1>, the sub-elements will be mapped to alternative steps where only one may be selected. Therefore, they will be identified by equal numbers between parenthesis;
  - Guideline 10.3: If the cardinality is <0..*n*> and n>1, the sub-elements will be mapped to alternative steps where none or at most *n* can be selected. Therefore, they will be identified by equal numbers between parenthesis, followed by a different letter for each alternative;
  - Guideline 10.4: If the cardinality is <*i*..*j*>, $i \neq 0$ and $j>1$, the sub-elements will be mapped to alternative steps where at least *i* and at most *j* can be selected. Therefore, they will be identified by equal numbers without parenthesis, followed by a different letter for each alternative;
  - Guideline 10.5: If there is cardinality only in the sub-elements, but not in the relationship, the sub-elements will be mapped to alternative steps and some of them may be optional, according to the cardinality. Therefore, they will be identified by equal numbers, followed by a different letter for each alternative. However, those with [0..1] cardinality should have their identifications between parenthesis;
- Guideline 11: The SR model should be analyzed to discover additional information about the use cases;
  - Guideline 11.1: Analyze contribution links from other elements (source) to softgoals (target). If the source element is part of a use case, then the target softgoal will be associated to this use case as a non-functional requirement;
  - Guideline 11.2: Analyze links between elements that generated steps to different use cases. These links may represent that an element is a pre-condition of a use case (obtained from the other element) or a relationship, such as "include" or "extend", between use cases (obtained from the related elements);
- Guideline 12: Analyze use cases to check if they can be refined or generate new use cases. Each scenario step

should be verified for the possibility of deriving a new use case;

- o Guideline 12.1: A new use case will be generated if a step represents an activity that requires several steps to be concluded;
- o Guideline 12.2: If the new use case was derived from a mandatory step, then it must be related to the original use case through the "include" relationship;
- o Guideline 12.3: If the new use case was derived from an optional step, then it must be related to the original use case through the "extend" relationship.

Applying Step 3 to the MobileMedia example, we obtained the description for the "Add Photo" use case (Table II).

TABLE II. "ADD PHOTO" USE CASE DESCRIPTION

Use Case 1: Add Photo

CHARACTERISTIC INFORMATION

Primary Actor: User
Feature: Add Photo
Scope: MobileMedia
Pre-condition: -
Success Condition: Photo added to album

PRIMARY SCENARIO

ID	User Action	System Response
1	Select "Add Photo" option [Add Photo]	
2	Select album [Album]	
3	Provide path of photo [Path]	
4	Select photo to be added [Photo]	
5	-	Photo is automatically saved [Save Automatically]
5	Choose for photo [Name] [Save by User]	Photo is saved with the chosen name
6	-	List of photos is updated

SECONDARY SCENARIOS

RELATED INFORMATION

Non-functional requirements: Integrity [Photo], Accuracy [Path], Quickness [Storage]

G. Use Case Scenarios Refinement

As it can be observed in Table II, the use case description generated by the guidelines proposed in the previous activity is not complete. The system response for some actions is missing, as well as the secondary scenarios (exceptional and alternative scenarios). This occurs because i* models are not meant for modeling behavior or exceptions. Therefore, this activity is required to complement the use case description.

The scenarios obtained on the previous activity may be succinct or written on a very high level; it will depend on the level of refinement achieved in the SR model. Hence, it is suggested to refine scenario descriptions until they reach the desired detail level. Due to the lack of space, the refined use case description for our running example will not be presented.

At the end of this activity, the requirements phase of the Domain Engineering process is concluded.

H. Product Configuration

This activity is actually a sub-process of the Application Engineering process. It will be executed every time a new product of the SPL has to be derived. This sub-process is an adaptation of Lima's Application Engineering process [11] and its three activities are described as follows:

1) Choice of Specific Configuration

In this activity, the client chooses the goals to be satisfied by the new product. There are two heuristics to guide the configuration:

- H1: All intentional elements with [1..1] cardinality must be present in the product configuration model, if their parent elements have been selected. Elements with [0..1] cardinality might be present depending on the stakeholders choices;
- H2: Sub-elements of means-end relationships will be present in the configuration model depending on the stakeholders choices, but obeying the relationship cardinality (if it exists);

Depending on the choices made by the client, there may be more than one possible product configuration, called alternatives. The SR model of each alternative will be analyzed in the next activity. In our running example, there are two alternatives: one with "Save Automatically" task (A1) and another with "Save by User" task (A2).

2) Prioritization of Variants

In this activity, the alternatives previously obtained are ranked based on the priority the client gave to the softgoals present in the SR model. The priority given to a softgoal must in the interval [0,10]. For each alternative, we have to take into account the number of positive and negative contributions from the elements of the SR model to the softgoals, considering also the degree (e.g.: help, hurt) of each contribution. The formula to calculate the priority of each variant will not be presented in this paper due to the lack of space.

The alternative with the highest priority value represents the most suitable configuration for the client's desires. For our example, if the softgoal "Quickness [Storage]" receives the highest priority, the priority value for A1 will be 7,5 and for A2 will be -7,5. Therefore, the alternative with "Save Automatically" feature is the most suitable for the client.

3) Product Artifacts Configuration

The purpose of this activity is the derivation of the artifacts for a specific product of the SPL. First, the configuration model is generated by eliminating, from the FM, all features that are not related to elements in the SR model of the chosen alternative. Then, all cardinality indications must be removed from the SR model, thus the i* model of the product is obtained. Finally, only use cases that are related to selected features will be present on the product's artifacts. The scenario descriptions must also be configured by eliminating the steps that are annotated with features that were not chosen.

IV. Related Work

Compared to G2SPL [3], our work has two main differences: (i) it uses PLUSS scenarios to capture behavioral characteristics of the SPL, while G2SPL cannot capture the SPL's behavior; (ii) our approach allows to choose a configuration based on the priority to the softgoals, while G2SPL does not provide such a mechanism.

Asadi et al. [12] also proposed a GORE approach for SPL using i*. It uses annotations on the feature model do relate goals and features. The advantage of this approach is that it has tool support, providing an automatic way to obtain the product's features from selected goals. However, the FM is not obtained systematically from stakeholders' goals, it is created separately and then annotated with goals. This approach does not capture the behavior of the SPL either.

Santos et al. [13] proposed a bidirectional mapping between feature models and goal models in the PL-AOVgraph approach. Thus, not only the FM can be obtained from the goal model, as in G2SPL and GS2SPL, but the goal model can be derived from the feature model too. The bidirectional mapping has tool support, but it covers only the Domain Engineering requirements phase, i.e., it does not provide a way to configure products.

Mussbacher et al. [14] proposed ASF (AoURN-based SPL Framework), a framework that integrates goal models, feature models and scenarios and is completely tool supported. The goal models are described in GRL (Goal-oriented Requirement Language), another i* extension, and is related to the FM through the feature impact model. Scenarios are described in UCM (Use Case Maps), a visual notation whose elements may be associated with GRL elements. However, this association must be defined by the Domain Engineer and there are no mapping rules defined on the framework. Goals and softgoals in GRL may have an "importance" attribute that allows stakeholders to indicate how important these elements are for them. This attribute is used during product configuration to prioritize the most important goals and softgoals. This represents an advantage compared to GS2SPL, since our approach prioritizes only softgoals.

Despite the fact that all ASF artifacts may be associated with each other, they are created separately, i.e., it is not possible to systematically obtain one model from another. Besides, the elaboration of the feature impact model is, according to Mussbacher et al. [14], a complex task, since it consists of a goal graph for each feature, describing how a feature affects stakeholders' goals.

V. Conclusion and Future Work

This paper presented a GORE approach for SPL, called Goals and Scenarios to Software Product Line (GS2SPL). GS2SPL allows the domain engineer to generate feature models and use case scenarios from i* goal models. The benefit of this generation is that the most relevant features and use cases to the stakeholders' goals are obtained in a systematic way. Another contribution of our approach is the inclusion of a configuration process that allows the clients to choose their product configuration on a higher abstraction level. Instead of choosing specific features, they can choose the goals they want to achieve. The configuration process also takes softgoals (NFRs) into account, providing a way to rank possible variants according to the softgoals' priority given by the client.

As future work, we plan to: (i) perform case studies in different domains to evaluate the strengths and weaknesses of GS2SPL; (ii) develop tool support for our approach; (iii) investigate how to identify feature model constraints from i* models; and (iv) investigate how to take feature interactions into account when generating use case scenarios.

References

[1] A. Lamsweerde, "Goal-oriented requirements engineering: a guided tour," in Proc. of the 5th IEEE International Requirements Engineering Conf. (RE'01), Washington, DC, USA, pp. 249-263, 2001.

[2] C. Borba and C. Silva, "A comparison of goal-oriented approaches to model software product lines variability," in LNCS, vol. 5833, pp.244-253, Springer-Verlag, 2009.

[3] C. Silva, C. Borba, J. Castro, "A goal oriented approach to identify and configure feature models for software product lines," in Proc. of the 14th Workshop on Requirements Engineering (WER'11), Rio de Janeiro, Brazil, pp. 395-406, 2011.

[4] N. Maiden, I. Alexander. Scenarios, stories, use cases: through the systems development life-cycle. 1st ed., Wiley, 2004.

[5] P. Clements and L. Northrop. Software product lines: practices and patterns. 1st ed., Addison-Wesley, 2002.

[6] K. Pohl, G. Böckle, F. van der Linden. Software product line engineering: foundations, principles, and techniques. 1st ed., Springer, 2005.

[7] E. Yu, "Modeling strategic relationships for process reengineering," in Social Modeling for Requirements Engineering, E. Yu, P. Giorgini, N. Maiden, J. Mylopoulos, 1st ed., MIT Press, 2011, ch. 2, pp. 11-152.

[8] E. Figueiredo et al., "Evolving software product lines with aspects: an empirical study on design stability," in Proc. of the 30th International Conference on Software Software Engineering (ICSE'08), Leipzig, Germany, pp. 261-270, 2008.

[9] M. Eriksson, J. Börstler, K. Borg, "Managing requirements specifications for product lines – an approach and industry case study," in Journal of Systems and Software, vol. 82, n. 3, pp. 435-447, 2009.

[10] J. Castro, F. Alencar, V. Santander, C. Silva, "Integration of i* and oject-oriented models," in Social Modeling for Requirements Engineering, E. Yu, P. Giorgini, N. Maiden, J. Mylopoulos, 1st ed., MIT Press, 2011, ch. 13, pp. 457-483.

[11] C. Lima, "E-SPL – a approach for requirements phase in domain engineering and application engineering with goal models," (in Portuguese: "E-SPL - um a abordagem para a fase de requisitos na engenharia de domínio e na engenharia de aplicação com modelos de objetivos") Dissertation (MSc), Center of Informatics, UFPE, Brazil, 2011.

[12] M. Asadi, E. Bagheri, D. Gasevic, M. Hatala, B. Mohabbati, "Goal-driven software product line engineering," in Proc. of the 26th ACM Symposium on Applied Computing (SAC'11), Taichung, Taiwan, pp. 691-698, 2011.

[13] L. Santos, L. Silva, T. Batista, "On the integration of the feature model and PL-AOVGraph," in Proc. of the 2011 International Workshop on Early Aspects (EA'11), Porto de Galinhas, Brazil, pp. 31-36, 2011.

[14] G. Mussbacher, J. Araújo, A. Moreira, D. Amyot, "AoURN-based modeling and analysis of software product lines," in Software Quality Journal (online first), 2011.

A Set of Inspection Techniques on Software Product Line Models

Rafael Cunha
Nokia Institute of Technology
INdT
Manaus, Brazil
rafael.cunha@indt.org.br

Tayana Conte
Grupo de Usabilidade e Engenharia de Software
Universidade Federal do Amazonas (UFAM)
Manaus, Brazil
tayana@icomp.ufam.edu.br

Eduardo Santana de Almeida
Reuse in Software Engineering (RiSE)
Universidade Federal da Bahia (UFBA)
Salvador, Brazil
esa@dcc.ufba.br

José Carlos Maldonado
Departamento de Ciência da Computação
Universidade de São Paulo (USP)
São Carlos, Brazil
jcmaldon@icmc.usp.br

Abstract— **Software Product Lines are an approach that enables organizations to develop a number of similar products in the same application domain reducing development and maintenance cost and increasing productivity. As in traditional software development approach, software product lines model need to be evaluated for improving software quality. This work proposes a set of inspection techniques, named SPLIT, for evaluating software product lines models. An in vitro experiment was conducted for comparing a defect type based inspection approach and the proposed set of techniques. Results indicated that our techniques found a greater number of defects than a defect type based inspection approach.**

Keywords-inspection technique; software product line; feature model; empirical study

I. INTRODUCTION

A Software Product Line (SPL) supports reusability by developing a set of products sharing a core com monalities and differing variabilities [1]. T hus, instead of in traditional software development where it usually models one product a time, it models a set of products.

An essential reason for introducing product line engineering is cost reduction [2]. Since every product uses the same core features and som e optional ones, it can be reused aiming in a cost reduction for each system. Although these artifacts can be reused, it is necessary investments for creating them planning the reuse mechanism, so that they are able to provide managed reuse. In a software product line, in general, the time to market indeed is initially higher, as the com mon artifacts have to be built first. H owever, it is considerably shortened as m any artifacts can be reused for each new product [2].

Software product lines need to address the same issues as in traditional software developmen t as inconsistencies betw een requirements and software specifications. One approach that is applied to traditional softw are development that im proves quality and reduces costs is inspection techniques, w hich is used to identify defects in ear ly stage of development [3]. Inspections of software design may be especially crucial since design defects can directly affect the quality of, and effort required for, the im plementation [3]. B ecause SPL models are quite different from single sy stem development, standard techniques are insufficient to address the specific characteristics of reusable systems [4], thus new inspection techniques tailored to the SPL models are needed.

This scenario has m otivated one of our research's goals: to define a set of inspection techniques, named SPLIT, tailored to support quality assurance concerned with specific models used for Software Product Lines specification. This paper proposes and validates, using a controlled experiment, a set of techniques to evaluate a feature model and a product map against software requirement document and feature model inconsistencies. We propose that our set of techniques find a greater number of defects than a defect type based inspection approach.

The remainder of this paper is structured as follow s: Section 2 presents background information on Software Product Lines. Section 3 presents the set of inspection techniques, SPLIT, proposed for software product line specifications. In Section 4, the controlled experiment to evaluate the set of the proposed techniques is discussed in detail, including goals and experi mental design. In Section 5, the results are presented and discussed. Section 6 describes threats to the validity of our study. Finally, conclusions and comments on future work are given in Section 7.

II. SOFTWARE PRODUCT LINE

Software Product Lines are a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particul ar market segment or mission and that are developed from a common set of core assets in a prescribed way [5]. The artifacts used in different products have to adaptable to fit each system created in the product line.

It means that throughout the development process, we have to identify and describe where the products created by the product line may differ in terms of the features they provide, the requirements they fulfill, or even in terms of the underlying architecture [5]. This flexibility is called variability and it is the basis for the software product line engineering since it represents the difference in each product produced by the Software Product Line.

Different models have been proposed for specifying Software Product Line, features models proposed in Feature-Oriented Domain Analysis (FODA) approach [6] is one of the most used [7]. Product line scoping is also an important phase in product line engineering to decide not only what products to include in a product line but also whether or not an organization should launch the product line [8]. One approach for product line scoping is product map [9] for selecting a subset of products to be created by the product line. The inspection techniques proposed use feature models and product map. These models are further discussed in the following sections.

A. *Feature Models*

Feature models are hierarchical models that capture the commonality and variability of a Product Line. This feature-oriented concept is based on the emphasis placed by the method on identifying those features a user commonly expects in applications domain.

The feature concept used for modeling the feature models is that it is a prominent distinctive user-visible aspect, quality, or characteristic of software system [6]. The purpose of the feature model is to represent the general capabilities for the applications in the domain.

Feature Models are created using mandatory, optional, alternatives and *or* features. The mandatory features represent the features, which must be contained in every product created by the product line. The optional features represents the features which can be contained or not in the product from the SPL. Alternative features represent an alternative between two or more others features so that only one of them must be contained in products from the SPL. And *or* features represent an alternative between two or more features in which at least one of them must be contained in the products from the SPL.

The feature models also have constraints elements: implication and exclusion. The implication notation implies that if one feature is selected, some other feature must be selected as well. The exclusion constraint indicates that two features must not be selected in the same product so they are mutually exclusive.

B. *Product Map*

The selection of optional or alternative features is not made arbitrarily. It is usually made based on a number of objectives or concerns that the end-user (and customer) has [6]. The product map [9] aims to complement the feature model by specifying the products, which are compliant with the software requirements.

The product map lists all features available in the y-axis and the products specified by the requirements in the x-axis. Every product in the x-axis matches a list of features according to the software product line requirements. Then the software product line only creates the products specified in the requirements.

III. Software Product Line Inspection Technique

A Software Product Line creates a set of products sharing common features and features which differs from product to product. Therefore, it is important to assure the quality of their models, as they specify several products. A mapping study [10] was conducted to obtain evidence about quality assurance techniques for SPL models. Mapping studies [10] are a formal process that uses a methodology to identify all research related to a specific topic [10]. Unlike informal literature reviews, mapping studies uses a scientific rigorous approach to assert an efficient survey of the current knowledge.

The databases used for this mapping study were the IEEE Xplore, ACM Digital Library and Scopus. Our formal process, as well as all selected studies, is described in a Technical Report [11]. The initial search returned a total of 841 papers, as described in Table I. After the first filter, in which the title and abstract from the selected papers were analyzed, it has been filtered to 90 papers. In the second filter, it was also analyzed the introduction and conclusion from papers selected after the first filter, resulting a total of 27 papers about quality assurance techniques for SPL models.

The data extracted presented that 23 (85%) papers used model checking techniques, 3 (11%) used ontologies for modeling and model checking and code inspections were presented in 2 papers (7%). Although the benefits of inspection techniques [3] and the need for inspections tailored for SPL [4], no inspection technique for assuring quality for software product line models was identified [11].

TABLE I. Papers selected in the mapping study

Database	Papers returned for the 1st filter	Papers selected after the 1st filter	Papers selected after the 2nd filter
IEEE Xplore	88	22	5
ACM Digital Library	482	28	10
Scopus	271	40	12

We have proposed a set of checklist based techniques, named Software Product Line Inspection Techniques (SPLIT), for verifying feature models and product map in comparison with themselves and the software requirements specification. The software requirement specification needs to enumerate functional and non-functional requirements and the products created by the software product line and its constraints. An overview of SPLIT is presented in Figure 1:

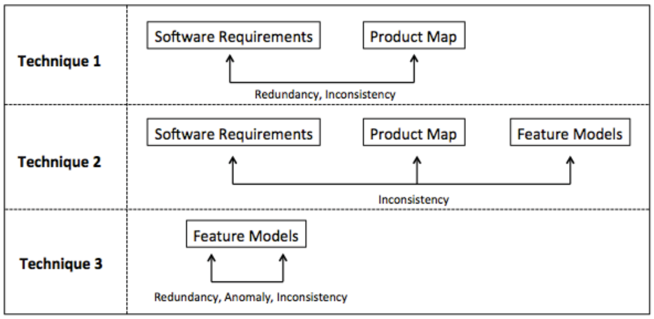


Figure 1. SPLIT overview

A. *Technique 1*

This technique analyzes the product map against the software requirement document, searching for defects that can be classified as:

- Redundancy: The product map presents products that have the same set of features and it should match products from the software requirement document.
- Inconsistencies: The product map presents inconsistencies when it does not match products from the software requirement document.

B. *Technique 2*

This technique analyzes the product map and the feature model against the software requirement document, looking for defects that can be classified as:

- Inconsistencies: The software product line artifacts (product map and feature model) present products that do not match the software requirement document. It differs from inconsistencies in the Technique 1 due to information that is strictly from the feature model.

C. *Technique 3*

This technique analyzes the feature model for defects. It has been based on [7] which identify redundancy, anomalies and inconsistencies on feature models. The defects found in this technique can be classified as:

- Redundancies: A feature model contains redundancy, if at least, one semantic information is modeled in a multiple way.
- Anomalies: A feature model contains anomalies, if potential configurations are being lost, though these configurations should be possible.
- Inconsistencies: A feature model contains inconsistencies, if the model includes contradictory information.

Table II shows a part of the checklist based Technique 3, in which is shown the defect type in the first column, the question addressed for the finding the defects in the second column and an image for helping the inspector to notice the defect easily in the feature model in the third column.

TABLE II. EXTRACT OF SOFTWARE PRODUCT LINE INSPECTION TECHNIQUE [12]

Inconsistency	Are two mutual exclusive features that are also full mandatory?	
	Are two alternative features that have an implication relationship?	

The complete set of techniques proposed is available in the Technical Report "Inspection Technique for Software Product Line Models" [12]. To assess this first version, we performed a feasibility study described in the next section, in which we compared the number of defects found by inspectors using SPLIT with the number of defects found by different inspectors using a defect type based inspection approach. We choose to compare the proposed technique with this defect type based approach, in which the subjects had been trained in the defect types they could found in SPL models, since we have not identified any inspection technique specific for software product line in the Mapping Study [11].

IV. THE EXPERIMENT

The experiment goal using the GQM (Goal/ Question/Metric) paradigm for formalizing characterization, planning, construction, analysis , learning and feedback tasks [13] is presented in Table III as:

TABLE III. EXPERIMENT GOAL USING GQM PARADIGM

Analyze:	The set of proposed inspection techniques for software product line models
For the purpose of:	Characterize
With respect to:	The number of defects found compared to a defect type based inspection approach.
From the point view of:	Software product line inspectors
In the context of:	The inspection of software product line models by undergraduate students

A. *Hypotheses*

The experiment has been design to test the following hypotheses (null and corresponding alternative hypotheses are given):

- H0: There is no difference in the number of defects found in software product line models using SPLIT and a defect type based inspection approach.
- HA1: The number of defects found in software product line models using a defect type based inspection approach is greater than using SPLIT.
- HA2: The number of defects found in software product line models using SPLIT is greater than using a defect type based inspection approach.

B. Instrumentation

The experiment was supported by a set of artifacts: consent forms, software product line specification, the SPLIT documentation for the group that would execute them, worksheet to support defects and follow up questionnaire. The software product line specification is composed by the requirement specification, product map and feature model.

The software requirement artifact is a software product line specification for a set of twitter client products with 13 features from popular twitter clients having 6 mandatory features, 3 optional features and 4 alternative features. It generates six different products sharing the common features and each of them having distinguished features. It describes each feature for the software product line detailing its relationship to others features, classifying it as mandatory, optional or alternative and enumerating the products in which this feature would be available.

The product map for the twitter client software product line listed all the features available in software requirement document and associates it to the product in which they are available. It enumerates the six products from the twitter client software product line and maps the features for each product.

The feature model for the software product line has been created according to the software requirements for the twitter client and it would generate all the products, which are specified in the product map. The feature model used in this experiment has been seeded with defects by the researchers based on deficiencies from feature models listed in [7]. It contains defects from modeling the feature model and information inconsistencies from the software requirements and the product map.

C. The Experiment Design

The subjects were divided in two groups, which would inspect the same software product line model example: the group A would evaluate using a defect type based inspection approach and the group B would use the set of techniques proposed in this paper. Since no student had a previous experience with SPL models, the subjects were assigned to each technique using completely randomized design. Each group was composed by 10 senior-level undergraduate students chosen by convenience from Analysis and Design class in Information System and Computer Science courses at Federal University of Amazonas, Brazil.

D. Preparation

The subjects signed a consent form and they had a tutorial about Software Product Line. It addressed an overview on Software Product Line concepts and specifications including feature models and product map. After the tutorial, the subjects have been trained for the experiment execution. As the classroom was divided in two groups, two different trainings were prepared. The Group A has been trained in software model inspection techniques. This training addressed the objectives of model inspection for software product lines. It described the defect types, which could be found by the inspectors when conducting the inspection in this study. The group B has been trained in the set of techniques proposed in the SPLIT. The two trainings had different instructors due to the fact that they occurred at the same time for avoiding communication between the two groups. However both materials have been prepared by the two instructors for having balanced knowledge about software inspection for mitigating the bias of prior knowledge of the subjects.

E. Execution

The experiment have been executed in a limited two hours' time box and occurred at the same time for both groups. The subjects were gathered in same room and then divided randomly in groups. Group A and B have executed the experiment in different rooms for avoiding any communication between subjects from different groups.

V. RESULTS

At the end of the experiment, the defect list form and the evaluation questionnaire about the proposed set of techniques were retrieved for the experiment analysis. The number of defects found per subject for each inspection is presented in Table IV and Table V.

TABLE IV. NUMBER OF DEFECTS FOUND IN DEFECT TYPE BASED INSPECTION APPROACH

Subject	Number of Defects Found
Subject 01	4
Subject 02	9
Subject 03	7
Subject 04	8
Subject 05	4
Subject 06	9
Subject 07	1
Subject 08	3
Subject 09	7
Subject 10	7

TABLE V. NUMBER OF DEFECTS FOUND USING SPLIT

Subject	Number of Defects Found
Subject 11	10
Subject 12	11
Subject 13	12
Subject 14	8
Subject 15	10
Subject 16	13
Subject 17	8
Subject 18	10
Subject 19	8
Subject 20	10

One example of defect that was found in the study is presented in Figure 2, where an extract from the twitter software product line feature model is shown and one defect is circled. In this defect, one mandatory feature "Post Message (F6)" is implying on optional feature "Notify update in timeline (F7)". In this case, all the products from the software product line must have the "Notify update in timeline (F7)" feature although it is an optional feature and possible product configurations would be lost.

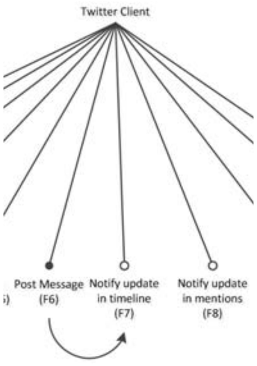
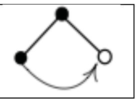

Twitter Client
)
Post Message (F6)
Notify update in timeline (F7)
Notify update in mentions (F8)

Figure 2. Feature model for the twitter client software product line

The defect highlighted in Figure 2 could be found by the set of techniques SPLIT as it was covered in the checklist in the technique 3, which is focused in the feature model. An extract from SPLIT which checks this defect is presented in Table VI.

TABLE VI. EXTRACT OF SOFTWARE PRODUCT LINE INSPECTION TECHNIQUE [12]

3.7 Is there a full mandatory feature that implies an optional feature?	

A. Quantitative Analysis

The statistical analysis was executed using the statistical tool SPSS V 19.0. 0 and α = 0.05. This choice of statistical significance was motivated by the small sample size used in this experiment [14].

Table VII presents a comparison between the defect type based inspection approach and the SPLIT set using the average and the standard deviation values for the number of defect found by the inspectors.

TABLE VII. DEFECT FOUND COMPARISON ANALYSIS

Inspection	Sampling Size	Defects Found		
		Average	*Std. Dev.*	*Std. Dev. %*
Defect type based inspection approach	10	5,9	2,726	46,20%
SPLIT	10	10	1,700	17,00%

Table VII suggests that the number of defects found by inspectors that used the technique proposed in the study is 60% higher than when executing a defect type based inspection approach. And it also has a smaller standard deviation, which reflects that the number of defects found when SPLIT has been used it shows a small variance.

Figure 3 presents a boxplot with the number of defects found by each inspection type. It can be analyzed as the mean value for the inspection when executed using the technique proposed is higher than when using than a defect type based inspection approach. We have also compared the two samples using the non-parametric Mann-Whitney test, which shows an asymptotic significances (p = 0.001) between the numbers of defects found.

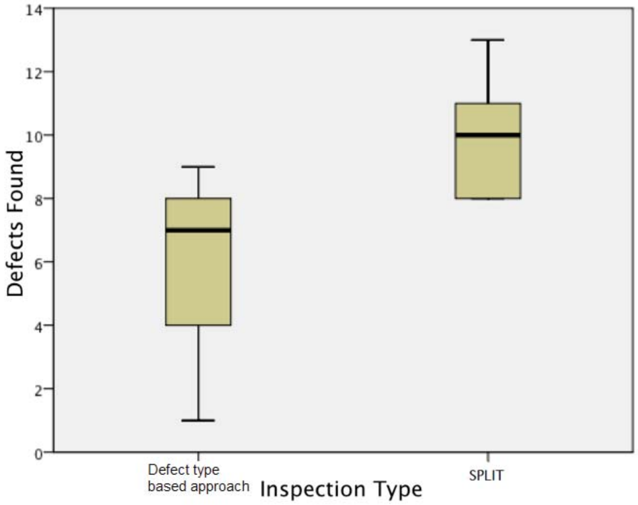


Figure 3. Boxplot for number of defects found per subject per inspection type in the experiment

These results rejects the null hypothesis H0 that there are no differences in the number of defects found by inspectors that have used a defect type based inspection approach to the ones, which have used the set of techniques proposed in the study. Due to the analysis of the boxplot and the Mann-Whitney test, the alternative hypothesis HA2 is valid for this experiment. These results suggest that the number of defects found in software product line models using the SPLIT is greater than using a defect type based inspection approach. We have not measured time for comparing the two inspection types according to its efficiency (number of defects found by hour) in this study as the time box had a limit of a two-hour class in the University.

VI. THREATS TO VALIDITY

The threats to validity have been identified in the experiment and categorized, according to [15], in one of the following categories below:

- Internal Validity: We considered the main threat that can represent a risk for an improper interpretation of the results the difference in training that the two groups have received. The first group had training in software inspection based on defect types and the second group had training on SPLIT. We have tried to minimize this threat by preparing equivalent training material for both groups.

- External Validity: We have considered two main threats concerned with generalization of the results: the subjects being undergraduate students and the software specification used in the experiment not being a real world example. For any academic laboratory experiment the ability to generalize results to industry practice is restricted by the usage of students as study participants. Although they were not real inspectors, it has been shown that empirical studies can have benefits when using students as subjects [16]. And the example used in the experiment was based in real world twitter clients. The specification of the twitter client software product line gathered features from several twitter clients available in the market.

- Conclusion Validity: The main threat concerned between the treatment and outcome used in the experiment is the small sample of data points for the statistical analysis. It is not ideal since it sometimes lacks statistical representation of phenomenon but it is a known problem difficult to overcome [17].

- Construct Validity: We have used the number of defect found by the inspector for measuring and comparing the two different inspection types used in the experiment. It is the commonly measurement used for evaluating inspection techniques as presented in [3; 18].

VII. Conclusion

In this paper, we have proposed and validated a set of inspection techniques (SPLIT) to evaluate software product line specification. It address the needs due to single system inspections are insufficient to address the specific characteristics of reusable systems [4]. SPLIT is composed by three techniques that address to find defects in feature models and product map based on the Software Product Line requirements.

We have conducted a formal experiment for comparing SPLIT against a defect type based inspection approach using a twitter client Software Product Line specification. The experiment result showed that the number of defects found when using SPLIT was greater than when executed a defect type based inspection approach. Founding a greater number of defects in early stages of development reduces costs and improves software quality.

Future work for this research should include: an improvement of SPLIT based on this experiment results and a replication of the experiment in the industrial environment.

Acknowledgment

We thank all the undergraduate students for their participation in the experiment. The authors acknowledge the support granted by CNPq and FAPESP to the INCT-SEC (National Institute of Science and Technology – Critical Embedded Systems – Brazil), processes 573963/2008 -8 and 08/57870-9; FAPEAM through process PRONEX-023/2009; and CAPES process AEX 4982/12-6. One of the authors of 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] Weiss, D.M, Lai, C.T.R.: Software Product-Line Engineering. Addison-Wesley,Reading (1999).J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

[2] Pohl, K., Böckle, G. and Van Der Linden, F. Software Product Line Engineering – Foundations, Principles, and Techniques. Springer, Berlin (2005).

[3] Travassos, G. H., Shull, F., Fredericks, M ., Basili, V.: Detecting defects in object-oriented designs: using reading techniques to increase software quality. ACM SIGPLAN Notices, vol. 34, n. 10, pp. 47-56. (1999)

[4] Denger, C; Kolb, R.: Testing and inspecting reusable product line components: first empirical results. In Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering (ISESE '06). ACM, New York, NY, USA, 184-193. (2006)

[5] Clements, P., Northrop, L.: Software Product Lines: Practices and Patterns. Addison-Wesley, Longman (2001).

[6] Kang, K.: Feature-oriented domain analysis (FODA) - feasibility study. Technical Report CMU/SEI-90-TR-21, SEI/CMU, Pittsburgh (1990).

[7] Massen, T., Lichter, H.H.: Deficiencies in Feature Models. In: Workshop on Software Variability Management for Product Derivation-Towards Tool Support. (2004).

[8] Lee, J., Kang, S., Lee D.: A Comparison of Software Product Line Scoping Approaches. In International Journal of Software Engineering and Knowledge Engineering. Vol. 20, No. 5. 637-663 (2010)

[9] Bayer, J.; Flege, O.; Knauber , P.; Laqua, R.; Muthig, D.; Schmid, K.; Widen, T.; DeBaud, J: PuLSE: a methodology to develop software product lines. In *Proceedings of the 1999 symposium on Software reusability* (SSR '99). ACM, New York, NY, USA, 122-131. (1999)

[10] Kitchenham, B., Budgen, D., Brereton, P.:The value of mapping studies – A participant-observer case study. In: Proceedings of Evaluation and Assessment of Software Engineering - EASE´2010, Keele, UK, v. 56, pp.638-651. (2010)

[11] Cunha, R.; Conte, T.: SPL Models Quality Assurance – a Mapping Study. Technical Report USES-TR-2011-004. Available at: www.dcc.ufam.edu.br/uses (2011)

[12] Cunha, R.; Conte, T.; Maldonado, J.,C.: Inspection Technique for Software Product Line Models. Technical Report USES-TR-2011-005. Available in: www.dcc.ufam.edu.br/uses. (2011)

[13] Basili, V., Rombach, H., “The TAME Project: Towards Improvement-Oriented Software Environments”, IEEE Transactions on Software Engineering, 14, (1988).

[14] Dyba, T.; Kampenes, V.; Sjoberg, D. A Systematic Review of Statistical Power in Software Engineering Experiments.Information and Software Technology. Elsevier. (2005).

[15] Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B. Wesslén, A.: Experimentation in Software Engineering – An Introduction. Kluwer Academic Publishers.(2000).

[16] Carver, J., Jaccheri, L., Morasca, S., and Shull, F., “Issues in using students in empirical studies in software engineering education”, Proceedings Ninth International Software Metrics Symposium, 3-5 Sept. (2003).

[17] Conte, T., Massollar, J., Mendes, E., Travassos, G.H.: Usability Evaluation Based on WebDesign Perspectives. In: International Symposium on Empirical Software Engineering and Measurement (ESEM) Madrid, Spain. (2007).

[18] Shull, F., Rus, I., e Basili, V.R.: How Perspective-Based Reading Can Improve Requirements Inspections. IEEE Computer, 33(7): 73-79. (2000).

Non-functional Properties in Software Product Lines: A Taxonomy for Classification

Mahdi Noorian[1], Ebrahim Bagheri[1,2], and Weichang 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*—In the recent years, the software product lines paradigm has gained interest in both industry and academia. As in traditional software development, the concept of quality is crucial for the success of software product line practices and both functional and nonfunctional characteristics must be involved in the development process in order to achieve a high quality software product line. Therefore, many efforts have been made towards the development of quality-based approaches in order to address non-functional properties in software product line development. In this paper, we propose a taxonomy that characterizes and classifies various approaches for employing non-functional properties in software product lines development. The taxonomy not only highlights the major concerns that need to be addressed in the area of quality-based software product lines, but also helps to identify various research gaps that need to be filled in future work in this area.**

I. Introduction

A. Software Product Lines

The Software Product Line (SPL) paradigm is a systematic reuse-based software development approach that is founded on the idea of identifying and capturing commonalities and variabilities of software products within a target domain [15]. Such an approach allows a new software product to be rapidly developed by exploiting a set of reusable assets, known as *core assets*, which support the management of commonality and variability. The SPL approach allows for improvements in software quality, time to market, and cost reduction [11]. The software product line approach consists of two main development lifecycles, namely *Domain Engineering* and *Application Engineering* [9]. Domain engineering involves analyzing and modeling the target domain as a whole and producing a set of reusable core assets. On the other hand, application engineering involves developing a domain-specific software product using and through the customization of artifacts that are developed in the domain engineering phase.

B. Non-functional Properties

As a part of the software development process, requirements engineering cover all activities that are involved in identifying, representing, documenting, and managing the set of needs, desired features and preferences of the stakeholders [10]. Requirements can generally be categorized into functional and non-functional. In software system requirement engineering [14], [19], the term Functional Properties (FPs) refer to the characteristics that specify the functions that the system must perform [1]; while, the term Non-functional Properties (NFPs) refers to the characteristics that are not related to the functionality of the software [4] but are essential for the operation and acceptance of the system. In general, FPs define the ‘what of a software system whereas NFPs address questions pertaining to the ‘how of the software system performance.

C. Objectives of This Research

The general purpose of our ongoing research is to provide a quality-aware framework for software product line development. To achieve this goal, it is required to understand how the NFPs can be managed and employed in the SPL development lifecycle. In order to cater quality-aware SPL development, there are several fundamental research questions that need to be answered first, such as:

- What are the main tasks/stages within the domain engineering and application engineering lifecycles that need to be cognizant of quality?
- What are the most frequent NFPs used by SPLs practitioners and how are they modeled and represented?
- What are the different types of NFPs that are used in different stages of SPL development?
- How can NFPs be systematically defined and measured in the context of software product lines? (This is specially more complex as a product line has the potential to develop a vast number of individual applications with different quality levels, c.f. [13]).

In the first step, we propose a taxonomy that can be used to classify the available research literature in intersection of NFPs and SPLs. The proposed taxonomy provides us with the opportunity to systematically investigate and extract the prominent information from existing research works in NFPs and SPLs and be able to draw valid conclusions about how to best pursue the incorporation of non-functional properties within software product lines.

D. Outline

The remainder of this paper is organized as follows: In Section II, the proposed taxonomy is presented. In addition, the dimensions of the taxonomy are discussed in detail. Section III is devoted to presenting some prominent work in the area of NFPs and SPLs. Then this set of representative work is classified according to our proposed taxonomy. We conclude the paper with conclusions and direction for future work in Section IV.

II. The proposed Taxonomy

The taxonomy characterizes and classifies the main aspects of quality-based approaches in the context of software product lines. In order to clarify the standpoint of NFPs in SPLs, we propose this taxonomy. The dimensions and sub-dimensions of the proposed taxonomy are depicted in Fig. 1. In the rest of this section, we look at each dimension and its related sub-dimensions in more detail.

A. Dimension 1- Main lifecycle

As defined by Kang et al. [9], SPL development consists of two main lifecycles, namely domain engineering and application engineering. In order to fulfill a quality-based SPL development process, it is required to study the impact of NFPs in both lifecycles separately. This first dimension is devoted to the introduction of the major quality-based tasks/steps that need to be performed in both domain and application engineering lifecycles.

Domain Engineering

Based on Kang's definition [9], domain engineering has three main phases, *1) domain analysis, 2) reference architecture development*, and *3) reusable component development.*

1) domain analysis

Domain analysis is the process of identifying, eliciting and modeling the requirements of a family of products. The major quality-based tasks that can be categorized in this phase are: *identification and elicitation* of functional and non-functional properties; *modeling*, which aims to model and represent NFPs in a structured format. In the context of feature models, which represent functional properties, the extended feature model [9] is an instance of a structured representation model for representing NFPs; *integration* of functional and non-functional models, which helps to have both models under one umbrella as it is useful to understand and identify the mutual impacts between them; and *model evaluation*, the derived model itself needs to be evaluated in terms of some expected quality attributes.

2) reference architecture development

The main purpose of the reference architecture development phase is to provide a common architecture or *reference architecture* for a particular domain. On the other hand, in quality-aware reference architecture in addition to FPs the developed architecture is influenced by predefined domain NFPs. In order to obtain quality-aware reference architecture it is required to consider: the *architecture design* stage in such a way that predefined NFPs are employed in the design phase, e.g., the components and connectors should be designed in such a way that support specific quality attributes; *trade-off analysis*, when designing the software architecture to meet any of the NFPs, it is necessary to consider the mutual impacts of the NFPs and analyze the tradeoffs between them. The importance or priority of each NFP differs from system to system; in *architecture assessment* it is required to assess the developed architecture in terms of its capability for supporting the expected NFPs.

3) reusable component development

The last phase of domain engineering is devoted to implementation. It is important to design a component and perform coding in such a way that the expected quality can be satisfied. For example, for achieving the expected security level, developers should follow certain secure coding standards. Therefore, in order to achieve quality-aware implementation, *component design* and *coding* sub-dimension are defined.

Application Engineering

In application engineering, there are also three phases [9], *1) user requirements analysis, 2) application architecture selection*, and *3) application software development.*

1) user requirements analysis

User requirement analysis is concerned with capturing and managing the target product requirements. In this phase, the main task is to derive an instance model from the general domain model that is developed in the domain engineering lifecycle. In the context of feature models, the derived model can be employed in the software product configuration process.

In quality-aware software product configuration, it is required to consider: an *objective function*, which captures the end-user's desirable functional and non-functional properties; *optimal feature selection*, the feature selection process should be conducted according to the end-user's predefined requirements. This process leads to an optimized product. The optimized product is the one that satisfies all desirable FPs and NFPs optimally; *trade-off analysis*, during the configuration process various trade-off scenarios should be defined in order to resolve the potential conflicts and maximize the end-user's intended NFPs; *model verification*, after the target product is configured the final product needs to be assessed based on some verification process. The verification aims to confirm that the target product is consistent with respect to domain and end-users predefined functional and non-functional properties.

2) application architecture selection

As mentioned before, the reference architecture covers all possible software architectures in a products family. In this phase, the main task is to provide an instance from the reference architecture and use it to develop a concrete software product. Therefore, the quality-aware *architecture instantiation* sub-dimension is defined to identify the work that consider both user-defined and domain NFPs in developing the software architecture for a concrete product.

3) application software development

In *application software development*, the main task is to implement target products using the common assets such as reusable components. The role of quality in this phase can be seen in the following steps: *component selection*, *component integration*, and *software product validation*. In both *component selection* and *component integration*, the concern is to select and integrate components in such a way that the target NFPs can be satisfied. In *software product validation*, the purpose is to validate the final implemented product in terms of its predefined NFPs and observe how much the NFPs are satisfied.

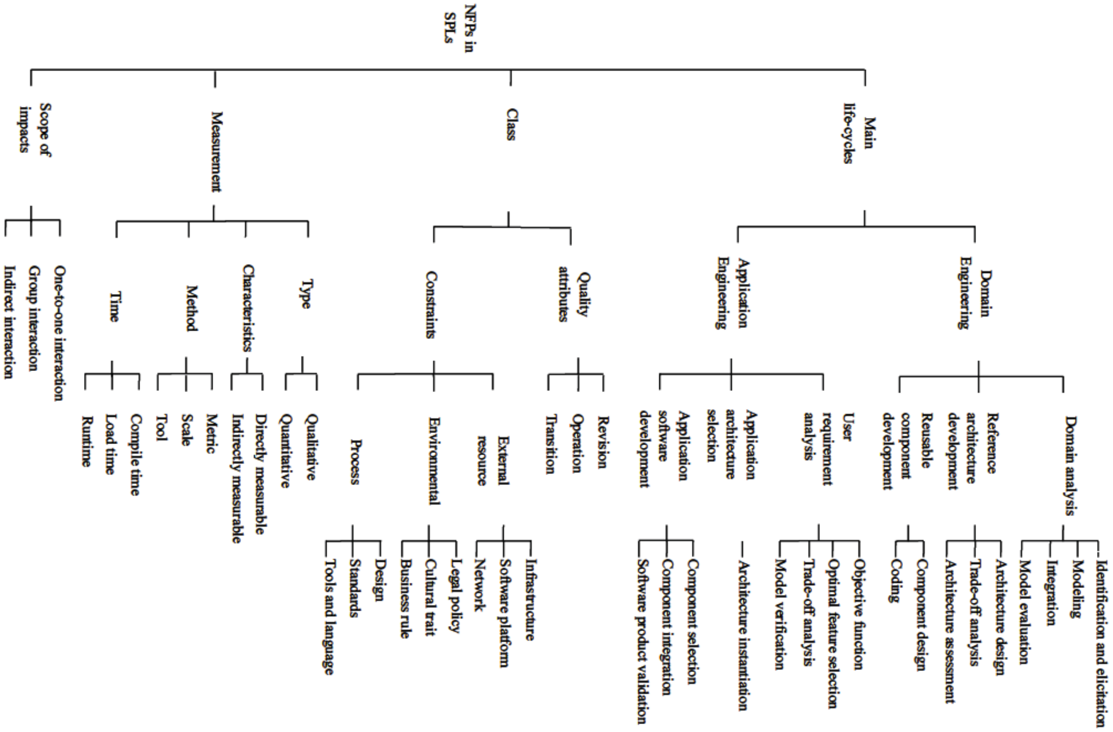


Fig. 1. A taxonomy for non-functional properties in software product lines.

B. Dimension 2- Class

The main goal of this dimension is to understand and categorize the NFPs and study the main NFPs that have been employed in SPL development. There are many classification frameworks for NFPs in the literature each of which have been proposed to classify NFPs in general [10], [6] or for a specific application domain [5]. Two classification frameworks were particularly influential to introduce Dimension 2, the McCall quality model [12] and the classification proposed in [18]. The non-functional properties in our proposed taxonomy are divided into two main categories: *1) quality attributes* and *2) constraints*.

The *quality attributes* sub-dimension addresses quality attributes and based on McCall quality model it can be categorized as: *product operations*, which are concerned with attributes that address the product's operation characteristics, e.g. performance and usability ; *product revision*, which is concerned with the attributes that address the product's ability to undergo changes, e.g. maintainability and testability ; and *product transition*, which is concerned with the attributes that address the products ability to adopt to new environments, e.g. portability and reusability.

On the other hand, we introduce the *constraints* sub-dimension. The NFPs sometimes appear as constraints and impose restrictions on various stages of development. In our proposed taxonomy, *constraints* are categorized as: *external resource*, *environmental*, and *process*. The *external resource* can be categorized as: *infrastructure, software platform* and *network*. The *external resource* can impose specific quality constraints on the target software product, e.g., the minimum network bandwidth for the software product *P* is 1 Mb/s, or the product *P* is only compatible with Windows OS platform. In addition, *environmental* constraints can be identified as: *legal policy, cultural trait* and *business rule*, e.g., in order to obtain Canadian market, the software product *P* needs to support both French and English languages. Also, *process* constraint can be categorized as: *design, standards* and *tool and language*, e.g., the software product *P* needs to be developed based on Web 2.0 standards and using the Java language.

C. Dimension 3- Measurement

There are various methods that have been proposed for measuring NFPs in traditional software engineering. Recently, a number of methods and techniques have been proposed for measuring NFPs in the SPL context. This dimension is devoted to studying the approaches that have been employed to measure the impact of non-functional properties. The *measurement* dimension is classified as *1) type, 2) characteristic, 3) method,* and *4) time*. Measurement *type* can be classified as *quantitative* and *qualitative*. Quantitative measurement is concerned with numerically measuring the NFPs for a particular artifact with continuous values while qualitative measurement is concerned with NFPs with non-numeric values. Measurement *characteristic* can be classified as *directly measurable* and *indirectly measurable*. The *directly measurable* approach is concerned with measuring the basic attributes that cannot be divided into other attributes. On the other hand, *indirectly measurable* NFPs are those that are dependent on other basic attributes and need to be divided into several measurable attributes. In the measurement *method* sub-dimension, we study *metric* and *scale* sub-dimensions and how they pertain to measurement of each individual NFP. In addition, we take a look at possible *tool* support for NFP measurement. The measurement *time* sub-dimension is categorized as, *compile time, load time*, and *runtime*. This sub-dimension targets the measurement stage. In other words, we intend to realize what NFPs should be measured in what stage of execution time. For instance, the security attribute should be measured at runtime and footprint can be measured at compile time.

D. Dimension 4- Scope of impacts

Functional/nonfunctional properties are usually interdependent and one attribute can affect others. Dimension 4 is defined to identify the research that consider the mutual impacts between functional/non-functional properties and see how this interaction can be measured. We classified this dimension to *one-to-one interaction, group interaction* and *indirect interaction*. We consider *one-to-one interaction* when one functional/non-functional property has impact on a functional/non-functional property based on one-to-one interaction, e.g., security can have direct negative impact on usability. In *group interaction*, a group of functional/non-functional properties have impact on one or group of functional/non-functional property(s). In *one-to-one* and *group* interactions the properties directly impact each other while in *indirect interaction*, one property indirectly impacts others.

III. Survey on Non-functional Requirements and Software Product Lines

In this section, some research work in the area of NFPs and SPLs has been selected for supporting the proposed taxonomy. The following four works are only introduced to show the potential of our taxonomy. A mapping of taxonomy to several selected work in NFPs and SPLs is shown in Table 1.

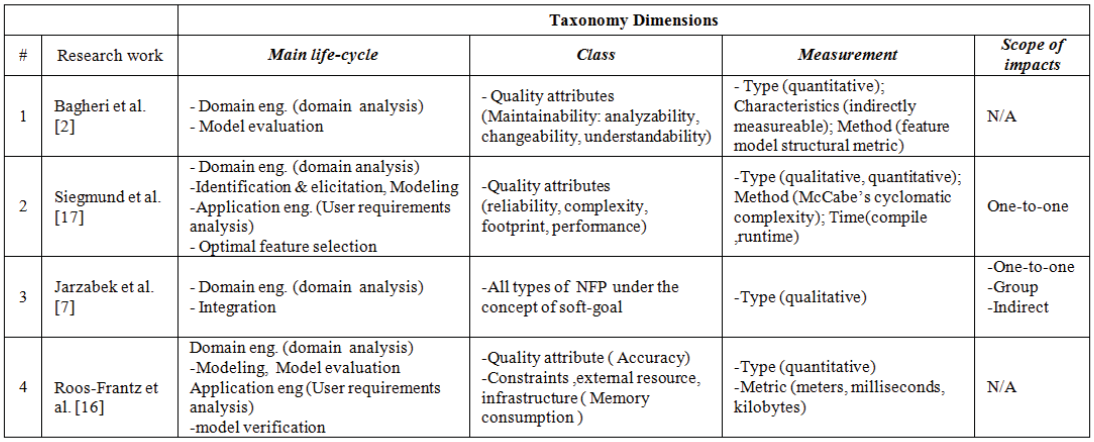
In [2], Bagheri et al. proposed an approach to predict the maintainability of a software product line by measuring the quality of the designed feature model in the early stages of the development process. For this purpose, a set of feature model structural metrics such as size and complexity are proposed. According to our proposed taxonomy, this work has been designed for the *domain analysis* phase and the quality-based task was *model evaluation*, which tried to evaluate the feature models quality of design. The non-functional property that is covered in this work is *maintainability* that can be categorized under the *revision* quality attributes. With respect to measurement dimension, *quantitative* measurement has been utilized by the authors. In addition, maintainability was measured *indirectly* using analyzability, changeability, and understandability sub-attributes. Furthermore, the feature model structural parameters such as feature model size, length and complexity are defined as metrics for measurement. The scope of impacts dimension is not applicable for this work since the NFPs mutual interaction is not considered by the authors.

For the work presented in [17], Siegmund et al. have proposed an approach named SPL Conqueror, that addresses the problem of automatically configuring a software product with respect to a set of predefined NFPs. The quality-aware tasks in this work can be categorized in both domain and application engineering. In *domain analysis*, the authors address two type of tasks: the *identification & elicitation* task through which the NFPs of the target domain can be elicited and specified by the domain expert; and the *modeling* task during which the feature model is enriched with the identified NFPs. In addition, the NFPs are measured and the results of the measurements are stored. On the other hand, in the *user requirements analysis* step, the optimal feature set is computed and the optimal software product is derived. Thus, this task would fall under the *optimal feature selection* sub-dimension. Regarding NFP *class* dimension, the selected NFPs by the authors are: reliability, complexity, footprint, and performance. With respect to *measurement* dimension, for *qualitative* measurement, the domain experts assign ordinal value to features. Also, for *quantitative* measurement the domain expert assign a proper metric to each NFP. A *tool* has been provided by the authors to measure the NFPs automatically. For measuring the complexity attribute, McCabe's cyclomatic metric is used. In addition, the Source Monitor tool is used by the authors in the measurement process. For the *time* sub-dimension, the footprint attribute measured is at*compile time* and the performance attribute at *runtime*. In the work by Siegmund et al. the concept of FP-FP interaction is employed which falls under the *one-to-one* interaction sub-dimension.

In the work presented in [7], the integrated modeling framework called feature-softgoal interdependency graph (F-SIG) is proposed. In order to support the concept of quality in SPL, the authors extend the Feature-Oriented Domain Analysis (FODA) [8] method with a goal-oriented approach. From the perspective of our proposed taxonomy, this work can be categorized in the *domain analysis* phase, which supports *integration* of two major modeling approaches namely, FODA and SIG [3]. In terms of the *class* dimension, the F-SIG model supports a wide variety of NPFs under the concept of soft-goal. Note that in this work, since the concept of NFPs is presented using the idea of soft-goals, there was no specific method for measuring the NFPs quantitatively and all NFPs are addressed qualitatively. With respect to the *scope of impacts* dimension, all types of interactions *one-to-one, group*, and *indirect* can be directly understood or inferred from the F-SIG model. The interaction can be considered as: feature-to-feature (FP-to-FP) structural interdependency, NFP-to-NFP structural interdependency, and FP-to-NFP interdependency (the influence of FP on specific NFP can implicitly be expressed).

In [16], Roos-Frantz et al. have proposed an approach for quality-based analysis in software product lines. For verification purposes, the quality-annotated variability model is mapped to a constraint satisfaction problem and using constraint solver the verification task is conducted automatically. Within our proposed taxonomy, this work can be classified under the *domain engineering* and *application engineering* lifecycles. The performed quality tasks in domain engineering are: *modeling*,where the orthogonal variability model (OVM) is extended with NFPs; *model evaluation*, during which the developed variability model can be evaluated to find a void model, dead elements and false optionals. Both *modeling* and *model evaluation* are categorized in the *domain engineering*

Table 1 TAXONOMY MAPPING ON RESEARCH WORK IN NFPS AND SPLS



		Taxonomy Dimensions			
#	Research work	***Main life-cycle***	***Class***	***Measurement***	***Scope of impacts***
1	Bagheri et al. [2]	- Domain eng. (domain analysis) - Model evaluation	- Quality attributes (Maintainability: analyzability, changeability, understandability)	- Type (quantitative); Characteristics (indirectly measureable); Method (feature model structural metric)	N/A
2	Siegmund et al. [17]	- Domain eng. (domain analysis) -Identification & elicitation, Modeling -Application eng. (User requirements analysis) - Optimal feature selection	-Quality attributes (reliability, complexity, footprint, performance)	-Type (qualitative, quantitative); Method (McCabe's cyclomatic complexity); Time(compile ,runtime)	One-to-one
3	Jarzabek et al. [7]	- Domain eng. (domain analysis) - Integration	-All types of NFP under the concept of soft-goal	-Type (qualitative)	-One-to-one -Group -Indirect
4	Roos-Frantz et al. [16]	Domain eng. (domain analysis) -Modeling, Model evaluation Application eng (User requirements analysis) -model verification	-Quality attribute (Accuracy) -Constraints ,external resource, infrastructure (Memory consumption)	-Type (quantitative) -Metric (meters, milliseconds, kilobytes)	N/A

phase. On the other hand, in application engineering for the *user requirements analysis* phase, the *optimal feature selection* sub-dimension can be considered as a proposed quality-based task in the work by Roos-Frantz et al. In this case, the general model can be checked to find whether any software product can be identified with respect to the defined NFPs. In this work, the NPFs form both classes, *quality attribute* and *constraints*, e.g., accuracy, cost, latency, and memory consumption. For measurement purposes, this work only support *quantitative* measurement. Based on the application domain, latency is measured with milliseconds and memory consumption is measured with kilobytes.

IV. Conclusions and Future Work

In this work, we have presented a taxonomy regarding the role of NFPs in SPL. This taxonomy focuses on the main aspects that need to be addressed for developing a quality-aware products and product lines in the SPL context. The proposed taxonomy consists of four main dimensions, *main lifecycle*, *class*, *measurement*, and *scope of impacts*. We have discussed each dimension and introduced the related sub-dimensions. In addition, in order to discuss our taxonomy, we briefly survey some prominent research work in the field and appropriately classify them into different categories according to the proposed taxonomy.

Our direction for future research is to perform a comprehensive survey in the field and classify all current research work and report the status in the area of NFPs and SPL. The collected information from the classification process will assist us to identify the possible enhancements to fill the existing gaps between these two areas. Our initial probe has shown that our proposed taxonomy is quite strong in providing the means to capture various aspects of work in NFPs and SPLs.

References

[1] Ieee standard glossary of software engineering terminology. *IEEE Std 610.12-1990*, page 1, 1990.

[2] Ebrahim Bagheri and Dragan Gasevic. Assessing the maintainability of software product line feature models using structural metrics. In *Software Quality Journal 19(3):579-612*. Springer, 2011.

[3] Lawrence Chung, Brian Nixon, Eric Yu, and John Mylopoulos. *Non-Functional Requirements in Software Engineering*. Kluwer Academic Publishers, New York, 2000.

[4] Lawrence Chung and Julio Cesar Prado Leite. Conceptual modeling: Foundations and applications. chapter On Non-Functional Requirements in Software Engineering, pages 363–379. Springer-Verlag, Berlin, Heidelberg, 2009.

[5] M. Galster and E. Bucherer. A for identifying and specifying non-functional requirements in service-oriented development. In *Services - Part I, 2008. IEEE Congress on*, pages 345 –352, july 2008.

[6] M. Glinz. On non-functional requirements. In *Requirements Engineering Conference, 2007. RE '07. 15th IEEE International*, pages 21 –26, oct. 2007.

[7] S. Jarzabek, B. Yang, and S. Yoeun. Addressing quality attributes in domain analysis for product lines. *IEEE Proceedings - Software*, 153(2):61–73, 2006.

[8] 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, November 1990.

[9] Kyo C. Kang, Sajoong Kim, Jaejoon Lee, Kijoo Kim, Gerard Jounghyun Kim, and Euiseob Shin. Form: A feature-oriented reuse method with domain-specific reference architectures. *Annals of Software Engineering*, 5:143–168, 1998.

[10] Gerald Kotonya and Ian Sommerville. *Requirements Engineering - Processes and Techniques*. John Wiley & Sons, 1998.

[11] Frank J. van der Linden, Klaus Schmid, and Eelco Rommes. *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

[12] James A. McCall. *Quality Factors*. John Wiley & Sons, Inc., 2002.

[13] Bardia Mohabbati, Dragan Gasevic, Marek Hatala, Mohsen Asadi, Ebrahim Bagheri, and Marko Boskovic. A quality aggregation model for service-oriented software product lines based on variability and composition patterns. In *The 9th International Conference on Service Oriented Computing (ICSOC 2011)*. Springer, 2011.

[14] Bashar Nuseibeh and Steve Easterbrook. Requirements engineering: a roadmap. In *Proceedings of the Conference on The Future of Software Engineering*, ICSE '00, pages 35–46, New York, NY, USA, 2000. ACM.

[15] Klaus Pohl, Günter Böckle, and Frank J. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[16] Fabricia Roos-Frantz, David Benavides, Antonio Ruiz-Cortes, Andre Heuer, and Kim Lauenroth. Quality-aware analysis in product line engineering with the orthogonal variability model. *Software Quality Journal*, pages 1–47. 10.1007/s11219-011-9156-5.

[17] Norbert Siegmund, Marko Rosenmuller, Martin Kuhlemann, Christian Kastner, Sven Apel, and Gunter Saake. Spl conqueror: Toward optimization of non-functional properties in software product lines. *Software Quality Journal*, pages 1–31. 10.1007/s11219-011-9152-9.

[18] Ian Sommerville and Pete Sawyer. *Requirements Engineering: A Good Practice Guide*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1997.

[19] P. Zave. Classification of research efforts in requirements engineering. In *Requirements Engineering, 1995., Proceedings of the Second IEEE International Symposium on*, pages 214 – 216, mar 1995.

A Proposal of Reference Architecture for the Reconfigurable Software Development

Frank José Affonso
Department of Statistics, Applied Mathematics and Computation
Univ Estadual Paulista - UNESP
Rio Claro - SP, Brazil
affonso.frank@gmail.com / frank@rc.unesp.br

Evandro Luis Linhari Rodrigues
Department of Electrical Engineering
University of São Paulo - USP
São Carlos - SP, Brazil
evandro@sc.usp.br

***Abstract*—The software development process is marked by constant changes in customer needs or technological innovations which have emerged in recent years. Moreover, we can highlight the search for tools and automated processes to assist in development activities. In this context, this paper aims to propose a reference architecture model for developing reconfigurable software, which has a specific feature, allowing changes to be incorporated without the need of its execution interruption. This model allows one to create a standard for the software development of this nature, making requests for changes which will be implemented naturally. We believe, with this paper proposal, a concrete architecture for reconfigurable systems can be defined.**

I. Introduction

The need for computational systems endowed with runtime reconfiguration characteristics is an old desire of software engineers to incorporate their customers' emerging needs [1]. To meet this desire, there are computational reflections [2], [3], [4] as a mechanism for the implementation of software reconfiguration in runtime. This mechanism can be applied to several system approaches in development, such as oriented objects, components, aspects, services, remote method invocation and the combination of both [5].

Regarding to the reconfiguration systems context, we have observed that the related work corresponds to individual projects, without the adoption of tools and engineering processes. Concerning the two last items, some authors Whitehead [6], Nakagawa [7], report about the lack of tools to fully meet the project phases (software life cycle) and, when they meet them, there is a problem in relation to investment (high cost) for acquisition and maintenance.

Given the mentioned context, reconfigurable software in runtime and engineering software tools to support all project phases, this paper proposes a reference architecture model for reconfigurable systems such as automated processes. This proposal is associated with use of a Reconfigurable Software Development Methodology (RSDM), a Reconfigurable Execution Environment (REE) and a software engineering tool, named ReflectTools, both developed in the authors' previous work [5], [8].

The RSDM provides a guidelines set for developing reconfigurable software (classes, components, aspects, services) - named software artifacts. Basically, its guidelines lead to the development based on requirements segmentation, leaving the artifact free of any non-functional requirements. Furthermore, it adopts the architectural style based on layers to organize the application and requirements combination (functional and non-functional). This combination can be made by aspects or dependency injection [5], [9].

The REE and ReflectTools represent the automatic mechanisms to support the RSDM of guidelines. Despite the project specific features, was established the ability to integrate with the tools (workbenches) development, such as Eclipse and Netbeans. These tools are used in the initial steps of RSDM (modeling and system developing). After completing these steps, the projects are transferred to ReflectTools so that a software engineer or developer can use it directly in the REE to automate tasks [5].

The reference architecture model for reconfigurable systems helps in the processes automation in the REE and software engineer tools (for example, ReflectTools). This model not only helps to understand the domain structure but also the vocabulary applied to reconfigurable systems. Thus, we believe, with this paper proposal, a concrete architecture for the use in reconfigurable systems can be defined.

II. Concepts, Definitions and Related Work

This section presents the concepts, definitions and related work that contributed to the development of this paper. Initially are presented concepts of Computational Reflection (CR) and their reconfiguration techniques. Following are presented related work on software architecture.

The CR [2], [10] can be defined as any activity performed by a system on itself. The main objective is to obtain information about its own activities, aiming to improve its performance, introduce new capabilities (features or functionality), or even solve its problems by choosing the best procedure.

According to Borde [10], Gomaa & Hussein [11], the CR is used for the software components adaptation in two ways: **(1)structural**, involving the component structures such as change interface or operation name, and **(2)behavior**, which affects the functional and non-functional properties [12]. This technique is centered on the packaging due to original interface incompatibilities. The existing features are preserved and others, relating to new requirements are added forming a new software component.

In Tanter et. al. [13], the Aspect-Oriented Programming (AOP) is used in structural and behavioral systems. Aspects are used to weave non-functional requirements to system objects. The weaving requirements (functional and non-functional) are performed by the tool named Reflex.

The component reconfiguration in distributed environments is approached by Chen [14]. His proposal is an RMI (Remote Method Invocation) extension, XRMI - eXtended Java RMI, which allows an application to monitor and manipulate invocations between components during a dynamic reconfiguration process. The remote objects are used as if they were local objects.

For Williams & Carver [15], the changes that occur with the software are inevitable. The main reasons are the users' changing needs and issues related to technological adaptation. At work, the author point out a study on the software architecture flexibility to meet the required changes without burdening the maintenance activity.

According to Navasa [16], software architecture should be easy modification or reconfiguration. The authors proposed the AspectLEDA - a language for description of software architecture by using aspects. Aspects act as a facilitating mechanism for software adaptation (manual or runtime). The software interests are divided in functional and non-functional, which can be identified, grouped and reused in various system units.

The RefASSET (Reference Architecture for Software Engineering Tools) is a proposal for a Software Engineering Environment. This environment has a tool set that operate in the following phases of software life cycle: documentation, configuration management, quality assurance, verification, validation, joint review, and audit. Basically, the proposed development is centered on a software model with the addition of non-functional requirements with aspects [7].

According to Santos [17], frameworks for domain-specific languages (DSL - Domain Specific Language) assist the product line software development. The author comments on increasing the productivity of software by using DSLs; however, it is necessary that software engineers make the architectural design and act in the construction of meta-models and code generators. On the architectural design, they mention the use of aspect-oriented programming as mechanism connector components.

Finally, the study conducted by Kazman [18], a contribution to the evaluation and design of software architecture called APTIA (Analytic Principles and Tools for the Improvement of Architectures) can be found. This paper offers a guideline set for an automated process, which helps the software engineer to improve the software architecture.

III. A Proposal of Reference Architecture

This section presents a reference architecture model for the reconfigurable software development such as automated processes [5]. This model represents a real solution (abstraction), based on a particular domain (reconfigurable systems) and experience (patterns) [5], [8] to treat the runtime software reconfiguration without the developers' participation.

The solution adopted for this model is directed to systems developed in programming language that have the following features: reflection, dynamic compilation and dynamic loading of artifacts. The reflection is associated with artifacts architectural flexibility, since the information on its structure and its execution state can be retrieved and reused when the artifact is modified [2], [4]. The dynamic compilation and dynamic loading of the artifacts are related to how these artifacts can be obtained, compiled and reinserted in the execution environment [2], [4], [5]. Figure 1 shows the reference architecture model, emphasizing the modules for artifacts runtime modifications.

The model presented in Figure 1 is organized in seven modules: annotations (`annotationManager`), state management (`stateManager`), reflection (`reflectManager`), source code generator (`sourceCodeManager`), query and rules (`queryManager`, configuration (`configurationManager`) and reconfiguration (`reconfiguratorURLManager`). The following are descriptions and relations between modules to perform reconfiguration automatically.

The annotations module (`annotationManager`) aims to assist the software engineer in the definition of artifacts reconfiguration level which are being developed. For the reflection module, represented by the `reflectManager` package, to identify what information can be modified (removed and/or inserted), a metadata (annotation) indicating the reconfiguration level supported by the artifact must be present. Furthermore, there is an interest, the reflection module, to identify if the artifact type represents a **logical class**, which only has functionalities; or, a **persistence class**, which has logical storage in database. It is recommended that this module has a functionality to verify if the annotations were inserted correctly, otherwise the reflection module cannot identify what information can be reconfigured.

The state management module (`stateManger`) aims to preserve the artifacts execution state. When a software artifact is selected for manual or automatic reconfiguration,

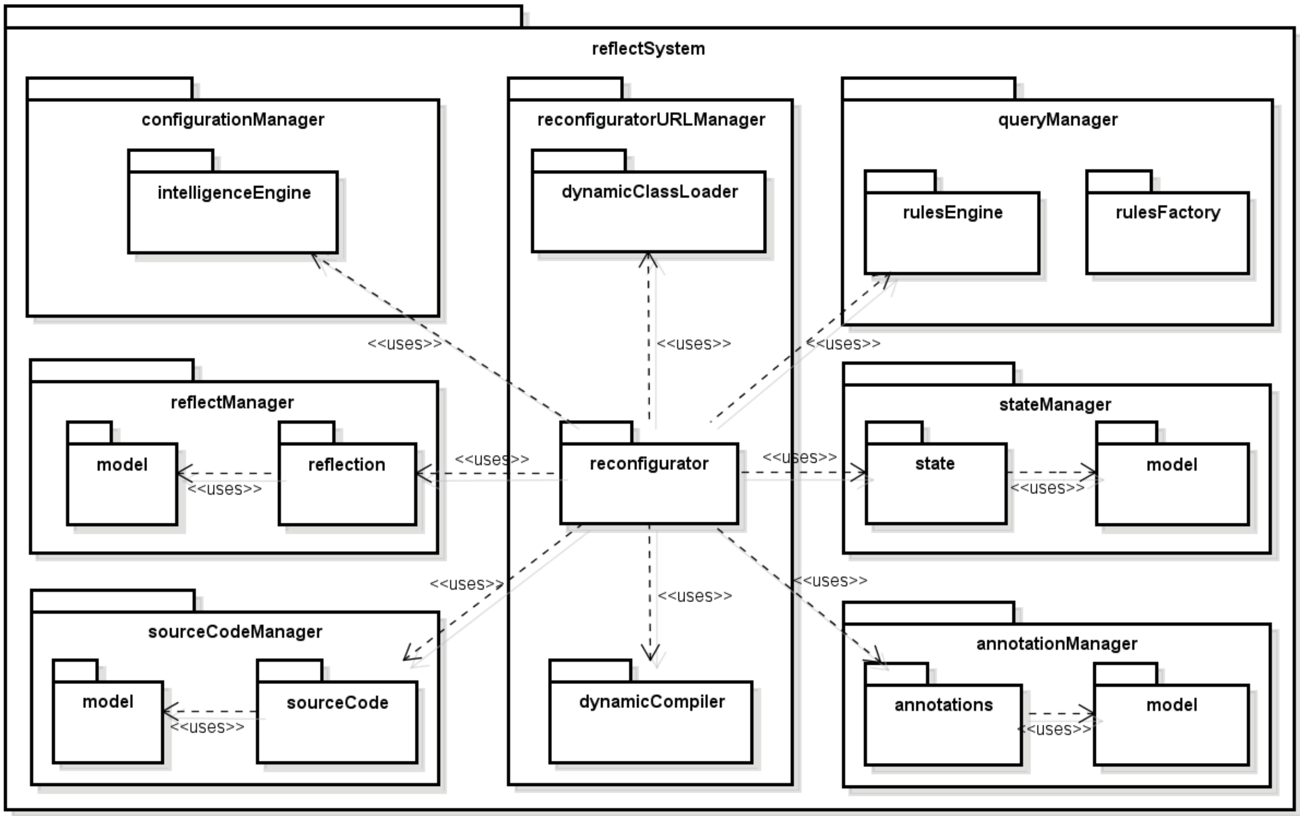


Figure. 1. Reference Architecture Model

the information contained in its current state should be preserved. The artifact is modified and the information is reinserted so that the execution is not interrupted. Basically, this module should have two functionalities to convert an artifact into a file (`.xml`) and vice versa. The choice of XML, eXtensible Markup Language, to perform these operations is related to the following facilities: files handling (reading and writing), integration with different programming languages and implementation facility.

The reflection module (`reflectManager`) aims to perform the artifacts "disassembly" to obtain structural information (attributes) and behavior (methods). Basically, the artifacts disassembly is conducted by using the `reflectManager.reflection` package, which uses the annotations module to identify changes that can be performed on the artifacts. This information is retrieved and inserted into a metamodel in the `reflectManager.model` package. After instantiating the metamodel, new information according to the clients' interests are added to create a new metamodel. This metamodel is transferred to the source code generator module to create the new artifact.

The source code generator module (`sourceCodeManager`) aims to generate the software artifacts based on metamodel (instantiated in reflection module). To execute this operation, the software engineer must provide an **artifact template** based on the architectural pattern (logical layer, persistence layer, and others). This module should has four functionalities to generate the source code that meets the reconfiguration interests: (1)when only the artifact source code, without modification, must be generated, (2)when only an attributes list should be added or removed from the artifact. In this case, specifically, the getters and setters methods that manipulate these attributes are modified, (3)when only a methods list should be added or removed from the artifact, and (4)when an attributes list and a methods list should be added or removed from the artifact.

The query and rules module (`queryManager`) aims to be consulted by software artifacts in REE repositories. When an artifact is developed and inserted into the REE, an automatic mechanism (`rulesFactory`) is responsible for disassembling this artifact and creating rules set that describes the artifact functionalities. These rules are stored in REE repositories and reused when a search (`rulesEngine`) is performed. The rules model (`rulesFactory` package) should be compatible with the artifact metamodel in the package `reflectManager.model`.

The configuration module (`configurationManager`) aims to control the size of software artifacts when the reconfiguration is performed. The artifacts are developed to meet specific requirements and to act in a specific domain. Changes may occur so that they adapt to operate in the same domain or in different domains. Therefore, the presence of a configuration manager that allows performing the growth control in the artifacts size and a number of adaptations performed is desirable.

Finally, the reconfiguration module (`reconfiguratorURLManager`) can be considered the "orchestrator" of the model presented in Figure 1, since it performs call and coordinates all the activities of the other modules. This module must implement a "**connection point**" as a system supervisor (`reconfigurator` package) between the dynamic compiler (`dynamicCompiler` package) and dynamic classloader (`dynamicClassCloader` package). The module functionalities aim to compile/recompile the artifacts software and upload its binary code in runtime (memory). A desirable characteristic for this module, specifically in dynamic compiler, is the ability to diagnostic errors in source code, since the error messages are useful information for the interpretation and corrections in the source code. To conclude the module description, it is recommended that the process is performed automatically and managed by software engineering tools, since it reduces implementation complexity and minimizes the generation of uncertainties.

IV. Conclusions

This paper presented a proposal for reference architecture model for the reconfigurable systems development using automated processes. This proposal is based on modules set (Figure 1), which are responsible for making artifacts adaptations in runtime. This process minimizes developers' involvement and therefore reduces the generation of uncertainty.

In the development context, it is also considered that standardization should be adopted in the design and development of reconfigurable systems (RSDM). The ReflectTools allow functionality with potential reuse in other systems which are identified and stored in information repositories. They can be reused as remote methods (Remote Objects) or services (Web Services) in the design of other systems: (1)increasing the development speed of new artifacts, or (2)contributing significantly as the information basis for the reconfiguration process (manual or automatic) [5], [6], [8].

Finally, it emphasizes the reusability of the reference architectural model (Figure 1) for reconfigurable systems such as automated process. We believe that the guidelines presented in Section III are the basis for the model instantiation to act on artifacts developed in other programming languages. The restriction for the use of this model is related to programming language, since it should have resources for reflection, dynamic compilation and dynamic loading of artifacts.

V. Acknowledgment

This work is supported by the PROPe/UNESP (Prorector of Research / Univ Estadual Paulista) and Fundunesp (Foundation for the development of unesp). The author received financial support to implement this work.

References

[1] X. Hongzhen and Z. Guosun, “Retracted: Specification and verification of dynamic evolution of software architectures,” *Journal of Systems Architecture*, vol. 56, no. 10, pp. 523 – 533, 2010.

[2] P. Maes, “Concepts and experiments in computational reflection,” *SIGPLAN Not.*, vol. 22, no. 12, pp. 147–155, Dec. 1987.

[3] I. R. Forman and N. Forman, *Java Reflection in Action (In Action series)*. Greenwich, CT, USA: Manning Publications Co., 2004.

[4] G. Coulson, G. Blair, and P. Grace, “On the performance of reflective systems software,” in *Performance, Computing, and Communications, 2004 IEEE International Conference on*, 2004, pp. 763 – 769.

[5] F. J. Affonso, “Metodologia para desenvolvimento de software reconfigurável apoiada por ferramentas de implementação: uma aplicação em ambiente de execução distribuído e reconfigurável,” Tese de doutorado, EESC/USP, maio 2009.

[6] J. Whitehead, “Collaboration in software engineering: A roadmap,” in *2007 Future of Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 214–225.

[7] E. Y. Nakagawa, F. C. Ferrari, M. M. Sasaki, and J. C. Maldonado, “An aspect-oriented reference architecture for software engineering environments,” *Journal of Systems and Software*, vol. 84, no. 10, pp. 1670 – 1684, 2011.

[8] F. J. Affonso and E. L. L. Rodrigues, “Reflecttools: Uma ferramenta de apoio ao desenvolvimento de software reconfigurável,” *Revista Brasileira de Computação Aplicada*, vol. 3, no. 2, pp. 73–90, 2011.

[9] P. J. Clemente, J. Hernández, J. M. Conejero, and G. Ortiz, “Managing crosscutting concerns in component based systems using a model driven development approach,” *Journal of Systems and Software*, vol. 84, no. 6, pp. 1032 – 1053, 2011.

[10] E. Borde, G. Haïk, and L. Pautet, “Mode-based reconfiguration of critical software component architectures,” in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE ’09. 3001 Leuven, Belgium, Belgium: European Design and Automation Association, 2009, pp. 1160–1165.

[11] H. Gomaa and M. Hussein, “Software reconfiguration patterns for dynamic evolution of software architectures,” in *Software Architecture, 2004. WICSA 2004. Proceedings. Fourth Working IEEE/IFIP Conference on*, june 2004, pp. 79 – 88.

[12] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin, “Aspect-oriented programming,” in *ECOOP’97 Object-Oriented Programming*, 1997.

[13] E. Tanter, R. Toledo, G. Pothier, and J. Noyé, “Flexible metaprogramming and aop in java,” *Sci. Comput. Program.*, vol. 72, no. 1-2, pp. 22–30, Jun. 2008.

[14] X. Chen, “Extending rmi to support dynamic reconfiguration of distributed systems,” in *Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on*, 2002, pp. 401 – 408.

[15] B. J. Williams and J. C. Carver, “Characterizing software architecture changes: A systematic review,” *Information and Software Technology*, vol. 52, no. 1, pp. 31 – 51, 2010.

[16] A. Navasa, M. A. Pérez-Toledano, and J. M. Murillo, “An adl dealing with aspects at software architecture stage,” *Information and Software Technology*, vol. 51, no. 2, pp. 306 – 324, 2009.

[17] A. L. Santos, K. Koskimies, and A. Lopes, “Automating the construction of domain-specific modeling languages for object-oriented frameworks,” *Journal of Systems and Software*, vol. 83, no. 7, pp. 1078 – 1093, 2010.

[18] R. Kazman, L. Bass, and M. Klein, “The essential components of software architecture design and analysis,” *Journal of Systems and Software*, vol. 79, no. 8, pp. 1207 – 1216, 2006.

A Variability Management Method for Software Configuration Files

Hiroaki Tanizaki, Toshiaki Aoki, Takuya Katayama
Japan Advanced Institute of Science and Technology
1-1 Asahidai, Nomi, Ishikawa, Japan
Email: {tani-h, toshiaki, katayama}@jaist.ac.jp

***Abstract*—Configuration files of software systems should be made correctly when software systems are operated. However, it is hard to figure out syntax and constraints of configuration files due to their complexity and size. Therefore, it becomes hard to make correct configuration files.**

We focus on two problems. One is syntax error that configuration file description is invalid. The other is semantic error that required functions do not perform due to mismatch between requirements and configuration file description. One solution to prevent these errors is to manage variability of configuration files. We propose a method which manages variability of configuration files. In particular, we use a model which organizes variability and constraints of configuration file description. Requirements and dependencies between requirements and organized information are also included in the model. The model is described by using the feature diagram. By using the feature diagram to describe the model, it becomes possible to check consistency between the model and a configuration file. Our method deals with detection and correction of errors of configuration files.

I. Introduction

Configuration files take an important role when software systems are operated because functions and performances of software systems depend on configuration file descriptions. When software systems are developed, specifications or source code are verified in order to check whether software systems have no error. Besides it is important to verify whether configuration files are correct. If configuration files have errors, software systems do not operate or functions which are needed by users do not execute. Errors of configuration files are syntax and semantic error [7]. One of causes of syntax error is scale and complexity of configuration files. Semantic error means that mismatch between user's purpose and configuration file descriptions. User's purpose means functions which are needed by users and is a kind of requirement for configuration files. One of causes of semantic error is that correspondence relations between configuration file descriptions and requirements are not clear. In order to make correct configuration files, variability of configuration files should be managed. Variability consists of syntax and correspondence relations between requirements. However, it is not easy to manage variability of configuration files because the complexity of configuration files is becoming higher and higher according to the increase of their size[2]. Besides, errors have to be corrected, but error correction is also not easy for users and administrators.

The purpose of our research is to propose a method which supports users and administrators to make correct configuration files. In order to make correct configuration files, our method detects syntax and semantic errors and corrects them. In particular, syntax and candidate of configuration file description and correspondence relations between requirements are modeled, and our method checks consistency between the model and a configuration file.

In this paper, we show our approach in the following way. In Section II, we introduce configuration files. Section III shows outline of our approach. In Section IV, we propose a model which organizes configuration file descriptions and requirements. In Section V, we explain a method for support making configuration files. Section VI shows an experiment using our approach. In Section VII, we discuss about the effectiveness of our approach. Section VIII shows related works. In Section IX, we conclude this paper.

II. Configuration files

Configuration files are widely used in software systems such as operating system, middleware software and application software. We focus on configuration files of servers. Configuration files of servers are directly made by users or administrators and should be made correctly.

Configuration files are used to customize configuration and functions of software systems. So, reliability of configuration files is important to operate software systems, and should be verified. Although there are some formats for configuration files, basically combinations of setting item and parameters are written. Each setting item has candidates of parameter. Besides, a combination of setting item and parameter may need other specific combinations in order to perform functions.

A. Requirements for configuration files

Users and administrators have their own purposes when they make configuration files. Their purposes mean that specific functions are performed. Therefore, we consider that purposes are requirements for making configuration files.

For server software, configuration which is appropriate to application which performs on the server is needed. For example, when administrators make configuration files of web servers and application servers which are used for a web application execution environment, administrators should configure functions for application which is executed on the environment. Therefore, it is possible to deal with purpose for configuration files as functions of application. By relating purpose for configuration files and functions of application, it becomes possible to specify what combinations of setting item and parameter are required for execution of application.

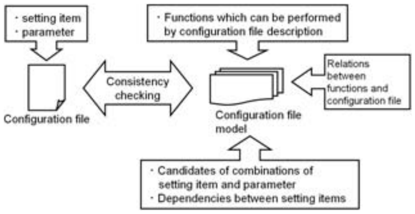
B. Errors of configuration files

We focus on syntax error and semantic error of configuration files. Syntax error means that combinations of setting item and parameter are invalid, or dependencies between combinations are not satisfied. Semantic error means that configuration files do not meet requirements for configuration files. Factors of configuration file error are size and complexity of configuration files. Therefore, it is needed to solve these factors in order to make error-free configuration files.

III. Approach

We take an approach to manage variability of configuration files. Configuration files have variability in syntax and correspondence relations between requirements. Variability of syntax means combinations of setting item and candidate of parameter. Variability of correspondence relations between requirements and configuration files means that requirements also have variability and what combinations of setting item and parameter are required in order to perform requirement.

Our approach consists of using a model and a consistency checking method. We propose configuration file model which organizes variability of configuration files. In particular, configuration file model represents candidates of combinations of setting item and parameter, constraints on making configuration files, requirements, and correspondence relations between requirements and configuration files. In order to check semantic error, we consider that requirements and correspondence relations between requirements and configuration files should be organized. Consistency checking method checks consistency between configuration file model and configuration files. Concept of our approach is shown in Fig. 1.



・setting item
・parameter
Configuration file
Consistency checking
・Functions which can be performed by configuration file description
Configuration file model
Relations between functions and configuration file
・Candidates of combinations of setting item and parameter
・Dependencies between setting items

Fig. 1. Approach of Our Method

In order to clarify structure of configuration file model, configuration file model is described using the feature diagram [5] [3]. Requirements and configuration files are described by using the feature diagram. Correspondence relations between requirements and configuration files are described by correspondence relations between feature diagram of requirements and feature diagram of configuration files. In our method, a diagram that configuration file model is described by the feature diagram is called configuration model. Besides, we use Alloy[1][4] in order to formalize and check the feature diagram. The details of configuration model and consistency checking method are shown in our previous work [9].

IV. Configuration File Model

A. Elements of configuration and relations

Basic elements of configuration file model are setting item, parameter and requirement. Setting item and parameter are specific elements of each configuration file. Requirement is function of application. Requirement should be considered in order to specify what combination of setting item and parameter are described in a configuration file. Configuration file model includes following relations between basic elements.

- Combination of setting item and parameter
  This combination shows valid candidates of parameter for each setting item.
- Dependency between specific combinations of setting item and parameter
  This dependency shows that some combinations of setting item and parameter may be required to describe other combination.
- Relation between requirements
  This relation shows that some requirements may be required to perform other requirement.
- Relation between requirement and combinations of setting item and parameter
  This relation shows candidates of combinations of setting item and parameter for execution of a requirement.

The concept of configuration file model is shown in Fig. 2.

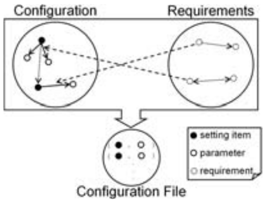


Fig. 2. Concept of Configuration File Model

B. Definition of Configuration File Model

The definition of configuration file model (CFM) is as follows.

CFM = (PN, PV, R, CR, Conf, Rel, RRel, Req)

The explanation of CFM is as follows. PN is set of names of setting item. PV is set of parameters. R is set of names of requirement. CR shows that what requirements are chosen by users or administrators for configuration file description. $Conf$ means PN $\rightarrow 2^{PV}$, and shows candidates of combination of setting item and parameter. Rel is subset of ((PN $\times$ PV) $\times$ (PN $\times$ PV)), and shows relations between combinations of setting item and parameter. $RRel$ is subset of R $\times$ 2^R, and shows relations between requirements. Req means R $\rightarrow 2^{(PN \times PV)}$, and shows combinations of setting item and parameter that corresponding to a requirement.

C. Correctness of configuration files

We define correctness of configuration files in order to check configuration files. First, we define a configuration file which is represented using configuration file model.

$\mathrm{CFile} = \{\ (\ n,\ v\ )\ |\ n \in \mathrm{PN},\ v \in \mathrm{PV}\ \}$

A configuration file (CFile) is defined as set of combination of setting item and parameter.

Second, we define four properties in order to define correctness of configuration files.

- constraint on combination of setting item and parameter
  $\mathrm{s\_conf} \Leftrightarrow \forall(a, b) \in \mathrm{CFile}.\ b \in \mathrm{Conf}(a)$
  This constraint means that combinations of item and parameter which are written in a configuration file have to be defined in configuration file model.
- constraint on dependency between combinations of setting item and parameter
  $\mathrm{s\_rel} \Leftrightarrow \exists\ x\ y.\ (x, y) \in \mathrm{Rel} \land x \in \mathrm{CFile} \rightarrow y \in \mathrm{CFile}$
  This constraint means that if a combination of setting item and parameter is written in a configuration file and requires other combinations, required combinations have to be written in the configuration file.
- constraint on relation between requirements
  This constraint means that if an element of requirement is chosen and requires other elements of requirement, required elements have to be chosen.
- constraint on relations between requirement and configuration file
  $\mathrm{s\_req} \Leftrightarrow \forall r \in \mathrm{CR}.\ \mathrm{Req}(r) \in \mathrm{CFile}$
  This constraint means that if a requirement is chosen and relates to combinations of setting item and parameter, the combinations of setting item and parameter have to be written in a configuration file.

Correctness of CFile is defined that CFile satisfies above properties.

V. Method for making configuration files

The workflow of our method is shown in Fig. 3.

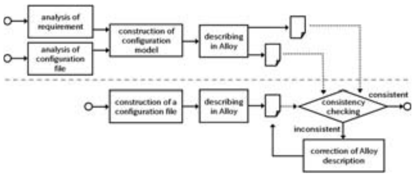

analysis of requirement
analysis of configuration file
construction of configuration model
describing in Alloy
construction of a configuration file
describing in Alloy
consistency checking
consistent
inconsistent
correction of Alloy description

Fig. 3. Workflow of the Method

We already defined configuration file model and correctness of configuration files. Properties which are used in order to prevent configuration file errors are organized in configuration file model. However, the properties are not easy to apply check configuration files because there is a gap between configuration file model and configuration file description. One solution is that to describe configuration file model and configuration files in common description method. Therefore, we use the feature diagram to describe configuration file model and configuration files. Besides, constraints of the feature diagrams are used to check correctness of configuration file.

A. Configuration model

Configuration model is a model that configuration file model is described using the feature diagram. Configuration model consists of requirement model which is a feature diagram of requirement, config model which is a feature diagram of configuration file, and correspondence relations between feature diagrams.

A configuration model is described as following method. Requirement model is described by analyzing commonality and variability of functions of application. Config model is described by analyzing candidates of combinations of setting item and parameter. Combinations of setting item and parameter have some patterns. So, we analyzed syntax of httpd.conf, which is configuration file of Apache web server, and extracted five patterns of combinations. The patterns are as follows.

- Pattern 1. setting item needs one parameter which is selected from set of candidates of parameter.
- Pattern 2. setting item needs some parameters which are selected from set of candidates of parameter.
- Pattern 3. setting item needs two parameters which are selected from different set of candidates of parameter.
- Pattern 4. setting item needs one parameter which is selected from candidates of parameter and some parameters which are selected from different set of candidates of parameter.
- Pattern 5. setting item does not need parameter.

The above patterns are used for constraints when combinations of setting item and parameter are chosen. Besides, the patterns are described using feature diagram. In order to describe the patterns, our method uses Mandatory, Optional, Alternative and Or relations of the feature diagram. Each pattern is described as follows.

- Pattern 1 is described by Alternative relation.
- Pattern 2 is described by Or relation.
- Pattern 3 is described by two Alternative relations.
- Pattern 4 is described by Alternative and Or relation.
- Pattern 5 is described only parent.

Additionally, there are dependency relations of combinations of setting item and parameter. These relations are able to describe by composition rules of the feature diagram.

An example of configuration model is shown in Fig. 4. In Fig. 4, the upper side diagram is requirement model and the under side diagram is config model. Dotted line arrows from requirement model to config model represent correspondence relations between requirement model and config model.

B. Error detection

We deal with checking correctness of configuration files as checking validity of chosen features from configuration model. Therefore, our consistency checking method checks whether set of features which are chosen from configuration model is an instance. An instance means a set of features which satisfies constraints of the feature diagram. In order to detect

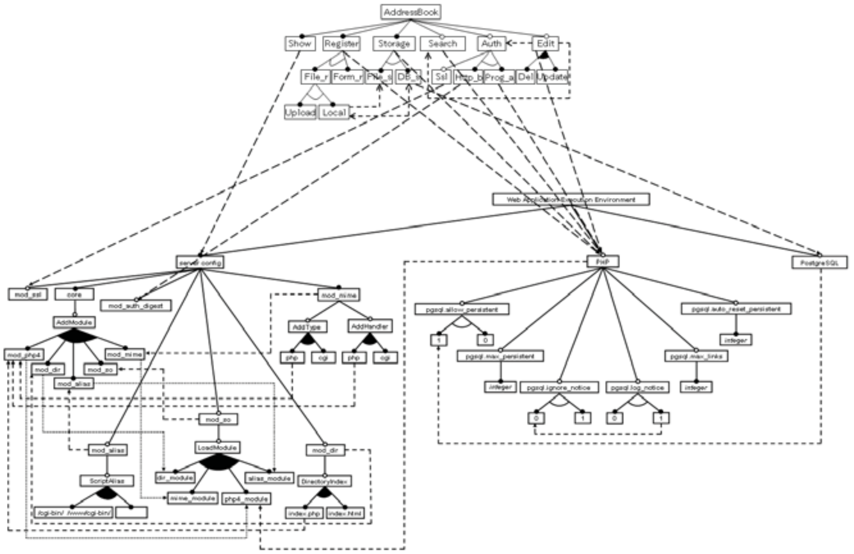


Fig. 4. Configuration Model

error, feature selections which do not satisfy each constraint should be clarified. For example, if a combination of setting item and parameters is described by Alternative relation and some parameters are chosen, this feature selection is invalid. If feature selection is invalid, it is an error.

Checking correspondence relations between requirement model and config model is used to check whether configuration files meet execution of requirements. Therefore, consistency checking of correspondence relations checks whether features which are related to an instance of requirement model are included in an instance of config model. It is an error when features which are required by the instance of requirement model are not included in the instance of config model.

C. Error correction

In our method, error correction means to obtain an instance when there is no instance of configuration model. In particular, error correction consists of detection of causes of errors and correction of them. For example, if a combination of setting item and parameters is described by Alternative relation, except for choosing setting item and one parameter, the other choosing is invalid. In particular, choosing no setting item and one parameter or choosing no setting item and some parameters are causes of error. In the case of the above example, choosing setting item is the correction method when no setting item and one parameter are chosen, and choosing setting item and one parameter from candidates is the correction method when no setting item and some parameters are chosen. In this way, causes of error are defined and correction methods for each cause are specified.

Correction method of correspondence relations between requirement model and config model is changing configuration files in order to satisfy requirements. Correction of configuration file is to add lacking features which are needed to perform requirement to an instance of config model.

Those correction methods are not enough to correct feature selection because only adding or deleting features may become causes of other errors. Therefore, we implemented a mechanism which obtains an instance in Alloy.

D. Consistency checking using Alloy

We use Alloy to check configuration model automatically. Therefore, structure and constraints of the feature diagram are described by Alloy description, and detection and correction method for errors are implemented in Alloy.

Alloy is a structural modeling language based on firstorder relational logic for expressing complex structural constraints and behaviors, and is described using finite sets and relations. Basic notations for describing alloy models are as follows.

- sig : A signature introduces a set of atoms
- fact : Always assumed constraints
- fun : A function to a relation from another relation
- pred : A true-false judging of a relation
- run : To search for an instance of a predicate ($pred$)

An Alloy model consists of sig and relations of sig, which are described in fact. Operations are described using fun. Conditions are described using pred. These described things are verified using the run command. The Alloy Analyzer which is a tool to verify Alloy models is provided. The Alloy Analyzer finds the Alloy models which satisfy a given specification of Alloy models automatically in scopes of sig.

1) Encoding of feature diagram: The structure of feature diagram is relations between features. Thus, name of each feature is enumerated, and relations between features are named and each relation is related to feature names which have relation between them. Constraints of the feature diagram are defined by predicate in Alloy.

2) Consistency checking: Consistency checking method for the feature diagram checks whether chosen features satisfy constraints of the feature diagram. In this method, constraints means patterns which are shown in section V.A. In particular, predicates, where constraints are described, are checked. Correction method of feature selection compares causes of errors with chosen features in order to detect causes of errors. The Alloy Analyzer automatically searches that what features should be chosen based on each correction method.

Consistency checking method for correspondence relations between requirement model and config model checks whether an instance of config model satisfies an instance of requirement model. This check is implemented by set operators of Alloy. In particular, Alloy checks whether a set of combination of setting item and parameter which is required by an instance of requirement model is subset of an instance of config model. Correction method of correspondence relations also searches instances. The Alloy Analyzer searches an instance of config model which includes the existing instance and features which are needed for correction.

VI. Experiment

In the experiment, we apply our method to a Web application system which consists of an application and an execution environment. We deal with an address book application as application. Execution environment consists of some software, and each software use configuration file.

A. Preparing for checking

1) Analysis of application: An address book application has some functions such as showing, registration and editing. Those functions have variations. There are some methods for data registration. For example, files which contain data are uploaded to a server, or users input data into a form on a web page. Files or database are used in order to save data. Editing data is data correction or deleting. In this way, functions of application are analyzed. Analysis of configuration file description for operation of each requirement is discussed later.

2) Analysis of configuration files: At least three kinds of software, web server, server-side programming language environment, and database server are needed. So, each of software's configuration file should be analyzed. For web server, we assume that Apache is used. Basic configuration for web server is needed. Besides, configuration for server-side programming or cgi is needed. For server-side programming environment, we assume that PHP is used. Configuration for connecting to database server is needed. For database server, we assume that PostgreSQL is used. Basic configuration for database server is needed.

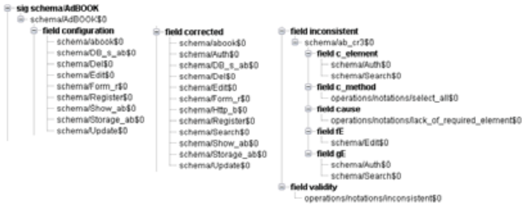
3) Analysis of correspondence relations: Which combinations of setting item and parameter are needed for executing each function is analyzed. Web server is needed in order to show contents of address book, CGI or PHP are needed in order to execute functions such as registration and deleting. Database server is needed in order to save data.

4) Describing configuration model: A configuration model is described based on analysis results. The configuration model is shown in Fig. 4. The configuration model has to be described in Alloy in order to check it. However, Alloy description is omitted this time.

B. Consistency checking

In our method, feature diagrams and correspondence relations between instances of them are checked separately. Because, it makes no sense to check correspondence relations, if there is no instance of requirement model or config model.

1) Feature diagram: We choose some features from the requirement model and check it using the Alloy Analyzer. The result is shown in Fig. 5.

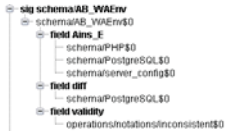


```
sig schema/AdBOOK
  schema/AdBOOK$0
    field configuration          field corrected             field inconsistent
      schema/abook$0               schema/abook$0              schema/ab_cr3$0
      schema/DB_s_ab$0             schema/Auth$0                 field c_element
      schema/Del$0                 schema/DB_s_ab$0                schema/Auth$0
      schema/Edit$0                schema/Del$0                    schema/Search$0
      schema/Form_r$0              schema/Edit$0                 field c_method
      schema/Register$0            schema/Form_r$0                 operations/notations/select_all$0
      schema/Show_ab$0             schema/Http_b$0               field cause
      schema/Storage_ab$0          schema/Register$0               operations/notations/lack_of_required_element$0
      schema/Update$0              schema/Search$0               field fE
                                   schema/Show_ab$0                schema/Edit$0
                                   schema/Storage_ab$0           field gE
                                   schema/Update$0                 schema/Auth$0
                                                                   schema/Search$0
                                                             field validity
                                                               operations/notations/inconsistent$0
```

Fig. 5. Result of Checking (Feature diagram)

In Fig. 5, "field configuration" shows set of chosen features. "field validity" shows presence or absence of error. "field inconsistent" shows causes of error and correction methods. "field corrected" shows a candidate of corrected chosen features. As the result of this checking, errors are detected, and causes of errors and correction methods are shown.

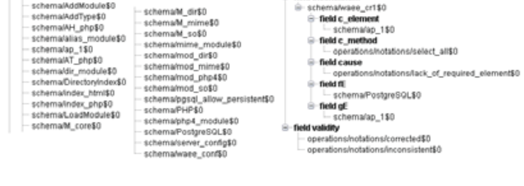
2) Correspondence relations: We assume that there are instances of requirement model and config model. A result of checking of correspondence relations is shown in Fig. 6.



```
sig schema/AB_WAEnv
  schema/AB_WAEnv$0
    field Ains_E
      schema/PHP$0
      schema/PostgreSQL$0
      schema/server_config$0
    field diff
      schema/PostgreSQL$0
    field validity
      operations/notations/inconsistent$0
```

Fig. 6. Result of Checking (Correspondence relation)

In Fig. 6, "field Ains_E" shows what features are required by the instance of requirement model. "field diff" shows that what features are lacking in the instance of config model. "field validity" shows presence or absence of error. As the result of checking, there is one error. The error means that "PostgreSQL" is not included in the instance of config model although the application requires database. So, correction method applies to the instance of config model. The result of correction is shown in Fig. 7.



```
field corrected                                                         field inconsistent
  schema/AddHandler$0                                                     schema/waee_cr1$0
  schema/AddModule$0          schema/M_dir$0                                field c_element
  schema/AddType$0            schema/M_mime$0                                 schema/ap_1$0
  schema/AH_php$0             schema/M_so$0                                 field c_method
  schema/alias_module$0       schema/mime_module$0                            operations/notations/select_all$0
  schema/ap_1$0               schema/mod_dir$0                              field cause
  schema/AT_php$0             schema/mod_mime$0                               operations/notations/lack_of_required_element$0
  schema/dir_module$0         schema/mod_php4$0                             field fE
  schema/DirectoryIndex$0     schema/mod_so$0                                 schema/PostgreSQL$0
  schema/index_html$0         schema/pgsql_allow_persistent$0               field gE
  schema/index_php$0          schema/PHP$0                                    schema/ap_1$0
  schema/LoadModule$0         schema/php4_module$0                        field validity
  schema/M_core$0             schema/PostgreSQL$0                           operations/notations/corrected$0
                              schema/server_config$0                        operations/notations/inconsistent$0
                              schema/waee_conf$0
```

Fig. 7. A Candidate of Corrected Instance

A candidate of corrected instance is shown in left side of Fig. 7. In right side of Fig 7, causes of inconsistencies and correction methods of them are also shown.

VII. Discussion

Our method checks configuration files based on the configuration model and deals with syntax error and semantic error. As the result which is shown in Fig. 6, one error is detected. The cause of the error is that using database is not configured although address book application needs database. The important part is that even if configuration files do not have error, errors occur when configuration files are used in order to perform application. Checking semantic error of configuration files statically is one merit of our method.

It is not easy to understand syntax and constraints of configuration files although those are explained in manuals. In our method, syntax and constraints are described structurally by using the feature diagram. Besides, checking configuration files based on constraints of the feature diagram becomes possible. We consider that a method which organizes configuration files using the feature diagram is effective.

Configuration model describe five patterns of configuration file syntax. These patterns are extracted from syntax of httpd.conf, but the patterns do not cover all syntax. In particular, 180 setting items are available for httpd.conf and patterns cover 153 setting items. So, the patterns can deal with 85 percent of setting items of httpd.conf.

A configuration model may be changed when needed configuration files are changed because of change of application, or candidates and constraints of configuration file description are changed. This change means adding or replacing pattern descriptions and is not difficult. Users or administrators only have to follow the notations of the feature diagram.

Syntax checkers are enough to check syntax error because each syntax checker is specialized in specific configuration files. However, syntax checkers are not enough to check semantic error because semantic error is not a target. We focus on relations between requirements and configuration files, and model those relations in order to check.

VIII. Related work

A method which checks configuration files based on rules is proposed [7]. Rules mean dependencies between modules or formats for data exchange. A method which checks validity of parameters of configuration files is proposed [8]. In this method, size, range and dependency of parameters are identified as constraints, and parameters are checked based on them. Our method takes the similar approach which was proposed in [7] [8]. Errors which can be detected by those methods depend on rules, constraints or model. That is to say, it is important what condition use to detect errors. On the other hand, using rules and constraints is not suitable for checking semantic errors because it is difficult to deal with requirements.

A method which deals with dependencies of configuration attributes of software stack is proposed [6]. Authors proposed Configuration Map (CM) which specifies dependencies and order of configuring of configuration attributes, and relations between configuration attributes and errors. By using CM, managing configuration attributes becomes easily and CM is effective for troubleshooting. As shown in [6], it is helpful to model complexity in order to make configuration. Our method deals with not only variability of configuration files but variability of application, and models them in order to make suitable configuration files for application.

An automatic generation method for configuration files of web system which consists of Web server, application server and database, is proposed [10]. In this method, configuration files are generated by some scripts based on templates and parameters are detected using heuristic algorithm. Automatic generation is also a support method to make configuration files. However, it is not easy to specify combinations of setting item and parameter because descriptions of configuration files have flexibility. Therefore, our method checks configuration files which are made manually instead of automatic generation of configuration files.

IX. Conclusion

In this paper, we proposed a method which manages variability of configuration files in order to prevent errors of configuration files. We define the configuration file model which organizes basic elements of configuration files and relations of them. Besides, a configuration file model is described by using the feature diagram in order to check.

Verifying configuration files is important as well as software because the complexity and size of configuration files are increasing drastically. Some applications have their own syntax checkers for configuration files of them. However, only checking syntax is not sufficient to ensure that what we want to configure the applications is correctly described in the configuration files. Thus, in our approach, requirements for the configuration files are modeled as a configuration model and check whether they meet the requirements or not.

References

[1] Alloy website : http://alloy.mit.edu/alloy/

[2] Aaron B. Brown, Alexander Keller and Joseph L. Hellerstein, A Configuration Complexity Model and Its Application to a Change Management System, In Proceedings of the 9th International IFIP/IEEE Symposium on Integrated Management, pp. 631 - 644

[3] Krzysztof, C. and Ulrich, E. : Generative Programming : Methods, Tools, and Applications, Addison-Wesley Pub (Sd), 2000, ISBN 978-0201309775.

[4] Jackson, D : Software Abstractions: Logic, Language, and Analysis, MIT Press, 2006, ISBN 0-262-10114-9.

[5] Kang, K. C., Cohen, S. G., Hess, J. A., Novak, W. E. and Peterson, A. S. : Feature-Oriented Domain Analysis (FODA) Feasibility Study, Technical Report CMU/SEI-90TR-21, Software Engineering Institute, Carnegie Mellon University, 1990.

[6] Kalapriya Kannan, Nanjangud C. Narendra and Lakshmish Ramaswamy, Managing Configuration Complexity during Deployment and Maintenance of SOA Solutions, 2009 IEEE International Conference on Services Computing, pp. 152-159

[7] Rajesh Kalyanaraman, A Rule based static configuration validation technique in an Autonomous Distributed Environment, Second International Conference on Systems, pp 53

[8] Emre Kycyman and Yi-Min Wang, Discovering Correctness Constraints for Self-Management of System Configuration, Proceedings of the 1st International Conference on Autonomic Computing, pp. 28-35

[9] Hiroaki Tanizai. and Takuya Katayama. : Formalization and Consistency Checking the Changes of Software System Confgurations Using Alloy, 15th Asia-Pacific Software Engineering Conference, pp.343-350.

[10] Wei Zheng, Ricardo Bianchini and Thu D. Nguyen, Automatic configuration of internet services, Proceedings of EuroSys 2007, pp. 219-229

Tool Support for Anomaly Detection in Scientific Sensor Data

Irbis Gallegos
The University of Texas at El Paso
Department of Computer Science
El Paso, USA
irbisg@utep.edu

Ann Gates
The University of Texas at El Paso
Department of Computer Science
El Paso, USA
agates@utep.edu

***Abstract*— Environmental scientists working on understanding global changes and their implications for humanity, collect data in near-real time at remote locations using a variety of instruments. As the amount and complexity of collected data increases, so does the amount of time and domain knowledge required to determine if the collected data are correct and if the data collection instruments are working correctly. The Sensor Data Verification (SDVe) tool allows scientists to detect anomalies in sensor data. SDVe evaluates if scientific datasets, which can be provided at near -real time or from file repositories, satisfy reusable data properties specified by scientists.**

Keywords- Big Data; Cyberinfrastructure; Data Quality E co-Informatics; Software Engineering.

I. Introduction

Scientists collect and analyze large amounts of data to determine the causes of changes in ecological ecosystems. Scientist use Eddy covariance (EC) towers [1] to collect measurements needed to understand such ecological changes. In particular, scientists use measurements taken by EC towers to monitor carbon dioxide (CO_2), water balance (H_2O), energy balance (irradiance), and other meteorological measurements such as temperature and atmospheric pressure.

Anomaly detection in eddy covariance data must not only identify instrument errors and problems with the sensors, but also evaluate how closely conditions fulfill the theoretical assumption underlying the method [2]. Anomaly detection must be done in real time or shortly after the measurements to minimize data loss by reducing the time to detect and fix instrument problems. The quality control procedures, instrument malfunctions, maintenance and calibration periods often remove 20 to 40% of the data. Efficient anomaly detection is an outstanding problem that is incompletely fulfilled in most of the FLUXNET networks [2].

In most cases, scientists manually evaluate eddy covariance data by using a variety of time consuming methods and a variety of customized software that has to be constantly modified and recompiled. In addition, the data evaluation is highly dependent on the expertise of the scientist, however, such knowledge is typically not captured nor reused. The sensor data verification (SDVe) tool mitigates the aforementioned limitations by allowing scientists to automatically identify anomalies produced by environmental events and equipment malfunctioning in scientific sensor datasets collected at near-real time or extracted from file repositories. SDVe evaluates whether a dataset satisfies a set of formally specified data properties. SDVe identifies anomalies, i.e., a deviation from an expected value, due to environmental variability or instrument malfunctioning, and raises alarms whenever anomalies are detected. SDVe does not require source code recompilation and allows expert-defined data properties to be reused to verify the datasets.

Section 2 provides the background on the software engineering techniques adapted and extended by SDVe. Section 3 describes the data property specification approach used by scientists to specify the data properties that are input to SDVe. Section 4 explains the SDVe tool. Section 5 describes the experimental setup and the results of using SDVe to detect anomalies in Eddy covariance data, and section 6 discusses some of the lessons learned from the experiment. Finally section 7 described some efforts related to SDVe, followed by the concluding remarks in Section 8.

II. Background

SDVe uses software engineering techniques, which have been used to provide assurance for critical software systems, to detect anomalies in scientific sensor datasets. This section provides the background information associated to such software engineering techniques.

A. Property Specification

The specification and pattern system (SPS) [3] was introduced to assist practitioners to formally specify software and hardware properties.

In the SPS, a specification consists of scopes and patterns. Scopes define the portion of a program over which the property holds. Patterns describe the structure of specific behaviors and define relationships between propositions. Propositions are used to represent Boolean expressions that are evaluated over the program execution.

SPS patterns are divided into two groups: *Occurrence* and *Order* patterns. *Occurrence* patterns deal with single event or condition and specify the rate at which that condition or event occurs. *Order* patterns relate two conditions or events and specify the order at which they occur. In this context, *conditions* are propositions that hold in one or more consecutive states. *Events* are instants at which a proposition changes values in two consecutive states.

The SPS supports mapping to several formalisms including discrete-time [4].

B. Runtime Monitoring

Run-time monitoring systems [5] and model checkers [6] can be used to check that critical software systems are functioning as expected with respect to a set of properties. Run-time monitoring allows practitioners to observe the behavior of a system and determ ine if it is consistent w ith a given specification.

A run-time monitor takes an executing software system and a specification of software properties and checks that the execution meets the properties. In most run -time monitoring frameworks, the app lication's code is instrum ented, i.e., the source code is injected at points of interest with checks representing the specified property. Usually, the application to be monitored has to be recompiled to include the checks, and monitoring is performed at the c ode level to ensure that the code is behaving as intended. Delgado et al. [6] have compiled a taxonomy of run-time monitoring frameworks.

Model checkers [7] are a formal technique for verifying finite-state concurrent system s and relies on building a finite model of a system and an algorithm that automatically traverses the system model to verify if a desired property (or a set of properties) holds in the model.

III. DATA PROPERTY SPECIFICATION

A. Data Property Specification

Gallegos et. al. [8] conducted a literature survey over scientific projects that collect sensor data to identify categories of data properties as captured by scientists. Based on the findings, a categorization of data properties was constructed.

The classification divided data properties into two major types: *experimental readings* and *experimental conditions*. *Experimental readings* properties specify expected values and relationships related to field data . *Experimental conditions* properties specify expected instrument behavior and relationships by defining examining attributes (e.g., voltage) and instrument functions based on readings. The data property categories are further subdivided based on the number of sensor dataset readings or streams and depending on whether the data categories depend on time or not.

B. Data Specification and Pattern System

The data property specification and pattern system D-SPS [8] uses and modifies the concepts from the SPS and its timing extension [9]. D-SPS properties are defined using one or more scopes, a pattern, and one to four Boolean statements. D-SPS properties are of type Boolean and are evaluated over datasets, i.e., a finite sequence of sensor data readings. For this w ork, a sensor data reading, δ, is a pair, $< \alpha, \beta>$, where α denotes a unique indexing value, and β denotes a sensor reading value. The unique indexing value can be either a tim estamp or a floating- point value. A scope is a subsequence(s) of a dataset of interest to be evaluated. A property pattern is a high-level abstraction describing how one or more Boolean statements are evaluate over a number of scopes. D-SPS uses the data property categorization to determ ine the applicable patterns that can be applied to a restricted number of scopes also restricted by the data property categorization. Boolean statements express data proper ties, which are defined using mathematical relational operators that are applied to data readings and relationships between data readings.

The SPS definitions included quantitative, not tim e-constrained, patterns and timed-constrained patterns. For timed patterns, units of tim e are assum ed to be ordered tim e stamps in datasets, w ith equal constant tim e resolution, T he scientist is assumed to map the unit of time used in the dataset to the unit of time used in the data property specification.

C. Data Property Specification Tool Support

The data property categorization and the D-SPS resulted in development of Data Property Specification (DaProS) [10], a scientist-centered prototype tool that uses the categorization to assist the user in specifying a data property. Through a series of guiding questions and selections, the user identifies the appropriate category and enters the scope, pattern, and Boole an statements needed to specify th e property, and the tool yields the appropriate specification as w ell as a disciplined natural language representation of the specification for validation purposes. The specification is generated as an XM L representation to allow the specified properties to be exported.

IV. SENSOR DATA VERIFICATION TOOL SUPPORT

A. Overview

The sensor data verification (SDVe) prototype tool allows scientist to evaluate D aProS-generated data property specifications over scientific sensor data streamed at near real time or extracted from file repositories. The SDVe tool takes as input a property specification file and a dataset file and verifies that the data in the dataset files adheres to the property specified in the property specification fi le. SDVe automatically identifies anomalies in sensor data without having to conduct neither complex data analysis nor complete statistical analysis of time series, thus satisfying the needs of scientific communities in [11]. SD Ve raises alarm s whenever a data reading does not satisfy the specified data property.

SDVe supports reusable properties , and does not requires practitioners to recom pile any tool's source code. R eusable properties are of interests to scientists because data quality is always a combination of different levels of control and site-specific tests. The data quality w ill differ from one site to another because of the topography and this must be taken into account.

The SDVe is not intended to indicate the reasons for the occurrence of legitim ate environmental variability events and possible instrument pr oblems, but to indicate to the scientists the data readings that might require further analysis to determine the possible causes of the anomaly. Such tool design decision was based on the literature [2] that indicates t hat automated techniques cannot unequivocally pinpoint differences between an environm ental variability event and an instrument problem. Furtherm ore, subjective analyses combined with an autom ated approach do better than the automated program alone for unusual conditions; data flagged

This research effort is supported by National Science Foundation grants No. HRD-0734825 and CNS-0923442.

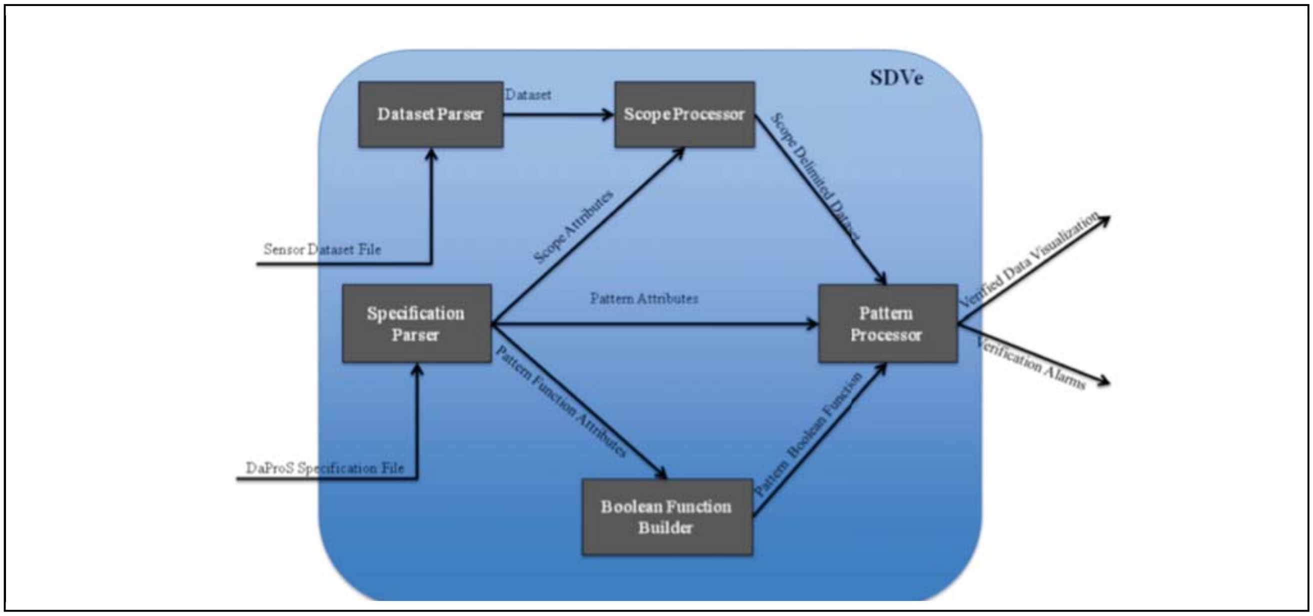


Figure 1. System overview of the SDVe prototype tool.

as suspicious and deemed as an environmental event after graphical inspection are often found to be the most unusual and interesting situations.

B. System Design

The system is composed of 5 components depicted in Figure 1. The *dataset parser* module encompasses a group of sensor dataset file parsers for supported data files. Even though SDVe is intended to support a number of different sensor data files, a dataset parser for such data file should be developed and placed in this module. However, this is the only change that has to be performed to the SDVe. The dataset parser reads a dataset file and transforms the dataset into an internal representation of the dataset that can be used by the scope processor to extract the data subsequences to be evaluated. The sensor data readings are stored as pairs indexed by an identifier, i.e. a unique time-stamp or numeric value associated to the sensor reading.

The *specification parser* module reads a DaProS-generated specification file and extracts the attributes needed to delimit the dataset scopes, to build the Boolean statements to be evaluated over the data, and to determine how to apply the pattern to the dataset.

The *scope processor* module uses the internal dataset representation populated by the *dataset parser* module and extracts the data subsequences over which the Boolean statements will be evaluated.

The *Boolean statement builder* module uses the Boolean statements attributes extracted by the *specification parser* module and determines the type of operation, the inputs, and the Boolean function code templates to evaluate the sensor readings in the scope(s). SDVe supports the Mathematical operators <, ≤, =, ≠, ≥, and >. Boolean connectives such as conjunction or disjunction are not supported, except for implication, which is used in some of the patterns to capture ordering between two data properties.

The *pattern processor* module uses the Boolean functions created by the *Boolean function builder* module, the scopes from the *scope processor* module and the pattern attributes from the *specification parser* module to evaluate the pattern. The *pattern processor* evaluates the Boolean functions over the scopes of sensor data as specified by the pattern's attributes, and raises alarms when the Boolean functions evaluations over the scopes do not satisfy the expected data property. To evaluate the Boolean statements over the scopes, SDVe interprets the DaProS specifications into code templates that evaluate the data. Each pattern code template in SDVe corresponds to a pattern in the D-SPS. The code templates have a predefined number of Boolean statements that can be evaluated by the pattern, and a predefined number of scopes that can be evaluated by each Boolean statement.

Once the patterns have been evaluated, SDVe generates two output files to document the violations of properties identified in the data.

The *verification summary* file includes the total results of the verification process over several sensor data files. The verification summary is used to provide an overall overview of the verification process. Each data property evaluated by SDVe is presented in the file along with a count of instances when sensor readings violate such property. The file also contains, a count of the total number of sensor reading evaluated, a count of the number of violations found, and a file detection rate calculated from the ratio between total violations found and the total number of checks. The *verification summary* file also provides an aggregation of the total time in milliseconds of the time needed to load the sensor reading files to be evaluated and the total time in milliseconds required to verify the sensor reading files.

The *verification* file includes the evaluated data properties along with the instances of the data that violate the data property.

V. Anomaly Detection in Eddy Covariance Data Experiment

A. EC Data Description

A series of experiments were conducted to determine if SDVe can be utilized to evaluate DaProS-specified data properties to find anomalies in sensor data. Upon further analysis of the data properties obtained from the literature survey in [8], it was determined that three data property types account for approximately 72.5% of the total number of data properties specified by scientists in the literature survey; *datum* properties (32.5%), i.e. properties that capture the behavior of a single data sensor reading, *datum relationship* properties (30.8%), i.e. properties that capture a relationship between two or more sensors, and *datum dependent instruments* (9.2%), i.e., properties that capture environmental data behavior effect on the data collection instruments. Based on these findings, an experiment was design to determine if SDVe can be used to identify anomalies for the most frequently specified data properties: *datum*, *datum relationship*, and *datum dependent instrument*.

A data property specification expert collaborated with expert scientists working with eddy covariance and biomesonet towers' CO2 data to develop a set of data properties of interest. The contributing scientists were working on building their first eddy covariance tower and were interested in capturing data properties extracted from sensor reference manuals [12] [13], climate and climatological variations in the research site literature [2], eddy covariance towers post-field data quality control literature [14] and their own expertise.

B. Experimental Setup

An initial experiment was conducted to validate the ability of the SDVe tool to detect anomalies. An eddy covariance error-free data file was randomly selected from a scientist-provided repository and seeded with a number of anomalies, based on a 95% confidence level calculation given the number of readings in the file, to evaluate a group of data properties of type *datum*, *datum-relationship*, and *datum dependent instrument*. The experiment identified all seeded anomalies and all events marked as anomalies were actually anomalies.

A second experiment was conducted to determine if SDVe can identify anomalies in Eddy Covariance sensor data and to illustrate how such anomalies can be identified and documented by cross referencing the results obtained from SDVe to existing metadata recorded during the data collection process. The experiment does not quantify the improvement in the overall quality of the data.

The scientists developed a matrix of all the relationships between sensors in the Eddy covariance tower of interest for the specific site. The matrix included raw sensor measurements and derived data aggregated at different temporal resolutions and as part of the combination of measurements from two or more sensors. The sensor relationship matrix consisted of approximately 118 sensor readings along with their associated relationships. In collaboration with scientist some of the sensor relationships from the matrix were used to create 23 data properties to be evaluated over the Eddy covariance datasets; the scientists specified properties of interest of type *Datum*, *Instrument*, and *Datum Relationship*. The properties were intended to capture anomalies in raw data at collection time. The sensor readings of interest were selected based on the relationship to other sensor readings and the derivation of aggregated values from them. In collaboration with scientists, data properties of interest were specified, refined, and validated using DaProS. The numeric thresholds used in the data properties were defined following scientific community's algorithms and protocols.

The Eddy Covariance (EC) data verified using SDVe were collected from July 06, 2010 to July 13, 2010 to capture EC summer behavior and from February 09, 2010 to February 16, 2010 to capture EC winter behavior. The sensors at the tower collected the EC data continuously, and a scientist manually split the data into 1-hour interval files to ease the verification process. 349 data files were evaluated for this work.

Two data property specification files were created, one for each season, and used to automatically evaluate individual data files according to the season to which they belonged. The data files and specification files were automatically inputted to SDVe to be evaluated. For each data file, the sensor data streams were extracted to separate data scopes according to the data property specifications. Then, the data scopes were evaluated by applying the specification Boolean statement and data pattern to every individual sensor reading in the scope. If the individual sensor reading in the scope did not satisfy the data pattern and Boolean statement, a flag was raised and stored to the verification file. Once the verification process had concluded, a verification summary file was generated aggregating the number of violations, i.e, anomalies, identified by each data property along with the aggregated processing times.

C. Results

SDVe performed a total of 219,800,854 evaluation calls of which 50,857,351 of the evaluation calls were captured as anomalies, approximately 23% . The evaluation process took approximately 20 hours to complete, of which approximately 1 hour was spent loading the files into the system and 19 hours were spent verifying the data. Assuming a data file takes 15 minutes on average to be manually processed and evaluated by a scientist manually processing the 349 files would take approximately 87 hours. SDVe automatically evaluates the data in one fourth of the time that it would take a scientist to manually evaluate the same amount of data.

For summer data, 124,958,406 evaluations calls took place, of which 24,417,791were of the evaluation calls were captured as anomalies, approximately 20%.

Datum properties identified the most anomalies (21,429,802), followed by *Data Relationship* (2,639,985), *Instrument* (348,004) and *Data Dependent Instrument* (0). The sensor datasets with the most anomalies included water vapor mass density (H_2O), atmospheric pressure (atm_press), carbon dioxide (CO_2), and temperature (Ts).

Once the datasets were evaluated using SDVe, the anomalies found by the tool were cross-referenced with the available data-collection process metadata compiled by the scientists in the field site and with historical meteorological data obtain from the *Weather Underground* website [15] to identify false positives. SDVe was able to identify environmental variability and instrument malfunctioning in the datasets. Abrupt changes in the number of violations detected in the summer season by the data properties correspond to rain events that occurred in the area. Specifically, rain events occurred in July 8, 2010 (07/08/2010) and July 11, 2010 (07/11/2010) represented as high concentrations and abrupt changes in the number of anomalies found at the time of the event occurrence. A similar rain event, occurred on February 10, 2010 (02/10/2010) during the winter season. Such rain events were identified as both, true environmental variability and as a cause for instrument malfunctioning. Rain events affected data properties monitoring environmental variability such has water vapor mass density (H_2O), carbon dioxide mass density (CO_2), vapor pressure (e_hmp), and data properties monitoring instrument functioning such as the automatic gain control (agc) flag.

The results were evaluated by an expert and it was determined that all of the environmental events captured by SDVe were true environmental events and no other environmental events occurred during that period of time. In addition, all detected equipment malfunctioning anomalies were associated to the detected environmental events as expected. Also, identified false negatives were determined to be mostly due to misrepresentation of environmental phenomenon in specifications due to seasonal and diurnal cycles.

VI. Discussion

In some instances, the efficiency of SDVe can depend on the quality of the specified properties. Some of the influencing aspects that can affect the results of the anomaly detection process are discussed in this section.

When defining data properties, scientist needs to consider subtleties in the specification such as in the operator used in the Boolean statements. For example, the scientists need to be aware of the distinction in the meaning associated with the "<" and the "≤" operators and their implications to the verification process.

Environmental events can only be characterized if various data properties associated with different sensors are violated during the event. For instance, solely looking at changes in temperature cannot identify a rain event, but changes to diagnostic flag and atmospheric pressure in addition to the temperature change can indicate such event.

Overlapping and conflicting properties can be determined either during the specification phase or the verification phase. Identifying the conflicts at the specification stage is difficult because it requires a deep understanding of the data being analyzed and requires the practitioner to keep track of the data properties and the relationship and side effects between them. Also, potential properties conflicts are harder to validate because there is no immediate way to quantitatively compare the expected output for the data properties that can isolate the conflict. If the conflicts are captured at the verification stage, the practitioner has access to the verification results and can cross-reference the results of the verification with the specified data properties and thus decide if properties conflict with each other.

Another challenge is to identify the most appropriate threshold to be used in a data property. Threshold values, when needed, are usually site specific, thus it is difficult to identify universal threshold values that could use by all of the scientific communities. Scientists must consider the intent of the property in addition to the site-specific information when selecting the thresholds for the data properties.

Scientists are interested in identifying the critical set of properties needed to identify environmental variability or instrument malfunctioning from the data. In this context, a critical set of properties is the group containing the minimum number of data properties that will allow a scientist to identify a weather or instrument feature in the data.

Data properties can also be used to assign confidence levels of quality to collected data. Data properties can be defined in terms of the diagnostic measurements in the data collection instruments that are affected by the conditions surrounding the experiment. Consider the clean window value (agc) from a 3-D Sonic Anemometer [12]. Typical clean windows values are between 55% and 65%. As dirt or water accumulates on the windows, or anywhere in the optical path, the agc value increases. Changes in the agc value can result from dust, pollen, vibrations, dew, and rain/snow [13]. Given that the higher the value of the agc, the higher the probability that the collected data results in bad/ not-expected data, the scientist can take advantage and specify data properties with confidence levels on them.

Data properties can be used to identify anomalies at different data granularities, i.e., the level of fineness to which a dataset is sub-divided. The challenge is to determine at which data granularity to verify the data such that the number of anomalies found is maximized. If the data is verified at the highest level, i.e., the smallest sub-division of a dataset, every data point can be examined and verified; however, it requires more computational power to do so. Lowering the data granularity relieves the computation power needed to verify the data, but can miss anomalies in the data. For this work, the data granularity used was the highest available --data measurements were taken every millisecond and stored in half hour files.

The effectiveness of SDVe depends on the precision with which the seasonal and diurnal cycles are modeled by the data properties. Due to differences in data types and data behavior according to seasonal and diurnal variability, in some cases it is necessary to define the data properties to be as specific as possible to an expected data behavior or time-related variability. For EC data, the expected data values measured by the sensors differ depending on the season, the diurnal cycle and the time of the day. Different data property sets, with specific data granularity and quantitative values, have to be built for specific parts of the season and diurnal cycles.

Data Properties can be specified to document scientific knowledge about processes or to identify anomalies in scientific sensor data. Data properties specified to detect anomalies in sensor data are typically specific about the sensor names and thresholds over which the data should be evaluated, can be interpreted and used to evaluate data by SDVe without further manipulation to the property, and can be used to document the scientific processes. Data properties that are specified for the sole purpose of documenting processes are typically general in their descriptions, e.g., only describe which sensor reading will be evaluated and how it will be used, but do not include the specific name or threshold values to be evaluated. Data properties for documenting processes might also include computational methods that need to be applied to data before the data can be evaluated. These types of data properties are not suitable for the current version of SDVe.

VII. Related Work

Some of the approaches that are frequently used to detect anomalies in sensor data are described in this section.

The Intelligent Outlier Detection Algorithm (IODA) [16] is a technique used to perform quality control on time series data. IODA uses statistics, graph theory, image processing and decision trees to determine if data are correct. The IODA algorithm compares incoming data, which are treated as images, to common patterns of failure. SDVe differs from this approach in that it is a scientist-centered approach in which the anomaly detection process is based on the expert scientific knowledge captured by the data properties.

Dereszynski & Dietterich [17] use a Dynamic Bayesian Network (DBN) [18] approach to automatic data cleaning for individual air temperature data streams. The DBN combines discrete and conditional linear-Gaussian random variables to model the air temperature as a function of diurnal, seasonal, and local trend effects. The approach uses a general fault model to classify different type of anomalies. SDVe differs from this approach in that no mathematical or logic knowledge has to be acquired by the scientists to verify their data as it is the case with the DBN.

EdiRe [19] is a software tool for eddy covariance and microclimatological measurement analysis. EdiRe is adaptable to most eddy covariance raw data formats and microclimate data, however it requires processing routines to be developed and redesigned to address the different areas associated with data analysis as opposed to SDVe that do not required any software implementation or recompilation.

VIII. Summary

Scientists collecting large amounts of complex environmental sensor data need to be assured that their data are correct in a timely fashion. The SDve prototype tool adapts software engineering techniques to allow scientists to verify sensor datasets against predefined formally specified data properties. SDVe automatically verifies datasets and raises flags whenever anomalies occur in the datasets. SDVe has been used to identify anomalies in eddy covariance data.

Acknowledgment

The authors would like to thank Dr. Deanna Pennington, Dr. Craig Tweedie and Aline Jaimes for their invaluable input towards this work.

References

[1] A. Jaimes. Presentation, Topic: "Defining Properties for Eddy Covariance Data," Cybershare-Center, The University of Texas at El Paso, 2010.

[2] X. Lee, W. Massman, B. Law. *Handbook of Micrometeorology: A Guide for Surface Flux Measurements and Analysis*. 1st ed., Kuwer Academic Publisher, 2004, pp. 181-208.

[3] M.B Dwyer, G.S. Avrunin, J.C. Corbett. "A System of Specification Patterns," *in Proc. of the 2nd Workshop on Formal Methods in Software Practice*, 1998.

[4] O. Mondragon, A.Q. Gates. "Supporting Elicitation and Specification of Software Properties through Patterns and Composite Propositions," *Intl. Journal Software Engineering and Knowledge Engineering*, vol. 14(1), Feb. 2004.

[5] D. Peters. "Automated Testing of Real-Time Systems." Technical report, Memorial University of Newfoundland, 1999.

[6] N. Delgado, A.Q. Gates, S. Roach. "A Taxonomy and Catalog of Runtime Software-Fault Monitoring Tools," in *IEEE Trans. Softw. Eng.* 30, 2004, pp. 859-872.

[7] G. Holtzmann. "The Spin Model Checker," *in IEEE Transactions on SE*, vol. 23(5), 1997, pp. 279-295.

[8] I. Gallegos, A.Q. Gates, C.E. Tweedie. "Toward Improving Environmental Sensor Data Quality: A Preliminary Property Categorization," *in Proceedings of the International Conference on Information Quality (ICIQ)*, 2010.

[9] S. Konrad, B.H.C. Cheng. "Facilitating the Construction of Specification Pattern-Based Properties," in *Proc. IEEE Requirements Engineering,* 2005, pp. 329-338.

[10] I. Gallegos, A.Q. Gates, C.E. Tweedie. "DaProS: A Data Property Specification Tool to Capture Scientific Sensor Data Properties," *in Proceedings of the Workshop on Domain Engineering DE@ER10*, 2010.

[11] J.A. Hourcle. "Data Relationships: Towards a Conceptual Model of Scientific Data Catalogs," in *Eos Trans. AGU,* vol. 89(53), 2009.

[12] Campbells Scientific. "Instruction manual: CSAT Three Dimensional Sonic Anemometer." Logan, Utah, Campbells Scientific Inc.: 70. 2008.

[13] Campbells Scientific. "Open Path Eddy Covariance Training." Logan, CSI. 2009.

[14] K.M. Havstad, L.F. Huenneke, W. H. Schlesinger., "Structure and Function of a Chihuahua Desert Ecosystem: The Jornada Basin Long-Term Ecological Research Site," *Oxford University Press*, 1st Edition, 2006, pp. 44-80.

[15] Weather Underground. "WeatherUnderground," Internet: http://www.wunderground.com/. [Feb, 2012].

[16] R.A. Weekley, R.K. Goodrich, L.B. Cornman. "An Algorithm for Classification of Outlier Detection of Time-Series Data," in *Journal of Atmospheric & Oceanic Technology*, vol. 27, pp. 94-107, 2010.

[17] E.W. Dereszynski, T.G. Dietterich. "A Probabilistic Model for Anomaly Detection in Remote Sensor Streams." M.A thesis, Oregon State University, USA, 2007.

[18] T. Dean, K. Kanazawa. "Probabilistic Temporal Reasoning," in *Proc. AAAI*, 1988, pp. 524-529.

[19] The University of Edinburgh School of GeoSciences. "EdiRe," Internet: http://www.geos.ed.ac.uk/abs/research/micromet/EdiRe/ [Feb, 2012]

Reconfiguration of Robot Applications using Data Dependency and Impact Analysis

Michael E. Shin, Taeghyun Kang
Department of Computer Science
Texas Tech University
Lubbock, TX 79409-3104
{michael.shin; th.kang}@ttu.edu

Sunghoon Kim, Seungwook Jung,
Myungchan Roh,
Intelligent Robot Control Research/ETRI
Daejeon, Korea
{saint; sejung; mcroh}@etri.re.kr

Abstract

This paper describes an approach to reconfiguring component-based robot applications against component failures using the data dependencies between components and their impact analysis. Most of the components constituting robot applications are activated periodically to process periodic data delivered by other components. A component depends on another component in terms of periodical data. In this paper, the impact of data dependency between components is analyzed as to how a component failure affects its dependent component. The impact levels are categorized as insignificant, tolerable, serious, and catastrophic. The data dependencies between components and their analyzed impacts are used to reconfigure robot applications against component failures. The proposed reconfiguration approach can be a basis for recovery and safety of robot applications. The approach is applied to the Unmanned Ground Vehicle (UGV) application.

1. Introduction

Most of the components constituting robot applications depend on other components in terms of periodic data. A robot application can be made by robot components, which are designed for specific functionalities. Most of the robot components perform their functions periodically to process periodic data delivered by other components. If a periodic component does not deliver the data to its dependent component, it causes the dependent components to fail to process the periodic data. Periodic components in robot applications should be activated periodically to process the periodic data within the specified cycle time. A component failure can spread out to local components nearby the failed component or to all the components in the robot application.

The impact of a periodic component failure to its dependent components differs depending on how important the component is to its dependent components. Some component causes a minor impact to its dependent components, whereas other component can be critical to its dependent components. Suppose that the Car Detection component in the Unmanned Ground Vehicle (UGV) application receives the periodic pictures from a Camera and the laser data array from a Laser Range Finder (LRF) to detect other cars trajectory. The Car Detection component may ignore the LRF failure because it can detect other cars trajectory with only Camera pictures. However, the Car Detection component may be seriously affected from a Camera component failure if it cannot receive the Camera picture periodically.

Functional or data dependencies [Mohamed10a, Mohamed10b, Vieira02, Ahn10, Popescu10] have been investigated to analyze the impact of failures. [Mohamed10a] analyzes the propagation of dependencies in terms of value, silent, and performance. In [Ahn10], fault impact levels are classified by ignore, reset, and stop, and different fault handling is performed according to the level of fault impact. [Mohamed10b] claims that a component failure increases the failure severity of an application if the component has a high propagation probability.

This paper describes an approach to reconfiguring component-based robot applications against periodic component failures using the data dependencies between components and their impact analysis. The impact of a component failure to its dependent components is analyzed and classified as different impact levels. The reconfiguration of robot application is carried out based on the data dependencies and impact levels. But recovery and detection of failed components are out of scope this paper.

2. Dependency and Impact Analysis

A feature of robot applications composed of components is that a component processes data periodically delivered by another component. Robot applications process periodic stimuli captured by sensors in the environment and respond to them via actuators. A component interfacing with a sensor receives and processes the sensor input periodically, and then it delivers to other components processing the data. The processed data may be sent to another component so that the data is processed further. The output corresponding to the sensor input is sent to the components interfacing with actuators that carry out the output.

Two components in robot applications have a data dependency relationship if a component processes the data received from another. The data dependency relationship between components is defined as the opposite direction of the dataflow between components. In this paper, the dataflow and data dependency relationship between components are represented with an arrow and a dotted arrow respectively. Fig. 1 depicts the dataflow between components for part of the Environment Modeling and Perception (EMP) in the Unmanned Ground Vehicle (UGV) application [Roh11]. The Laser Range Finder (LRF) and Camera components generate LRF data array and 24 bits raw RGB respectively. The Car Detection component produces other cars trajectory by analysing the data that comes from the LRF and Camera components. Using the data from the Car Detection component, the Dangerous Situation component determines wheather it should generate an alarm to prevent critical accidents.

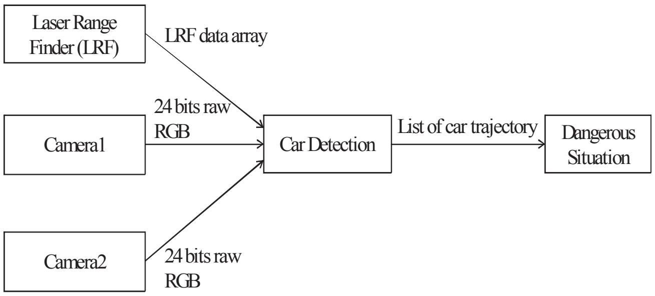


Fig. 1 Dataflow between Components for Environment Modeling and Perception in UGV application

The impact of a component failure to another component is analyzed using the data dependency relationship between components. The impact is defined depending on how much a component is affected from other component failure, and the impact level is determined by considering the importance of received data. The impact level of a component failure to its dependent components is categorized as insignificant, tolerable, serious, and catastrophic. An insignificant impact level describes a data dependency in which a component uses an extra data from another component to verify or increase the reliability of component output additionally. A component can provide its full functionality required without any problem even though there is no extra data from a failed component. The impact is represented with an impact level on the data dependency relationship between components.

Fig. 2 depicts the data dependency relationships between components for the Environment Modeling and Perception (EMP), which are defined based on the dataflow in Fig. 1. The LRF component is insignificant for the Car Detection component (Fig. 2), which receives data from the LRF and two Camera components to trace other cars trajectory. Even though the LRF component fails, the Car Detection component can still produce other cars trajectory using only the data generated from the Camera components. The LRF component is added to the EMP as an extra component so that the Car Detection component can produce high quality of other cars trajectory.

Failures of a component, which is tolerable to its dependency component, do not disrupt the normal function of its dependent component. The dependent component may have a minor impact from a component failure. The Car Detection component (Fig. 2) is supported by two Camera components, each impact level of which is defined as tolerable. Even though one of two Camera components fails to capture other cars pictures, the Car Detection component can make other cars trajectory using the remaining Camera component.

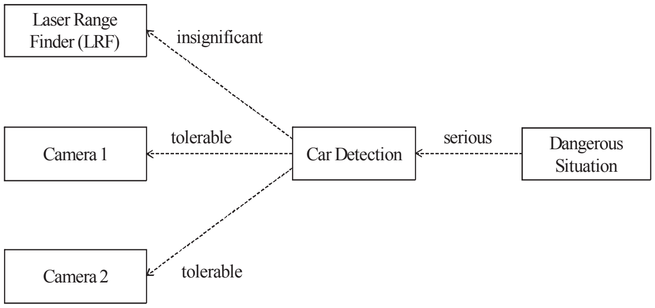


Fig. 2 Data Dependency Relationship between Components for Environment Modeling and Perception in UGV application

A component failure that makes serious impact to its dependents component causes the dependent to stop processing data. This is because the dependent component cannot generate a reliable output any more without the data from the failed component. Stopping the dependent components could give a ripple effect to the next dependent components so that it can paralyze part of an application. But, stopping the dependent components does not lead the application to a total failure. The application can provide still partial services using the remaining components, which may not have critical impact from the stopped components. The data generated by the Car Detection component is critical to the Dangerous Situation component in Fig. 2, so the impact of Car Detection component to the Dangerous Situation component is defined as serious. When the Car Detection component fails, the Dangerous Situation component should stop processing the data. This is because the Dangerous Situation component cannot generate the reliable output without the data from the Car Detection component.

Catastrophic impact is used to describe a situation where a component failure needs to stop all components constituting an application. An application may encounter a critical accident if all the components do not stop immediately. Catastrophic impact may be associated with safety of robots. The Virtual Robot component in the UGV application controls real devices such as engine and steering wheel. A failure of Virtual Robot component may cause the UGV to encounter an accident, which could lead to lose human life.

Some kind of data dependency relationships between components, which have the same impact level, can have a cardinality constraint that describes a minimum number of data dependencies required for keeping the impact level. A component may receive data from the same type of multiple components. In this case, the component depends on the multiple components that have the same impact level. If the dependent component cannot receive data from at least some number of same type components, it can have additional impact from the missing data. In this case, the impact level of the same type components to its dependent component needs to increase one level up. Each Camera component (Fig. 2) has a tolerable impact to the Car Detection component, but the tolerable impact should change to serious if both Camera components encounter failures at the same time. This is because the Car Detection component requires the data from at least one Camera component so as to generate the reliable output.

3. Reconfiguration

3.1 Prototype

An application configuration manager as a prototype is developed to validate the reconfiguration of a robot application using the data dependency relationships between components and their impact analysis. The application configuration manager decides which components need to be reconfigured using the data dependency relationships between components and their impact levels in response to a component failure. The scope of reconfiguration is decided depending on the criticality of component dependencies, such as insignificant, tolerable, serious, or catastrophic. For this, the application configuration manager generates a reconfiguration plan by considering ripple effects of the failure using the data dependency relationships and their impact levels. This plan includes all the components being affected from the failure.

The application configuration manager reconfigures an application by interacting with executors in the OPROS engine [Song08, Jang10]. The OPROS engine is a platform for robot applications that executes robot applications composed of components. Periodic components that process data periodically are activated and run by executors in the OPROS engine. Each executor runs components that have the same periodic cycle times. When an executor detects a component failure, it requests reconfiguring the components associated with the failed component from the application configuration manager. Each executor has the worst case execution times of components, detecting a component failure if the component cannot finish its periodic cycle time within its worst case execution time.

Figure 3 depicts the outline of our prototype for reconfiguring components against component failures using the data dependency relationships between components and their impact analysis. Suppose two executors in the OPROS engine execute periodic and same cycle-timed components. When the first component in the Executor2 fails, the executor notifies the application configuration manager of the failure. The application configuration manager checks how much the failed component has impact on its dependent components in the application. A component in the Executor1 depends on the failed component and the impact level is serious. Now the component in the Executor1 waits for some data from the failed component. Without the failed component, the component in the Executor1 cannot process the data. The application configuration manager requests the Executor1 to stop the dependent component. Similarly, the second component in the Executor1 depends on the second and third components in the Executor2. However, the dependent component can survive even though the second component in the Executor2 fails. This is because the impact level is tolerable.

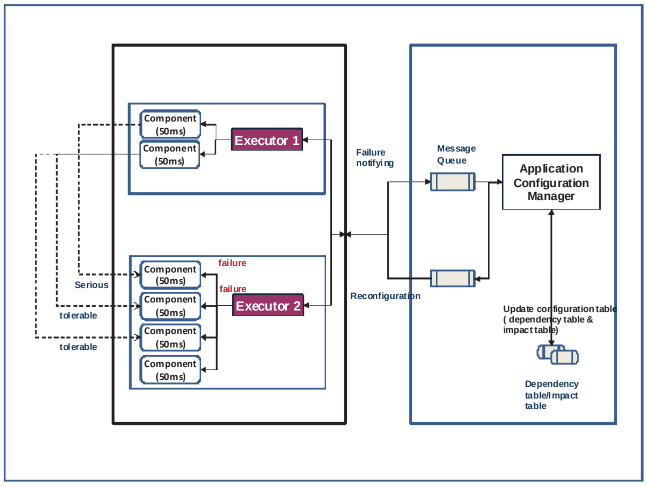


Fig. 3 Outline of Reconfiguration in OPROS

3.2 UGV Application

The approach proposed in this paper is validated with the Unmanned Ground Vehicle (UGV) application [Roh11], which drives a car to the destination place safely without a human driver's intervention. Fig. 4 depicts the data dependency relationships between components and their impact levels for the Pedestrian Detection that is used for validating our approach, along with the Environment Modeling and Perception (Figs. 1 and 2) described in section 2. The Thermal Imaging Camera (TIC) renders infrared radiation as a visual light. The Car Detection component produces other cars trajectory by analysing the data that comes from LRF, Camera, and TIF components. The Pedestrian component detects people on the road and the Localization component generates the world space absolute coordinates. Using the data from the Localization, Car Detection and Pedestrian components, the Dangerous Situation component determines if it should generate an alarm to prevent critical accidents.

For test purpose, we injected failures to components and the reconfiguration result were compared to the expected reconfiguration. For instance, failures were injected to the Camera1 first and then to the Camera2 to check if the Camera1 and Camera2 failures affects the Car Detection and Pedestrian components. Also, we tested the reconfigurations for the LRF component that is insignificant to the Car Detection componen, and Pedestrian component that are serious to the Dangerous Situation component. These test results were the same as the expected reconfiguration.

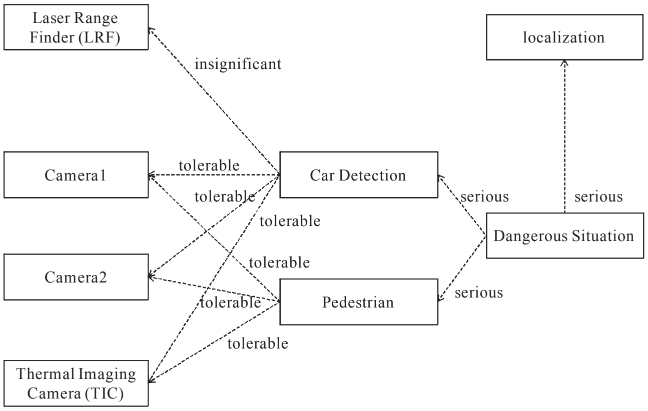


Fig. 4 Pedestrian Detection, and Environment Modeling and Perception in UGV Application

4. Conclusions

This paper has described an approach to reconfiguring component-based robot applications using the data dependency relationships between components and their impact analysis. The impact of a component failure to its dependent components are analyzed and classified as different impact levels, such as insignificant, tolerable, serious, and catastrophic. Based on different impact levels, a robot application is reconfigured against a component failure. To validate, a prototype for reconfiguration has been developed with the UGV application and it has been tested with OPROS engine.

This paper can have future research. The constraints on impact levels between data dependency relationships between components need be specified formally. The impact level between data dependency relationships can be changed dynamically and should be specified by means of constraints. Also the proposed approach needs to be applied to large-scale robot applications to validate further. Moreover, this paper could be extended to include event and functional dependency relationships between components for reconfiguring a robot application.

Acknowledgement

This work was supported by the Industrial Foundation Technology Development Program of MKE/KEIT, Rep. of Korea [10030826, Development of Reliable OPRoS Framework].

References

[Ahn10] Ahn, H., D. Lee, and S. Ahn, “Hiearchical Fault Tolerant Architecture For Component-based Service Robots,” 8th IEEE International Conferences on Industrial Informatics(INDIN), Osaka, Japan, 2010, pp.487-492 .

[Jang10] Jang, C., S. Lee, S. Jung, B. Song, R. Kim, S. Kim, and C. Lee, "OPROS: A New Component-Based Robot Software Platform," ETRI Journal, vol.32, no.5, Oct. 2010, pp.646-656.

[Mohamed10a] Mohamed, A., and M. Zulkernine, “Failure type-Aware reliability Assessment Component Failure Dependency,” 4th International Conference on Secure software Integration and reliability Improvement(SSIRI) Singapore, 2010, pp.98-105.

[Mohamed10b] Mohamed, A., and M. Zulkernine, “The Level of Decomposition Impact on Component Fault Tolerance,” IEEE 34 th Annual Computer Software and Applications Conference Workshops(COMPSACW), Seoul, Korea, 2010, pp.57-62.

[Popescu10] Popescu, D., “Impact Analysis for Event-Based Components and Systems,” ACM/IEEE 32nd International conference on Software Engineering, Cape Town, South Africa, 2010, pp.401-404.

[Roh11] Roh, M. C., J. Byun , and S. Kim, “Design of the UGV System based on Open Platform Robot of Software,” 6th Korean conference on robots, Seoul, Korea, June, 2011.

[Song08] Song, B., S. Jung, C. Jang, and S. Kim, “An introduction to Robot Component Model for OPROS (Opend Platform for Robotic Services),” International Conference on Simulation, Modeling, and Programming for Autonomous Robots, Venice, Italy, 2008.

[Vieira02] Vieira, M., and D. Richardson, “Analyzing dependencies in Large Component-based systems,” 17th IEEE International Conference on Automated Software Engineering (ASE) , Edinburgh, UK, 2002, pp.241-244.

Spacemaker: Practical Formal Synthesis of Tradeoff Spaces for Object-Relational Mapping

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

Sang H. Son
University of Virginia,
151 Engineer's Way,
Charlottesville, VA 22903 USA
son@virginia.edu

***Abstract*—Developing mixed object-relational (OR) mappings that achieve desirable quality attribute tradeoffs for object-oriented applications is difficult, tedious, costly, and error-prone. We contribute a practical, automated technique for exhaustive, formal synthesis of large spaces of such mappings, and the clustering of individual mappings in these spaces into multi-dimensional quality equivalence classes. This technique can help engineers to design effective persistence layers for object-oriented applications. Our approach is to use a formal language to describe both a space of mappings and multiple quality attribute valuation functions on points in such a space. We use a constraint solver to exhaustively enumerate points in this space and their valuations. We then cluster the results into quality attribute equivalence classes. This work promises to reduce the cost and time required to develop mixed OR mappings, ensure their formal correctness, and help engineers to understand and make tradeoffs quantitatively. We conducted application-oriented experiments to test feasibility and scalability of our approach. In one experiment we synthesized an OR mapping tradeoff space for a real e-commerce application, synthesizing and classifying hundreds of thousands of mappings in just a few minutes.**

***Index Terms*—Design; Database; Object-relational mapping; Design space exploration; Alloy Language.**

I. Introduction

Object-oriented applications often need to use relational databases for persistent storage. Transformations between instance models in these two paradigms encounter the so-called *impedance mismatch* problem [9]. Object-relational mapping (ORM) systems are now widely used to bridge the gap between object-oriented application models and relational database management systems (DBMS), based on application-specific definitions on how object models are to be mapped to database structures.

The problem we address is that today one has to choose between automatic generation of mappings using *pure* mapping strategies [3], [9], or the manual design of mixed mappings, in which different mapping strategies are applied to individual classes rather than to entire inheritance hierarchies. Producing pure mappings automatically is easy, but it often leads to sub-optimal results. Developing mixed mappings by hand can achieve much higher quality, but it is hard and error-prone. Among other things, it requires a thorough understanding of both object and relational paradigms, of large spaces of possible mappings, and of the tradeoffs involved in making choices in these spaces.

To address this problem, we present an approach that provides both the quality benefits of mixed mappings and the productivity benefits of automated synthesis. We present a practical formal automated technique, implemented in a prototype tool that we call *Spacemaker*, for exhaustive synthesis of mixed OR mappings and their classification into quality attribute equivalence classes. We take as inputs a formal object model and optional class-specific mapping strategies for those classes that the user wants mapped in a specific manner. We then use an automated constraint solver to exhaustively generate the space of mappings subject to the given constraints, along with multiple quality attribute measures for each mapping. Next we cluster mappings into quality attribute equivalence classes and present candidates from each class to the engineer along with the measures of its quality attributes. The engineer can then select a mapping to satisfy tradeoff preferences. Our prototype tool uses Alloy as a specification language [10], and the Alloy analyzer as a constraint solver.

In more detail, we claim four main contributions: (1) We present what is to our knowledge the first formalization of fine-grained and mixed ORM strategies by means of mapping functions; (2) we contribute a fully automated approach for formally precise synthesis of mapping tradeoff spaces, based on this formalization; (3) we present an experimental demonstration of technical feasibility and scalability to practically meaningful applications; and (4) we develop the Spacemaker tool [1], which we make available to the research and education community. Data from our experiments support the claim that our technique can reduce the time to develop high-quality OR mappings, ensure their correctness, and enable engineers to make tradeoffs based on automatically computed quality attributes for a full range of possible mappings rather than on intuition or hard-to-acquire expertise in OR mapping strategies.

The rest of this paper is organized as follows. Section II presents our approach. Section III reports and discusses data from the experimental testing of our approach. Section IV surveys the related work. Section V concludes the paper with an outline of our future work.

II. Approach

We present our approach in three parts. We first formalize application of object-relational mapping strategies. We then use these formalizations to automate synthesis of quality equivalence classes of OR mappings. Third, we describe algorithms that are important for the scalability of our approach. In a nutshell, they serve to decompose large object models into smaller components for which mapping problems can be solved independently. This element of our work helps avoid combinatorial explosion in constraint solving.

A. Formalization

The issue of mapping an object model to a set of relations is described thoroughly in the research literature [3], [9], [11], [14]. To provide a basis for precise modeling of the space of mapping alternatives, we have formalized OR mapping strategies in an appropriate level of granularity. As an enabling technology, we chose Alloy [10] as a specification language and satisfaction engine for three reasons. First, its logical and relational operators makes Alloy an appropriate language for specifying object-relational mapping strategies. Second, its ability to compute solutions that satisfy complex constraints is useful as an automation mechanism. Third, Alloy has a rigorously defined semantics closely related to those of relational databases, thereby providing a sound formal basis for our approach.

The principal mapping strategies are explained in terms of the notations suggested by Philippi [14], and Cabibbo and Carosi [3]. To manage association relationships, we have formally specified three ORM strategies of *own association table*, *foreign key embedding* and *merging into single table*. We have also defined three more ORM strategies for inheritance relationships: *class relation inheritance (CR)*, *concrete class relation inheritance (CCR)* and *single relation inheritance (SR)*. Furthermore, as the aforementioned ORM strategies for inheritance relationships are just applicable to the whole inheritance hierarchies, we have defined three extra predicates for more fine-grained strategies: *Union Superclass*, *Joined Subclass* and *Union Subclass*, suitable to be applied to the part of an inheritance hierarchy to let the developer design a detailed mapping specification using the combination of various ORM strategies. To make our idea concrete, we illustrate the semantics of one of these strategies in the following.

```
 1 pred UnionSubclass[c: Class]{
 2    c in (isAbstract.No) =>{
 3      one Table<:c.~tAssociate
 4    }
 5    (c.isAbstract=No) =>{
 6      all a:Attribute|a in c.attrSet =>{
 7        one f:Field|f.fAssociate=a
 8        && f in (c.~tAssociate.fields) }
 9    }
10    (c.isAbstract=No)&&(c.^parent != none) =>{
11      all a:Attribute | a in c.^parent.attrSet =>{
12        one f:Field|f.fAssociate=a &&
13        f in (c.~tAssociate.fields) }
14    }
15    (c.~tAssociate).foreignKey = none
16    ...
17 }
```

Listing 1. Part of the Alloy predicate for the *UnionSubclass* strategy

Listing 1 partially outlines the Alloy predicate for the *Union Subclass* strategy, where each concrete class within the hierarchy is represented by a separate table. The strategy predicate then states, in lines 5–14, that each table encompasses relational fields corresponding to both attributes of the associated class and its inherited attributes. As such, to retrieve an individual object, only one table needs to be accessed. Finally, this strategy implies no referential constraint over the mapped relations.

B. Design Space Exploration

In the previous section, we showed how executable specifications can be used to formalize OR mapping strategies. In this section, we tackle the other aspect that needs to be clarified: how we can apply a design space exploration approach to generate *quality equivalence classes* of OR mappings based on those specifications.

A design space is a set of possible design alternatives, and design space exploration (DSE) is the process of traversing the design space to determine particular design alternatives that not only satisfy various design constraints, but are also optimized in the presence of a set of objectives [15]. The process can be broken down into three key steps: (1) Modeling the space of mapping alternatives; (2) Evaluating each alternative by means of a set of metrics; (3) Traversing the space of alternatives to cluster it into equivalence classes.

Modeling the Space of Mapping Alternatives

For each application object model, due to a large number of mapping options available for each class, its attributes and associations and its position in the inheritance hierarchy, there are several valid variants. To model the space of all mapping alternatives, we develop a generic mixed mapping specification based on fine-grained strategies formalized in previous section. This generic mixed mapping specification lets the automatic model finder choose for each element of the object model any of the relevant strategies, e.g. any of the fine-grained generalization mapping strategies for a given class within an inheritance hierarchy.

Applying such a loosely constrained mixed mapping strategy into the object model leads to a set of ORM specifications constituting the design space. While they all represent the same object model and are consistent with the rules implied by a given mixed mapping strategy, they exhibit totally different quality attributes. For example, how inheritance hierarchies are being mapped to relational models affects the required space for data storage and the required time for query execution.

We called this mapping strategy loosely constrained because it does not concretely specify the details of the mapping, such as applying, for example the *UnionSubclass* strategy to a specific class. An expert user, though, is able to define a more tightly constrained mixed mapping by means of the parameterized predicates *Spacemaker* provides, as we demonstrate in the next section. The more detailed the mapping specifications, the narrower the outcome design space, and the less the required postprocessing search.

Measuring Impacts of OR mappings

Mapping strategies have various kinds of impacts in terms of quality attributes of applications. There are several approaches proposed in the literature dealing with the challenge

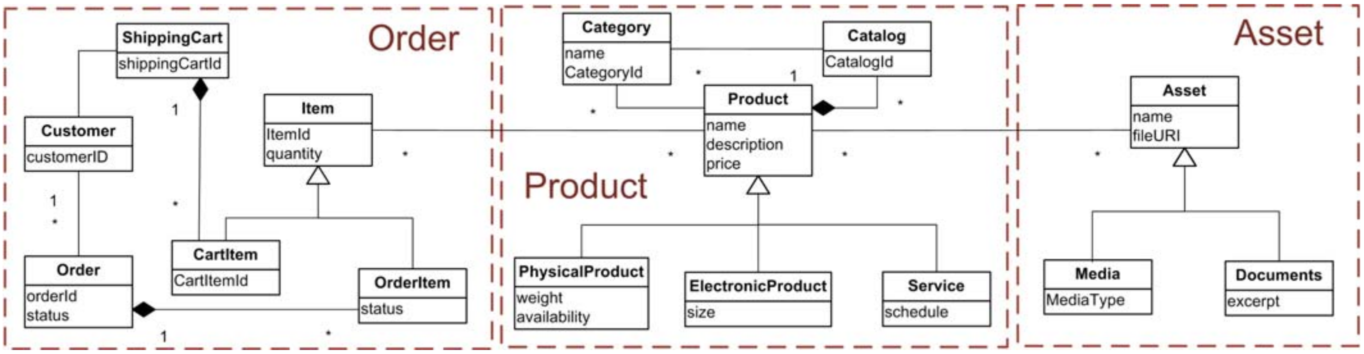


Fig. 1. The ecommerce object model.

of defining metrics for OR mapping impacts on non-functional characteristics. It has been shown that efficiency, maintainability, and usability, among the set of all quality attributes defined by the ISO/IEC 9126-1 standard, are characteristics significantly influenced by OR mappings [8]. For each of those attributes, we use a set of metrics suggested by Holder et al. [8] and Baroni et al. [2]. The metrics are *Table Access for Type Identification* (*TATI*), *Number of Corresponding Table* (*NCT*), *Number of Corresponding Relational Fields* (*NCRF*), *Additional Null Value* (*ANV*), *Number of Involved Classes* (*NIC*) and *Referential Integrity Metric* (*RIM*).

To measure these metrics, we developed a set of queries to execute over synthesized alternatives. For brevity, and because it suffices to make our point, we concisely describe one of these metrics and the corresponding query in the following. *Spacemaker* supports the others as well.

The *Number of Corresponding Relational Fields* (NCRF) metric specifies the extent of change propagation for a given OR mapping. Specifically, the NCRF metric manifests the effort required to adapt a relational schema after applying a change, such as inserting or deleting an attribute, over a class. According to the definition, given a class *C*, *NCRF(C)* specifies the number of relational fields in all tables that correspond to each non-inherited, non-key attribute of *C*. The specification of a query we designated to measure the NCRF metric over synthesized alternatives is given below:

NCRF(C) = #(C.attrSet - C.id). fAssociate. fields

The Alloy dot operator denotes a relational join. While *attrSet* specifies a set of non-inherited attributes of a class, *fAssociate* is a relation from a table field to its associated class attribute. The query expressions then, by using the Alloy set cardinality operator *(#)*, defines the NCRF metric.

Exploring, Evaluating and Choosing

The next step is to explore and prune the space of mapping alternatives according to quality measures. Spacemaker partitions the space of satisfactory mixed mapping specifications into equivalence classes and selects at most a single candidate from each equivalence class for presenting to the end-user.

To partition the space, Spacemaker evaluates each alternative with respect to previously described relevant metrics. So each equivalence class consists of all alternatives that exhibit the same characteristics. Specifically, two alternatives a_1 and a_2 are equivalent if *value*(a_1, m_i) = *value*(a_2, m_i) for all metrics (m_i). Because equivalent alternatives all satisfy the mapping constraints, it suffices to select one alternative in each equivalence class to find a choice alternative. Given that quality characteristics are usually conflicting, there is generally no single optimum solution but there are several pareto-optimal choices representing best trade-offs.

C. Model Splitting

As with many formal techniques, the complexity of constraint satisfaction restricts the size of models that can actually be analyzed [7]. Our approach also requires an explicit representation of the set of all quality equivalence classes of mapping alternatives, which in general grows exponentially in the number of elements in a model.

To address these scalability problems, we split the object model into sub-models. The key idea is that since for association relationships with cardinality of many-to-many, there is just one applicable mapping strategy, i.e. *own association table*, we make use of such relations to split the object model into sub-models.

We consider an object model as a graph, $G_{objModel} =< V, E >$, where nodes V represent classes, and there is an edge $< v_i, v_j >$ joining two nodes v_i and v_j if there is a direct relationship including association and generalization link between them. We assume that $G_{objModel}$ is connected. Otherwise, we consider each sub-graph separately. An edge joining two nodes v_i and v_j in a graph is a *bridge* if removing the edge would cause v_i and v_j to lie in two separate subgraphs [5]. A bridge is the only route between its endpoints. In other words, bridges provide nodes with access to parts of the graph that are inaccessible by other means. So, to decompose a graph $G_{objModel}$, we remove all bridges of type many-to-many association.

To make our idea concrete we consider the ecommerce domain model we adopted from Lau and Czarnecki [13]. According to the diagram shown in Figure 1, there are two such bridges: $< Product, Asset >$ and $< Item, Product >$. By removing those bridges we obtain three smaller sub-graphs. The gain then comes from the reduction in the sizes of the constraint solving problems. That is, we replace a large

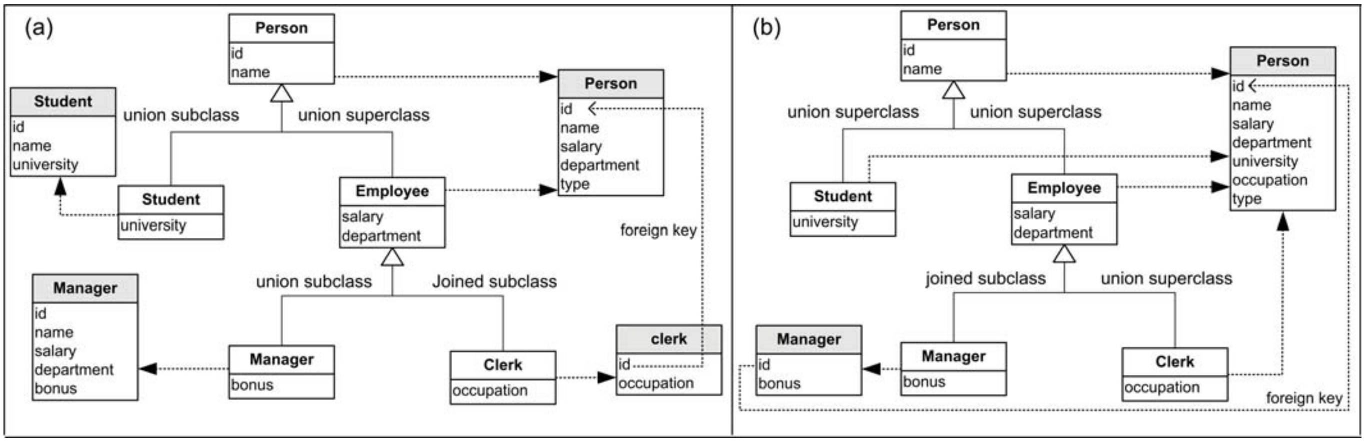


Fig. 2. Examples of mixed mapping strategies

constraint solving problem with smaller and more manageable problems that can in particular be addressed by our formally precise synthesis technique.

III. Evaluation

The claim we make in this paper is twofold: (1) It is feasible to formalize the *correctness constraints* for object-relational mapping strategies, thereby to automate the synthesis of an exhaustive set of mixed object-relational mapping candidates, and that it is possible to statistically analyze each of the candidates in dimensions of six major mapping quality metrics, and thereby to cluster them into quality equivalence classes; (2) for non-trivial systems, the performance of the technology implementation based on a bounded model checker is reasonable (on the order of minutes).

To test the feasibility hypothesis, we develop a prototype tool that implements it, called Spacemaker [1], which is available for download. We show that our ideas are practical by applying Spacemaker to several case studies from the object-relational mapping literature. We then compare the discrepancies between our formally derived OR mappings and the manual mappings published in the literature. The differences revealed problems with their mappings, suggesting again that manual development of OR mappings can be error-prone.

Figure 2 shows two applications of mixed ORM strategies, adopted from Holder et al. [8]. White boxes represent classes, while boxes having grey titles represent corresponding mapped tables. Black and white arrows represent mapping and inheritance relationships, respectively. Finally, foreign keys as well as the applied mapping strategies are also mentioned in the diagrams.

Listing 2 formally describes part of the *personObjectModel* according to the diagram. At the top, it imports the declaration of *objectModel*, and then defines *Person* and its attribute, *name*, using signature extension as a subtype of *Class* and *Attribute* types. The other characteristics of the class are also specified.

To specify a mixed OR mapping, the developer can call fine-grained ORM strategies, given as inputs those classes to be mapped in a specific manner. *Spacemaker* then automatically generates the corresponding mapping specifications, should they exist. The followings outline the high-level definition of mapping specifications for Figure 2a.

```
module personObjectModel
open objectModel

one  sig Person extends Class {}{
  attrSet = identifier+name
  id=identifier
  no parent
  isAbstract = No  }

one sig name extends Attribute {}{
  type in string  }
```

Listing 2. Example of object model (elided) in Alloy

```
open ORMStrategies
open personObjectModel

UnionSubclass[Manager]
JoinedSubclass[Clerk]
UnionSuperclass[Employee]
UnionSubclass[Student]
```

Figure 3 illustrates the computed result for the example of Figure 2a. The diagram is accurate for the result automatically computed, but we have edited it to omit some details for readability (fields of tables and primary key relationships, for example). In this diagram, Table 1 is associated to *Person* and *Employee* classes, which are being mapped by the *union superclass* strategy. Separate tables are associated to both *Student* and *Manager* classes, according to the *union subclass* strategy. Finally, application of the *joined subclass* strategy leads to a separate table for *Clerk* with a foreign key, omitted in the diagram, to its superclass corresponding table.

To enumerate the space of mappings for the given object model, we use the *genericMixedStrategy*, with the set of classes within the hierarchy as inputs. This generic strategy lets the automatic model finder, here Alloy, to choose for each class any of the fine-grained strategies and to see whether their combinations applied to classes within the hierarchy is satisfiable or not. Alloy guarantees that all computed mapping candidates conform to the rules implied by mapping predicates formalizing correctness constraints.

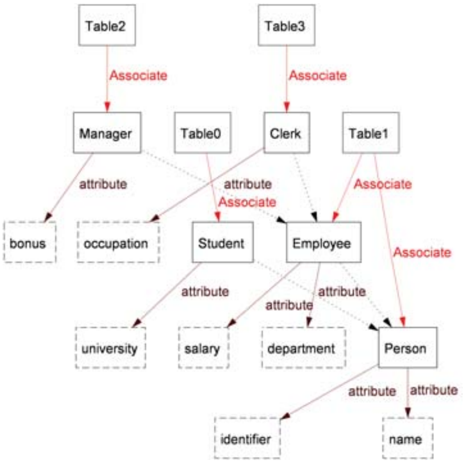


Fig. 3. Mapping diagram for Figure 2a derived automatically based on mixed mapping strategies of *union superclass*, *joined subclass* and *union subclass*

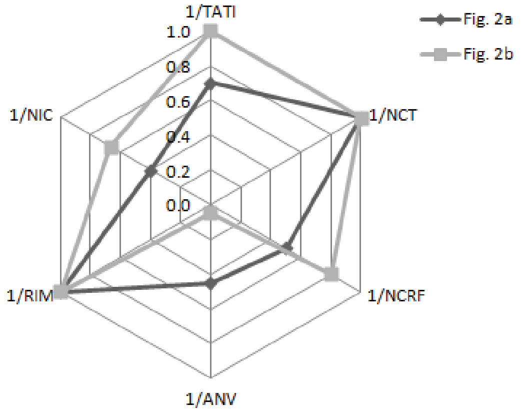


Fig. 4. Multi-dimensional *quality measures* for two mapping candidates represented in Figure 2

We used a PC with an Intel Core i5 2.67 Ghz processor and 4 GB of main memory, and leveraged *SAT4J* as the SAT solver during the experiments. Given all the specifications and mapping constraints, Spacemaker using the Alloy Analyzer then generate 760,000 mapping candidates, assess them, and reduce them to 40 equivalence classes, in less than 10 seconds.

The spider diagram, shown in Figure 4, illustrates the 6-dimensional "quality measures" for two mapping candidates represented in Figure 2. To display quality measures in one diagram, we normalized the values.

According to the diagram, if the designer opts for the resource utilization, the mapping depicted in Figure 2a would be a better option. More specifically, with respect to the ANV metric, representing additional storage space in terms of null values, the mapping of Figure 2b requires more *wasted space*. This is because instances of four different classes, namely Person, Student, Employee and Clerk, are stored together in a shared table. Thus, each row in the shared table that represents an instance of the Student class, for example, contains a null value at each relational field corresponding to the other classes. On the other hand, if the designer opts for maintainability and performance, the mapping depicted in Figure 2b would be a better choice. More precisely, the mapping of Figure 2a negatively affects the NCRF metric reflecting the effort required to adapt the relational schema. This is partly because applying the *UnionSubclass* strategy results in duplication of relational fields. With respect to the TATI metric which is a performance indicator of polymorphic database queries, this mapping also poses performance problems.

Focusing on the second hypothesis, to test that Spacemaker is able to handle also non-trivial OR mappings, we select an object model of a real ecommerce system [13]. This object model, shown in Figure 1, represents a common architecture for the kind of open source and commercial ecommerce systems. It includes 15 classes connected by 9 associations and consists of 7 inheritance relationships.

Without decomposition, the Alloy Analyzer ran out of memory before synthesizing the whole space of mapping alternatives. Given the splitting algorithm, we decompose the object model to three sub-models and feed them into the Spacemaker. Figure 5 represents the results.

ObjectModel	Solutions	Eq.Classes	$T[min]$
Product	137,000	67	46
Asset	124,000	31	$1 >$
Order	93,000	31	16

Fig. 5. Ecommerce Experiment performance statistics.

Interpretation of data shows that similar to the former experiment, the synthesis time for the *Asset* sub-model is in the order of seconds, but for the other two sub-models is in the order of minutes. This is mainly because in the former case, the analyzer just considers inheritance mapping strategies as there are no associations in those models, while in the other models the constraints of both inheritance and association mapping strategies are involved. So it takes more time for the model finder to generate satisfying solutions for them.

As Spacemaker solves sub-models separately, the constraint solving bottleneck depends on the largest sub-model to solve. Although the number of valid solutions is high, i.e. hundreds of thousands of satisfiable solutions, Spacemaker is able to generate quality equivalence classes of mappings in an acceptable amount of time, which confirms that the proposed synthesis technology is feasible.

Discussion

This work shows that ORM strategies can be formalized and implemented as executable specifications, and that Spacemaker can automatically synthesize and prune the space of mapping alternatives in an effective manner. Our formal recapitulations of previous studies also reveals some problems. For example, the referential integrity constraint in Figure 2b, is not mentioned in the source paper [8], but exists in the mapping specifications automatically derived using *Spacemaker*. Our discovery of such inconsistencies provides an example of how our formal synthesis technique can help designers in an error-prone task of developing OR mappings.

Overall, this work appears to support the idea that shifting the responsibility of finding an optimized mapping specification from technicians —who better understand mapping strategies, their implications, and techniques for mapping object models to relational models— to the domain experts, more aware of requirements and specifications is a plausible aspiration.

IV. Related Work

We can identify in the literature two categories of work that are closely related to our research. The first one concerns the research that deals with deriving database-centric implementations from Alloy specification. The second one encompasses all works that have been done in the object-relational mapping research area.

Focusing on the first category, Krishnamurthi et al. [12] proposed an approach to refine Alloy specifications into PLT Scheme implementations with special focus on persistent databases. Cunha and Pacheco [4] are similarly focused on translating a subset of Alloy into the corresponding relational database operations. These works share with ours an emphasis on using formal methods. However, our work differs in its focus on separating application description from other independent design decisions, such as choices of OR mapping strategies. Furthermore, we use Alloy not only to specify the object model, but also to model the spaces of OR mappings consistent with both the given object model and choice mapping strategy, and to automate the mapping process.

Regarding the second area, a large body of work has focused on object-relational mapping strategies and their impacts to address the impedance mismatch problem [3], [9], [11], [14]. Among others, Philippi [14] categorized the mapping strategies in a set of pre-defined quality trade-off levels, which are used to develop a model driven approach for the generation of OR mappings. Cabibbo and Carosi [3] also discussed more complex mapping strategies for inheritance hierarchies, in which the various strategies can be applied independently to different parts of a multi-level hierarchy. We share the same problem domain with these approaches, but our focus is on automating formal derivation of equivalence classes of mapping specifications.

Drago et al. [6] also considered OR mapping strategy as one of the variation points in their work on feedback provisioning. They extended the QVT-relations language with annotations for describing design variation points, and provided a feedback-driven backtracking capability to enable engineers to explore the design space. While this work is concerned with the performance implications of choices of per-inheritance-hierarchy OR mapping strategies, it does not really attack the problem that we are addressing, namely the automated and exhaustive synthesis of the equivalence classes of mixed OR mapping specifications.

V. CONCLUSION

While a wealth of research has been performed on bridging application models and databases to address the *impedance mismatch* problem, little has been done on automated support for the derivation of mapping specifications for ORM frameworks. In this paper, we presented a novel approach that substantially supports automatic generation of such mapping specifications to deliver the quality of expert-hand-crafted mappings and the productivity benefits of fully automated techniques. This approach ultimately promises to reduce the engineering personnel costs involved in producing high-quality modern software systems. The new mapping approach exposes many interesting research challenges. These challenges include exploring symmetry breaking techniques customized for the specific domain of OR mappings to reduce the size of the solution space and integrating the mapping compiler with industrial object-relational mapping tools.

Acknowlegements

This work was supported in part by the National Science Foundation under grant #1052874.

References

[1] Spacemaker tool suite. http://www.cs.virginia.edu/~hb2j/Downloads/Spacemaker.zip.

[2] A. L. Baroni, C. Calero, M. Piattini, and O. B. E. Abreu. A formal definition for ObjectRelational database metrics. In *Proceedings of the 7th International Conference on Enterprise Information System*, 2005.

[3] L. Cabibbo and A. Carosi. Managing inheritance hierarchies in Object/Relational mapping tools. In *Proceedings of the 17th International Conference on Advanced Information Systems Engineering (CAiSE'05)*, pages 135–150, 2005.

[4] A. Cunha and H. Pacheco. Mapping between alloy specifications and database implementations. In *Proceedings of the Seventh International Conference on Software Engineering and Formal Methods (SEFM'09)*, pages 285–294, 2009.

[5] David Easley and Jon Kleinberg. *Networks, Crowds, and Markets: Reasoning About a Highly Connected World*. Cambridge University Press, 2010.

[6] M. L. Drago, C. Ghezzi, and R. Mirandola. A quality driven extension to the QVT-relations transformation language. *Computer Science - Research and Development*, 2011.

[7] Ethan K. Jackson, Eunsuk Kang, Markus Dahlweid, Dirk Seifert, and Thomas Santen. Components, platforms and possibilities: Towards generic automation for MDA. In *Proceedings of International Conference on Embedded Software*, 2010.

[8] S. Holder, J. Buchan, and S. G. MacDonell. Towards a metrics suite for Object-Relational mappings. *Model-Based Software and Data Integration*, CCIC 8:43–54, 2008.

[9] C. Ireland, D. Bowers, M. Newton, and K. Waugh. Understanding object-relational mapping: A framework based approach. *International Journal on Advances in software*, 2:202–216, 2009.

[10] D. Jackson. Alloy: a lightweight object modelling notation. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 11(2):256–290, 2002.

[11] W. Keller. Mapping objects to tables - a pattern language. In *Proc. of the European Pattern Languages of Programming Conference*, 1997.

[12] S. Krishnamurthi, K. Fisler, D. J. Dougherty, and D. Yoo. Alchemy: transmuting base alloy specifications into implementations. In *Proceedings of FSE'08*, pages 158–169, 2008.

[13] S. Q. Lau. *Domain Analysis of E-Commerce Systems Using Feature-Based Model Templates*. Master's thesis, University of Waterloo, Canada, 2006.

[14] S. Philippi. Model driven generation and testing of object-relational mappings. *Journal of Systems and Software*, 77(2):193–207, 2005.

[15] T. Saxena and G. Karsai. MDE-based approach for generalizing design space exploration. In *Proceedings of the 13th international conference on Model driven engineering languages and systems*, MODELS'10, pages 46–60, 2010.

A formal support for incremental behavior specification in agile development

Anne-Lise Courbis[1], Thomas Lambolais[1], Hong-Viet Luong[2], Thanh-Liem Phan[1], Christelle Urtado[1], and Sylvain Vauttier[1]

[1]LGI2P, école des mines d'Alès, Nîmes, France, First.Last@mines-ales.fr
[2]Laboratoire Ampère, UMR 5005, INSA-Lyon, Lyon, France, Hong-Viet.Luong@insa-lyon.fr

Abstract

Incremental development is now state of the practice. Indeed, it is promoted from the rational unified process to agile development methods. Few methods however guide software developers and architects in doing so. For instance, no tool is proposed to verify the non-regression of functionalities, modeled as behavior specifications, between increments. This work helps to incrementally specify software functionalities using UML state machines. It provides an on-the-fly evaluation of a specified behavior as compared to that of previous increments. The proposed contribution is based on two formally specified relations that are proved to preserve refinement when composed. Architects and developers are free to choose their preferred behavior specification strategy by iteratively applying them, so as to develop the required functionalities, having at each step the benefit of a formal non-regression checking to guide the global specification process. Our proposal is implemented in a proof-of-concept tool and illustrated by a didactic case-study.

***Keywords**: UML state machines, incremental development, agile methods, state machine verification, conformance relations, refinement.*

1. Introduction

The evolution of software system development processes currently follows two apparently contradictory main trends. *Agile and extreme programming* promote fast development of small increments that will altogether constitute the expected complete software [3]. These development methods are very popular as they are concrete, foster the sense of achievement among development teams and best satisfy clients as well as stakeholders by early, fast and regular deliveries of usable and valuable software that incrementally integrates all the required functionalities. However, the lack of a big picture to guide the development process towards well defined goals may lead to harmful inconsistencies such as regressions or substitution mismatches. *Model driven engineering* (MDE) promotes models as the main artifacts to capture both requirements and the designed solution. They are used, via automated or assisted transformations, to create the implementation of the system. MDE concentrates developers' efforts on the early development steps, trying to specify once and then generate implementations to various target technologies or execution frameworks, skipping, as much as possible, tedious repetitive design or coding tasks. MDE provides an effective support to capture the big picture of a specification and reason about design decisions. However, MDE does not yet fully support disciplined incremental development. Indeed, non regression is often prevented by the means of tests [7]. MDE lacks formal tools to perform behavioral model verifications.

This paper advocates that it is possible to combine the advantages of both trends by providing tools to compare the behavior specifications of increments and evaluate the existence of refinement relations in order to verify the global consistency of the development process. This enables incremental but disciplined development processes, supported by tools that provide guidance to enforce consistency. UML state machines are used as a uniform means to model behaviors throughout the development process, from initial, partial and abstract specifications, that interpret requirements as functionalities, to detailed designs, that fully define system dynamics. This way, incremental behavior specification and design schemes can be proposed thanks to three relations, that we have adapted from process algebra literature to state machines in previous work [17]:

- the behavior *extension* relation (noted ext) captures the fact that a machine adds behaviors to another one, without impeding existing mandatory behaviors.

- the behavior *restricted reduction* relation (noted redr) captures the fact that a machine does not add extra observable behaviors and that mandatory behaviors are preserved: non observable behaviors may be detailed and optional behaviors may be removed.
- the behavior *refinement* relation (noted refines) links an abstract machine to a more concrete one and enforces that all the mandatory behaviors specified in the abstract machine are preserved in the refined one. Some optional behaviors may be removed, while new observable ones may be added, provided they do not conflict with existing ones.

These relations are going to serve as a basis for the incremental development of behavior models. The idea of the paper is that they altogether form a formal yet not constraining means to evaluate the consistency of artifacts produced when using agile development processes.

Whichever relations are composed, the latter machines are guaranteed to be conform implementations of the formers. When none of theses relations can be asserted between two successive machines, a rupture is detected in the refinement process. This paper advocates for a guided revision mechanism that helps analyze the cause of the inconsistency and decide which machine should be modified: either the proposed implementation may be erroneous, or the abstract machine may be over-specified and impossible to be properly implemented. Once this ambiguity is resolved, the system might also help designers propagate involved corrections to other machines so as to establish the required relations. These two cases show how composing the restricted reductions and extensions might constitute an agile but disciplined method for specifying the behavior of systems.

The remainder of the paper is structured as follows. Section 2 describes a didactic motivating example that is used as an illustration throughout the paper. Section 3 presents our proposal. It describes the three proposed relations we choose to support state machine development and then presents how they can be used to support an agile development scenario. Section 4 discusses our approach against state of the art before concluding in Section 5 with some perspectives.

2. Motivating example

Informal specification of a Vending Machine. The specification a Vending Machine (Figure 1) contains mandatory parts (refund the customer unless goods are obtained), as well as optional parts (maintenance, cookies).

Successive UML state machines. In order to progressively design the behavior of this vending machine, the designer produces several intermediate state machines (Figure 2). He starts from mandatory behaviors, considering coin and cancel signals only. Hence, the Minimal Machine is a rather stupid machine, which specifies that after any amount of coins, the user can be refunded and that is the only thing he can ask for. The drink signal is always ignored. In the second NeverEmpty Machine, the designer adds the ability to react to the drink signal, in *some cases* after the coin signal. At that time, the user can still be refunded. If he chooses a drink, the machine will eventually distribute it and give him his money back. Note that this machine is *nondeterministic*[1].

The system delivers goods after the customer inserts the proper credit into the machine. Goods are drinks, but could also be cookies. Optionally, a technician can shutdown the machine with a special code. When used by a customer (not a technician), the system runs continuously. An important feature is that the system must not steal the user: if the customer has not inserted enough money, changes his mind or if the system is empty, the system refunds the user.

Figure 1: Informal specification

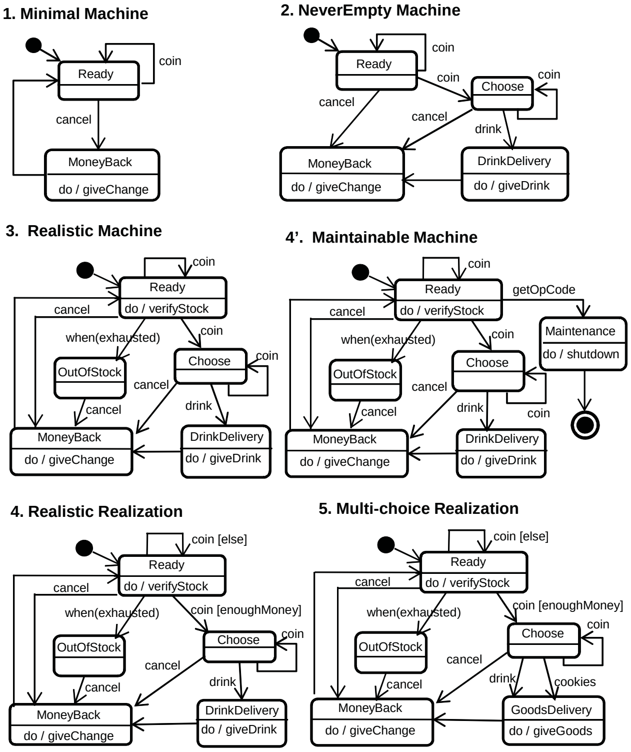


Figure 2: Incremental development proposing several UML state machines

The third Realistic Machine considers the fact that the

[1] Although nondeterminism is not allowed in UML, we consider that final models have to be deterministic, but that initial and intermediate models may be nondeterministic.

machine may be empty which should leave solely the cancel action to the user. This machine describes every *mandatory* part of the above informal specification. Other features (cookies and maintenance) are options. Hence, it can lead to the concrete state machine (Realistic Realization) which fixes the nondeterministic points and can be used as a basis for a first implementation. This concrete machine can then be extended (Multi-choice Realization) to add the second choice of goods with the cookies signal. Alternatively, from the third Realistic Machine, we could also extend the behaviors and consider the getOpCode signal in the Ready state. These development sequences show that concrete machines can be derived from intermediate abstract models, which only describe the most important features. In that sense, they obey an agile development process where simplified products are quickly produced. Machines 4 and 5 are called *realizations* since they are concrete, deterministic machines, which describe all the mandatory behaviors of the informal requirements of Figure 1.

Verification needs. Having such development scenarios in mind, we focus on the following properties:

r_1 : *Implementation.* At any step of a development process, the resulting machine has to fulfill the requirements expressed by the first specification model.

r_2 : *Liveness preservation*. Liveness properties state a system has to react to some signals after precise signal sequences. At any time, our example system must react to the cancel button and refund the user; after a given sequence of coin signals and if the machine is not empty, it must react to the drink button.

r_3 : *Progress*. During the development process, when an M' machine is supposed to be *an extension* of an M machine, we want to be able to verify that any behavior offered by M is also offered by M' in the same conditions.

r_4 : *Safety preservation*. Safety properties state that some actions are forbidden. When an I machine is *a restricted reduction* of an M machine, we need to guarantee that I does not implement behaviors not described by M. On the example, delivering a product that has not been paid for or delivering two products instead of one are such forbidden actions.

r_5 : *Composability.* When chaining extensions, we want the result to be an extension of the initial machine, and similarly for reductions. When combining extensions and reductions, we need to know the relation between the resulting machine and the initial one.

3. Relations to support incremental development processes

The verification technique we choose to satisfy these properties is to compare models between them. This excludes the developer to separately describe liveness and safety properties in an another language, as in [10]. At first, we mainly focus on properties r_1, r_2, r_3 and r_4.

3.1. Behavior conformance relation

Conformance testing methodologies proposed by ISO [13] compare an implementation to a standard specification. Recommendations define mandatory and optional parts. An implementation is in conformance to a specification if it has properly implemented all the *mandatory parts* of that specification [19]. We consider conformance as our reference behavior *implementation* relation.

Formalizing the conformance relation [5] consists in comparing the event sets that *must be accepted* by the compared models, after any trace of the specification model. A *trace* is a partial *observable* sequence of events and/or actions that the machine may perform. The set of traces of an M machine is noted $\mathrm{Tr}(M)$. An implementation model conforms to a specification model if, after any trace of the specification, any set of events that the specification *must* accept, *must* also be accepted by the implementation model. We refer to [14] for a study of this relation on Labeled Transition Systems (LTSs), and to our works [17] for an implementation technique and a translation from UML state machines to LTSs.

In the example of Figure 2, $\mathrm{Tr}(\textsf{Minimal Machine}) = \{\textsf{coin}^*, (\textsf{coin}^*.\textsf{cancel})^*\}$. Minimal Machine must accept coin and cancel events after any trace $\sigma \in \mathrm{Tr}(\textsf{Minimal Machine})$. This property is satisfied by the NeverEmpty Machine, which consequently conforms to Minimal Machine.

NeverEmpty Machine conf Minimal Machine

The conformance relation is suited for implementation (r_1 property) and liveness (r_2 property). However, it is too weak to guarantee progress and safety properties (r_3, r_4), and, since it is not transitive, it cannot answer property r_5. It thus cannot be considered as a refinement relation.

3.2. Behavior refinement relation

Considering conf as an implementation relation, the refinement relation (refines) is defined as the largest relation satisfying the following *refinement* property: For all machines M_1 and M_2,

$$M_2 \textsf{ refines } M_1 \Longrightarrow \forall I, I \textsf{ conf } M_2 \Rightarrow I \textsf{ conf } M_1. \quad (1)$$

The refines relation has the following properties:

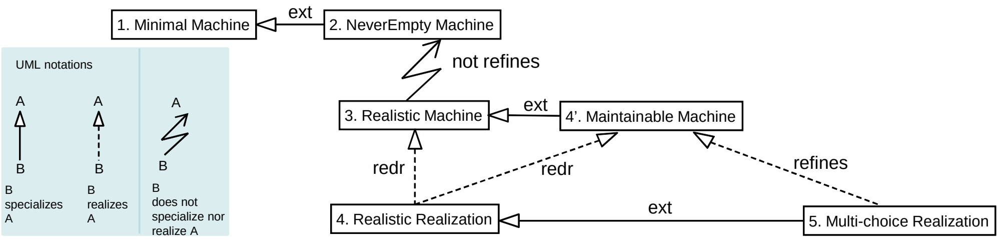


Figure 3: Synthesis of relations

- refines $\subseteq$ conf: refines can be used as an implementation relation (property r_1) and inherits properties of the conf relation, such as liveness preservation (property r_2);
- it is transitive;
- If M_2 refines M_1, for any trace σ of M_2 which is not a trace of M_1, M_2 must refuse everything after σ.

This definition of refinement is large enough to encompass both notions of classical refinement [2] and incremental constructions (extension).

3.3. Specialized behavior refinement relations

Extension. The *extension* relation is defined by ext $=_{def}$ refines $\cap \supseteq_{Tr}$, where, for two machines M_1 and M_2, $M_2 \supseteq_{Tr} M_1 =_{def} \mathrm{Tr}(M_2) \supseteq \mathrm{Tr}(M_1)$. The ext relation inherits implementation and liveness preservation property from refines (properties r_1, r_2). It is moreover defined to satisfy the progress property (r_3). The ext relation is a refinement relation that reduces partiality and nondeterminism. In Figure 2, the NeverEmpty Machine offers the possibility to ask for a drink, without preventing the user from doing something he could do with the Minimal Machine:

$$\text{NeverEmpty Machine ext Minimal Machine} \qquad (2)$$

Restricted reduction. To overcome the fact that ext does not preserve safety (one cannot know whether the drink action, which is new, is safe or not), restricted reduction is defined by redr $=_{def}$ refines $\cap \subseteq_{Tr}$. redr inherits properties from refines and adds safety preservation property (r_4). redr is very similar to classical refinement (it reduces abstraction and nondeterminism). In Figure 2, the RealisticRealization is a reduction of the Maintainable Machine: the getOpCode signal can be refused by Maintainable Machine after any trace in $\{\varepsilon, \text{coin}^*\}$;

$$\text{RealisticRealization redr Maintainable Machine} \qquad (3)$$

To summarize, we keep three refinement relations which are transitive and preserve liveness properties: refines is the largest one, ext is the subset of refines ensuring progress and redr is the subset of refines ensuring safety preservation. Our goal now is to study the composition (property r_5) between these three relations.

Analysis on the example. It appears that:

$$\text{not(Realistic Machine refines NeverEmpty Machine)} \qquad (4)$$

The Realistic Machine may refuse coin after the empty trace ε, whereas NeverEmpty Machine must always accept coin. Considering the development process proposed in Figure 2, result (4) gives information to the designer. He has to answer the question whether the coin event must always be accepted initially or may be refused, by forcing him to ask for the cancel event. This point is not clear in the informal requirements (Figure 1). If the developer considers the coin action is mandatory, he must correct Realistic Machine by adding for instance a self-transition triggered by coin on state OutOfStock. Otherwise, if coin can be refused when the machine is empty, Realistic Machine is to be considered as the new reference specification.

Figure 3 sums up relations ext, redr, and refines on the six proposed machines. In an agile development process, having proposed Realistic Realization model, the designer can come back to Maintainable Machine, checking that the following result (5) is satisfied. Then, knowing result (3), he can propose Multi-choice Realization and verify relation (6):

$$\text{Maintainable Machine ext Realistic Machine} \qquad (5)$$

$$\text{Multi-choice Realization ext Realistic Realization} \qquad (6)$$

3.4. Composing relations to support agile development processes

We call *strategy* the successive steps that a designer chooses to achieve a development. With our disciplined framework, this amounts to choose to add behaviors (horizontal refinement) or details (vertical refinement) to the current behavior model, producing a new behavior model that must verify an ext or a redr relation. Figure 4 shows how an Agile development process can be managed as an instantiation of such a strategy. From partial and abstract requirements (S_0), a specification of the first increment to implement is defined (S_1). This specification is detailed through several design steps to produce eventually an implementation model (S_3). Then, a new increment is defined (S_4), extending the previous, and so on, until all requirements are implemented. Thereafter, an evolution of the software, based on revised requirements (S_7), can be developed as a new development process. A crucial issue is to guarantee that the development process leads to conformant implementations (for instance S_3 as compared to S_1 or S_0) whereas only local consistency is stated by the refinement relations that are built between successive models. This implies to verify that refinement relations can be composed into implementation relations.

Local composition. Locally, two ext and redr relations commutatively compose refines relations:

$$\mathsf{redr} \circ \mathsf{ext} = \mathsf{ext} \circ \mathsf{redr} = \mathsf{refines} \tag{7}$$

This result comes from the definition of redr and the properties of conf (see 3.2 and 3.3). As redr and ext relations are easier to check, it is interesting to deduce refines relations from them.

Global composition. Globally, ext and redr relations compose with refines relations as follows:

$$\mathsf{refines} \circ \mathsf{ext} = \mathsf{ext} \circ \mathsf{refines} = \mathsf{refines} \tag{8}$$

$$\mathsf{refines} \circ \mathsf{redr} = \mathsf{redr} \circ \mathsf{refines} = \mathsf{refines} \tag{9}$$

These two properties derive from the fact that for any preorder A and any relation $X \subseteq A$, we have: $A \circ X = X \circ A = A$.

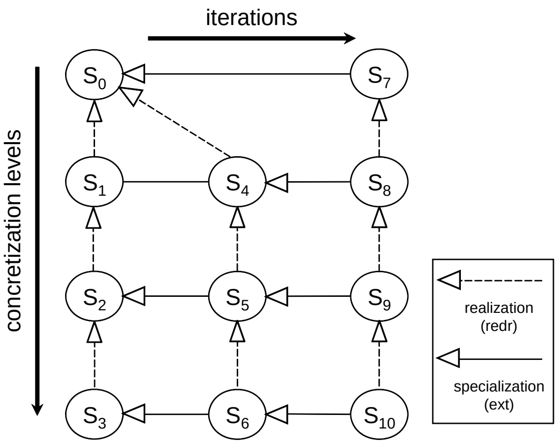


Figure 4: Instantiation of an Agile development process

Applied to the example, result (10) is inferred from computed relations (3), (6) and by property (7):

$$\mathsf{Multi\text{-}choice\ Realization\ refines\ Maintainable\ Machine} \tag{10}$$

With property (9), we also conclude that:

$$\mathsf{Multi\text{-}choice\ Realization\ refines\ Realistic\ Machine} \tag{11}$$

Such calculations provide an effective support to agile development. At any step, designers can freely choose their preferred strategy and leverage inferred relations to carry on development.

4. Discussion and related works

Process algebra implementation and refinement relations. In order to satisfy property r_1, we find a large number of behavior model comparison relations in the context of process algebras and LTSs. Milner's observational equivalence and congruence, based on bisimulations [18], are well known relations to compare a detailed implementation model to an abstract specification model. Observational congruence can be considered as an implementation relation in a strong sense, where mandatory as well as optional behaviors must be present in the implementation model. They have been implemented in several toolboxes such as [10]. Milner's observational congruence preserves safety and liveness. However, it does not satisfy the r_3 property: observational congruence cannot be used in an incremental process.

An interesting result is that conformance is weaker than Milner's observational congruence: any observationaly congruent models are also conformant. Hence, the refines relation still distinguishes dangerous from harmless livelocks, as in Milner's theory, which is not the case of Hoare's CSP refinement relations [12].

Incremental construction versus refinement. *Refinement* has to be discussed since it has various interpretations. It is a well-known and old concept [22] used in some reference works about state machine refinement [1] or specification refinement [2], where it is considered as a relation for comparing two models in order to reduce non determinism and abstraction. This relation corresponds to a reduction: it consists in introducing details into models in order to get an implementation model. It has been implemented in languages such as B [2] and Z [8]. From our point of view, founding a development process on such a relation is restrictive. We prefer the definition given by [4], in accordance with definition 1 of section 3.2. Note that this relation is called *consistency* in [14] and some researchers of the UML community prefer this term rather than *refinement*. This definition is interesting because it includes the conventional refinement based on reduction but does not exclude the extension of initial specifications. The benefits of our approach compared to the conventional refinement processes are manifold. It is close to the way of reasoning and to the practice of designers to finalize complex models. It is close to agile processes to allow rapid delivery of high-quality software meeting first requirements of customers. It attests the feasibility of a first implementation model before enhancing it to get the final one. Finally, it might help to support evolution because "systems, and consequently their design, are in perpetual evolution before they die" [15].

Related approaches for analyzing state machine consistency. Few works deal with incremental development of state machines. [6] addresses the problem at architecture level (state machine assembly). [9] does alike and guides assembling with rules.

Works about state machines verification have to be mentioned despite their different objectives, as they focus on consistency between a software and its specification. Many works are based on model checking techniques. UML is thus transformed into the modeling language of the model checking tool: it can be PROMELA to use SPIN as it is done in [16] or LTSs to use JACK as in [11]. Some works analyze consistency using pre and post-conditions as it is done in [4] using the Z formalism. Lastly, consistency can be expressed through transformations as it is done for refactoring in [21]. Such techniques require to explicitly express liveness properties.

5. Conclusions and future works

In this paper, we address the issue of the incremental construction of state machines to support agile development processes. It implies a composition of successive vertical and horizontal refinements that must globally achieve a consistent implementation of the initial software specification. The study of existing works points out that these two aspects are never considered as a whole, despite they are key points to define development strategies.

We demonstrated the computational feasibility of our proposal by developing a JAVA tool named IDCM (Incremental Development of Conforming Models). It implements the verification of conf, ext, redr and refines relations [17] by transforming UML state machines into LTSs and analyzing their relations. Analysis provides designers with feedback about detected warnings or errors.

Beyond the several experimented case studies, we plan to evaluate our proposal and tool on full size projects. We also currently study the adaptation of this work to component-based architectures, in other words to coarse-grained, reuse-centered development approaches, to address complexity and scalability issues.

References

[1] M. Abadi and L. Lamport. The existence of refinement mappings. In *Logic in Computer Science*, pages 165–175, 1988.

[2] J. Abrial. *The B-Book : Assigning Programs to Meanings*. Cambridge University Press, 1996.

[3] S. W. Ambler. *The Object Primer: Agile Model-Driven Development with UML 2.0*. 3rd edition, 2004.

[4] E. Boiten and M. Bujorianu. Exploring UML refinement through unification. In *Critical Systems Development with UML*, LNCS, page 47—62, 2003.

[5] E. Brinksma and G. Scollo. Formal notions of implementation and conformance in LOTOS. Technical Report INF-86-13, Twente University of Technology, Department of Informatics, Enschede, Netherlands, Dec. 1986.

[6] S. Burmester, H. Giese, M. Hirsch, and D. Schilling. Incremental design and formal verification with UML/RT in the FUJABA Real-Time tool suite. In *SVERTS*, 2004.

[7] A. Cicchetti, D. D. Ruscio, D. S. Kolovos, and A. Pierantonio. *A test-driven approach for metamodel development*, chapter Emerging Technologies for the Evolution and Maintenance of Software Models, pages 319–342. IGI Global, 2012.

[8] J. Derrick and E. Boiten. *Refinement in Z and object-Z: foundations and advanced applications*. Springer-Verlag, 2001.

[9] G. Engels, J. H. Hausmann, R. Heckel, and S. Sauer. Testing the consistency of dynamic UML diagrams. In *Proc. 6th Int. Conf. on Integrated Design and Process Technology*, 2002.

[10] H. Garavel, F. Lang, R. Mateescu, and W. Serwe. CADP 2010: A toolbox for the construction and analysis of distributed processes. In P. Abdulla and K. Leino, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 6605 of *LNCS*, pages 372–387. Springer, 2011.

[11] S. Gnesi, D. Latella, and M. Massink. Modular semantics for a UML statechart diagrams kernel and its extension to multicharts and branching time model-checking. *Journal of Logic and Algebraic Programming*, 51(1):43–75, Apr. 2001.

[12] C. A. R. Hoare. *Communicating sequential processes*. Prentice Hall, June 2004.

[13] ISO/IEC 9646-1. Information technology – Open Systems Interconnection – Conformance testing methodology and framework – Part 1: General concepts, 1991.

[14] G. Leduc. A framework based on implementation relations for implementing LOTOS specifications. *Computer Networks and ISDN Systems*, 25(1):23—41, 1992.

[15] M. Lehman. On understanding laws, evolution, and conservation in the large-program life cycle. *Journal of Systems and Software*, 1:213–221, 1980.

[16] J. Lilius and I. Paltor. Formalising UML state machines for model checking. In *UML conf.*, 1999.

[17] H. Luong, T. Lambolais, and A. Courbis. Implementation of the Conformance Relation for Incremental Development of Behavioural Models. *Models 2008, LNCS*, 5301:356–370, 2008.

[18] R. Milner. *Communication and concurrency*. Prentice-Hall, 1989.

[19] S. Moseley, S. Randall, and A. Wiles. In Pursuit of Interoperability. In K. Jakobs, editor, *Advanced Topics in Information Technology Standards and Standardization Research*, chapter 17, pages 321–323. Hershey, 2006.

[20] D. E. Perry and A. L. Wolf. Foundations for the study of software architecture. *ACM SIGSOFT Software Engineering Notes*, 17(4):40–52, 1992.

[21] G. Sunyé, D. Pollet, Y. L. Traon, and J. Jézéquel. Refactoring UML models. In *UML conf.*, pages 134–148, 2001.

[22] N. Wirth. Program development by stepwise refinement. *Communications of the ACM*, 14(4):221–227, 1971.

A Process-Based Approach to Improving Knowledge Sharing in Software Engineering

Sarah B. Lee Ph.D., Kenneth Steward
Department of Computer Science and Engineering
Mississippi State University
Mississippi State, MS, USA
sblee@cse.msstate.edu, kcs111@msstate.edu

Abstract—**Information technology organizations often assume that to improve knowledge sharing, a knowledge management system must be independently developed or purchased. Also often assumed is the idea that implementing tools for knowledge sharing will provide a foundation for the evolution of processes and culture that support knowledge sharing. The intent of this research is to illustrate that an effective approach to increased knowledge sharing and collaboration must consider first the processes that make up a knowledge workers daily context, and the knowledge that supports that workflow. Organization of people and process must be considered in order for knowledge creation to be efficient and effective.**

Keywords- knowledge management; process improvement

I. Introduction

Knowledge has become increasingly relevant for organizations since the shift from an industrial economy to a global, decentralized, and knowledge-based economy. Organizations now work, compete, and cooperate on a worldwide scale [1]. Today, employees are often geographically separated within a single city or on a corporate campus. Loss of knowledge in the handoff of work products is an area of concern. Offices in different locations may use different terminology and tools, making knowledge sharing across departmental boundaries a challenge [2]. A variety of technologies have been employed to attempt to close this communication gap including company intranets, directories, and groupware [3].

When applied to software engineering, knowledge management deals with a wide domain including project management, communication with clients and end users, problem solving, code reuse, staff development, and maintenance and support. Project success rates continue to be an area of concern, and one reason given for repeated failures is that organizations do not adequately use existing experience to avoid prior mistakes. This occurs when knowledge created during a project is not captured or shared for later use [4]. To avoid this continuous cycle, organizational knowledge produced during the software development lifecycle (SDLC) must be recognized as a valuable asset and leveraged to meet objectives.

II. Background

While software development shares many characteristics with other engineering disciplines, it presents new challenges due to its reliance on the knowledge and creativity of individual software developers and their interactions. In addition to the engineering aspects of software development, cognitive aspects and human activity must be considered. The development of a software system requires integration of knowledge from a variety of sources. Software development can be thought of as “the crystallization of knowledge into the software systems” [5].

With a global software development model, the gathering of requirements may involve a co-located team of analysts and users. Business requirements must be translated to software requirements specifications, leading ultimately to development work which may be performed by a team located separately from the team that worked on requirements analysis and development. Throughout the software development process, knowledge producer and consumer roles emerge.

A Software Requirements Specification (SRS) is produced that includes all requirements needed by the developers to address the business requirements. The SRS must be reviewed to validate that it meets the business need as defined, with traceability to the business requirements documented. The development and infrastructure support teams should collaborate to confirm if the SRS is feasible and verifiable. The SRS is typically used to validate technical designs, code, and test cases.

The importance of the software architecture design which documents the components and their connectors that comprise a software system should not be ignored. Typically all the knowledge regarding the design decisions on which the architecture is based are implicitly embedded in the architecture. This lack of explicit representation leads to increased complexity, a high cost of making changes, and design erosion. In the design phase, the main concern is on which design decision should be made. In the coding phase, it is often important to know why certain decisions have been made [6].

A. *Codification of Knowledge*

Explicit knowledge is represented in a way that is easily transferred from one person to another. Data files, reports, and other physical representations of knowledge are explicit. Tacit knowledge is the knowledge that people maintain inside themselves and is harder to formalize. It is inferred through the behaviors of individuals. Access to tacit knowledge is typically obtained through personal contact and trust.

Nonaka describes how organizational and individual knowledge are created through continuous dialogue among individuals involving their tacit and explicit knowledge [7]. Organizational knowledge is created in a continual cycle through: socialization, internalization, externalization, and combination. These four modes of knowledge creation are described in Table 1.

TABLE I. MODES OF KNOWLEDGE CREATION

	To Tacit	**To Explicit**
From Tacit	Socialization	Externalization
From Explicit	Internalization	Combination

With socialization, the key to gaining tacit knowledge is some form of shared experience through interactions with individuals. The tacit-to-tacit socialization mode is initiated when a group of people forms and interaction begins. Externalization encompasses the conversion of tacit to explicit knowledge and is triggered through the documentation of shared tacit knowledge and new knowledge. This explicit knowledge is combined with existing data and available knowledge in the combination mode, forming new concepts and concrete knowledge. Internalization involves explicit to tacit knowledge; learning occurs when participants translate explicit knowledge gained into forms of individual tacit knowledge [7].

The combination mode of knowledge creation can be demonstrated through the engineering of new knowledge from existing knowledge sources. Lee et al. described the use of knowledge engineering to visualize complexity in the software engineering domain. This experience showed the value of knowledge engineered from legacy application metadata [8]. The goal of determining the risk as sociated with each application was realized and successfully applied to modernization strategy planning, supporting the goal of reducing complexity in domains with high business risk. Knowledge engineering of application meta-data into estimates of resource impact and potential risks and costs is applicable in a wide array of problem domains [8].

B. *Personalization of Knowledge*

Informal discussions represent a very important form of knowledge sharing where cross-fertilization of knowledge occurs. Finding a way to capture the knowledge exchange in these informal communications is challenging. Informal communication in communities can be promoted through the use of social networks and expert lists. The willingness of each person to participate is critical to success [5]. By integrating a community-based environment with a traditional knowledge repository approach, both tacit and explicit knowledge sharing is encouraged and supported.

Communities of practice, informal networks of individuals with shared interests, objectives, and social networks highlight the important link between social capital and knowledge resources in effective knowledge management. A community of practice refers to the process of social learning that occurs and the shared practices that emerge when people with common goals interact. A similar concept is a community of interest, a group of people who share a common interest. Communities reflect the “interests and/or expertise of people that are free to join one or more communities for: receiving help on specific fields; recommending or publishing any piece of information; informally discussing on themes; rating or inserting comments on elements; and so on” [9]. A community workspace may contain formal documents and informal comments. A software project may be tied to several communities. By understanding how these communities can successfully maintain and share knowledge, the potential exists to greatly enhance sharing of knowledge [10]. Marks et. al. argue that the stronger an individual identifies with a group, the more apt that person is to share information [11].

C. *Knowledge Sharing Philosophy*

Ezingeard et. al. discuss the importance of building a knowledge sharing culture as a foundation. The knowledge management sandcone in Fig. 1 demonstrates this concept [12] [13].

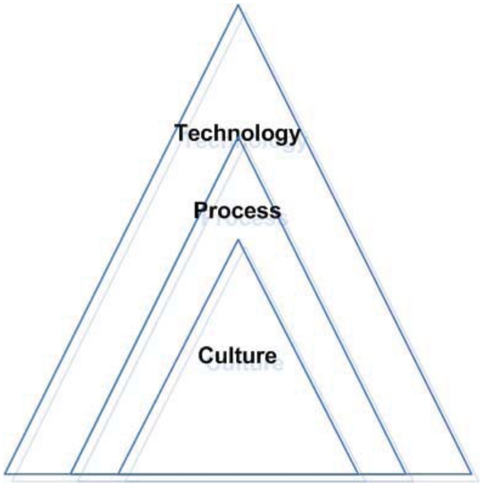


Figure 1. A depiction of the knowledge management sandcone concept [12].

Appropriate processes must support a knowledge-sharing culture, and then the technology layer can be implemented to facilitate the acquisition and distribution of knowledge [13].

Kock and Davison underlined the importance of process improvement in the knowledge management domain [14]. They concluded that using simple asynchronous collaborative technologies to support process improvement initiatives is likely to stimulate knowledge sharing among participants in cross-functional organizational processes. Through process improvement activities, knowledge sharing will increase with no extra resource commitment required from the organization other than that already committed to support process improvement activities [14]. Jennex and Olfman address this aspect with their KMS Success Factors, supporting work processes that incorporate knowledge capture and use [15].

III. The Experiment

The original context chosen for this research consisted of the activities required to define, analyze, and achieve the non-functional requirements of a software system. Through participatory action research, the primary author was embedded in the environment within which direct actions were taken to improve the performance within the target IT community, with observation of the effects of those changes [16].

Research was conducted within the IT department of a large Fortune 100 company over a period of two years. The context of the research project was initially defined as the flow of knowledge between infrastructure support teams and other stakeholders with particular focus on activities required to define, analyze, and achieve the non-functional requirements of a software system. No knowledge management strategy was employed in this environment. Documents were stored in a variety of sources, with no single point of access. Previously, there were internal efforts within small groups to improve collaboration through the use of Wiki and intranet portals. With employees given temporary and rotating assignments across projects, finding a point of contact for a particular knowledge area is often challenging [17].

Interviews were conducted across business, software development, and infrastructure support departments. The purpose of the interviews was to capture how stakeholders perceive knowledge sharing and process effectiveness within and across teams. Based on gaps identified through the interviews, the anticipated benefits to be realized by improved knowledge management were identified as follows:

- improved communication across SDLC stakeholders,
- consolidation of knowledge about infrastructure activities impacting development and business teams, and
- simplification of all stakeholder interactions regarding non-functional software requirements by improving first contact resolution and reducing the need to ask and respond to the same questions repeatedly.

Work was conducted in two phases. The second phase was developed based on reflections from phase one and a new cycle research question. Data was collected before and after action events in a variety of formats including observations and discussions with subjects. Statistics were gathered regarding access of individual intranet pages.

A. *Phase 1*

The cycle research question for the first phase was asked as follows. In what ways will an intranet user portal that provides a consolidated and coordinated view to knowledge artifacts and knowledge owners lead to increased knowledge sharing and improved collaboration across IT teams?

A knowledge hub was developed with a primary goal of promoting multi-way dialogue between stakeholders. An intranet-based portal enabled identification of user profiles and leveraged skill set inventories to facilitate *knowing who* and *knowing who knows what* [17].

Results were mapped to top KMS Success Factors, defined by Jennex and Olfman [15] as follows. The developed solution met the success factor calling for an integrated technical infrastructure through a knowledge sharing portal that integrated technologies currently available into the knowledge workers' daily work context. Sources of explicit knowledge were presented, along with author and storage formats for documents, partially meeting the success factor expectation of identifying users, sources, processes, and storage strategy. The intranet-based interface provided an easy to use view of frequently used documents as well as extended search capability which included artifacts in the document repository. This addressed the success factor of having an enterprise-wide and easy to understand structure with search, retrieval, and visualization functions. A most important result is that additional management support, another key success factor, was gained in support of continuing the study [15].

The concept of sharing knowledge about team members' interest and expertise was well-received by the participants interviewed. Leveraging communities of interest along with proactive notification of knowledge availability based on user profiles were documented requirements for future consideration [17].

A drawback to the approach in the first cycle was that development work, while minimal, required dedicated highly skilled resources which in turn could not be secured without management approval. Competition for development resources is fierce and allocation of these resources to further knowledge sharing cannot be reasonably prioritized over support for revenue generating business system support. An approach of converging technology to improve knowledge sharing, without the need for highly sought-after development resources, was considered for cycle two.

As the researcher became more embedded in the problem domain, observations revealed that a lack of just-in-time knowledge during processes related to the movement of application software to production caused process delays and degraded communication to upstream and downstream stakeholders. When participants in the process did not have the knowledge needed to execute their step in the process, the process stalled and often additional team members were brought into the situation. This increase in time and number of human resources increases the cost of executing the process. Tacit knowledge is created and shared during process execution, as well as creation of new explicit knowledge.

B. Phase 2

Based on the results of the first cycle, the second cycle research question was defined as: How will organizational and process changes increase knowledge sharing and collaboration both within infrastructure support teams and between those teams and other software engineering stakeholders?

For this phase, the scope chosen was the set of activities required to move an application from one data center to a new primary data center on a different set of physical infrastructure. Frequently, applications must make coding or other changes to adhere to new technology standards that are instituted at the new center. The process of bringing business applications up to those new standards was chosen as a very specific area within which to study and attempt to improve team collaboration and knowledge sharing. Specific deliverables of the process included the following artifacts: a technical architecture requirements document, a technical architecture design that includes required infrastructure, and an environment build document that documents configuration requirements for system administrators to follow when building machines for the target application.

Problems with this process included frequent handoffs, lost time when information is shared but not received, sequential steps, bottlenecks, and multiple points of contact. All of these problems are barriers to knowledge sharing. Improvements to the process are expected to reduce cycle time for the process, reduce error, increase satisfaction both with the development teams and the infrastructure support teams, and increase knowledge sharing and collaboration between subject matter experts and between service providers and their internal customers.

While participating in the process improvement activities, the role of the action researcher in this study was to contribute to the design of a collaborative technology to support a streamlined process. An important part of this process analysis was analyzing the data flows in and out of the process. An inventory exercise was conducted, identifying what data and information was needed at each point in the flow of the process. Data owners and data receivers were also identified. The process timeline was identified as one to six months depending on the technical requirements of the project and the individuals involved. With multiple handoff points, there was increased time and the potential for error introduced. The process was very human-intensive, so progression was directly dependent on the availability, responsiveness, and knowledge of each person in the process.

Resulting recommendations for process improvement following evaluation of this phase included the following:

- Reduce the number of analysts involved. By combining roles, handoffs between project initiation and implementation is eliminated. This also increases the likelihood that the information and knowledge used for infrastructure configuration and setup is more accurate, as the same person has accountability from the beginning of the process until the end.
- Automate data collection throughout the process cycle. This ensures consistency of data collected across process iterations. It also ensures that input received across project implementations will be shared.

IV. Conclusion

Observations in the first cycle revealed that, in an environment where revenue generating applications are given priority for development resources, limited support for development of a knowledge sharing system is available. A creative framework that utilizes existing technology with minimal development for a very specific problem domain is the most effective way to address cross-boundary knowledge sharing concerns. However, to be most effective the scope must be bounded by specific functional processes, which are the ties that bind a community of practice.

In addition to the technology layer, an effective approach to increased knowledge sharing and collaboration must consider the multiple processes that make up a knowledge workers daily context, and the knowledge that must flow in order to support that workflow. The organization of people and process must be considered in order for knowledge creation to be efficient and effective. The layers in the knowledge management sandcone depicted in Fig. 1 should be reordered as shown in Fig. 2. Process must first enable knowledge sharing, with organizational changes considered a part of process improvement. The technology layer can then be applied. Only then is there enablement for cultural change.

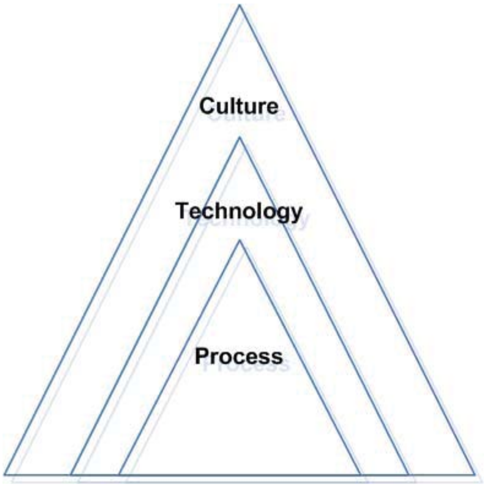


Figure 2. Reordered layers of the knowledge management sandcone.

The recommended strategy for knowledge sharing resulting from this study includes:

- IT organizations must recognize the importance of processes and the direct impact a process can have on the effectiveness of a collaborative environment.
- IT organizations should develop a knowledge management strategy from a community of practice perspective. Through workflow automation, a repository of business rules and knowledge essential to process flow can be developed for the participants in a defined community with common goals.
- IT Organizations should identify and use existing technologies to facilitate knowledge sharing without introducing a new toolset.

V. FUTURE WORK

Callele, et. al. describe how software engineering in the context of video game development is not well understood, thus the application of process improvement to this domain is limited. With a diverse set of roles in the game development process, the application of the knowledge sharing recommendations presented here is an opportunity [18].

The game development process can be broken down into two large phases: pre-production and production. Pre-production involves all required tasks to create the game before actual coding starts. Production involves all tasks that are required to implement the game. In the beginning of the pre-production phase, the lead game designer must successfully create and share the idea of what the game should become. A game design document is created by the lead designer to help convey these ideas. This document, which expresses the game concept in a non-technical manner, is referred to throughout the entire game development process.

After the initial pitch, a design team is created in order to further plan the look and feel of the game. This requires many revisions to the game design document. It is imperative that the design document express the ideas of the design team as clearly as possible because from this document the development team will later create a software requirements document.

In the production phase, the design document is handed off to the development team. The team starts by taking the design document that was created in preproduction and then formally creating an SRS. Requirements such as art, music, levels, artificial intelligence, and other factors are all considered. When this document is completed the information must be shared through the entire development team in order to actually start implementation. In implementation, all art, music, code, and other assets are actually created from the SRS. Post implementation, the development team is still tasked with controlling maintenance for any discovered errors in the game. The entire game team is expected to also reflect and record the positive and negative aspects of the project [19].

As in any software design and development process, a large amount of knowledge must be shared in order to create an accurate design and implementation that meets the customers' expectations. If the integrity of knowledge is not preserved during the game design and development process, the resulting game is not likely to satisfy the original vision and intent.

The most critical area of knowledge sharing that, poorly handled, could cause the most damage to a game development project is the shift from pre-production to production. Problems may arise in the creation of the requirements document by translating the game design document into more specific and technical terms. Even though the game design document is written as specifically as possible, the development team can still misinterpret the meaning of what was documented. It is true that the development team can meet with the design team and try to completely understand the true meaning in the game design document, but this causes a delay in the start of implementation which can be detrimental to the project. It is also possible for the team to create an incorrect SRS and begin development using this document. This can cause a delay in product release [18].

At any point during coding, the development team may encounter a situation where ideas of the design team cannot be actually implemented in a way that would is considered enjoyable to a user. When this occurs, the information must be shared with the design team and then the design document must be updated to reflect any changes. After that the requirements document must be updated to reflect the same changes. Within this cascading flow of updates to project information, there is much opportunity for error and miscommunication [19].

The differences in the game design and development process from more traditional software development offers an opportunity to apply recommended knowledge sharing improvement practices. Callele et. al. supports the idea that process improvement will increase the likelihood that knowledge is captured and disseminated when needed [18].

ACKNOWLEDGMENT

The views expressed herein are those of the above named authors and not necessarily those of their current or former employers.

REFERENCES

[1] Hustad, E. (2004). Knowledge networking in global organizations: the transfer of knowledge. *Proceedings of the 2004 SIGMIS Conference on Computer Personnel Research: Careers, Culture, and Ethics in a NetworkedEnvironment*, 55-64. doi:10.1145/982372.982384

[2] Bossen, C., & Dalsgaard, P. (2005) Conceptualization and appropriation: the evolving use of a collaborative knowledge management system, *Proceedings of the 4th decennial conference on Critical computing: between sense and* sensibility*, 99-108*. doi: 10.1145/1094562.1094574

[3] Teigland, R., & Wasko, M. (2000). Creative ties and ties that bind: examining the impact of weak ties on individual performance, *Proceedings of the Twenty-First International Conference on Information Systems*, 313-328. Retrieved from http://portal.acm.org/citation.cfm?id=359640.359758

[4] Rus, I., & Lindvall, M. (2002). Guest editors introduction: Knowledge management in software engineering. IEEE Software, 19, 26-38. doi:10.1109/MS.2002.1003450

[5] Ye, Y. (2006). Supporting software development as knowledge-intensive and collaborative activity. *Proceedings of the 2006 international Workshop on Workshop on interdisciplinary Software Engineering Research, 15-22.* doi:10.1145/1137661.1137666

[6] Jansen, A., & Bosch, J. (2005). Software architecture as a set of architectural design decisions. *Proceedings of the Fifth Working IEEE/IFIP Conference on Software Architecture*. 109-120. doi: 10.1109/WICSA.2005.61

[7] Nonaka, I. (1994). A dynamic theory of organization knowledge creation. *Organization Science, 5*(1), 14-37. doi:10.1287/orsc.5.1.14

[8] Lee, S., Braunsdorf, K. & Shiva, S. (2010). Knowledge engineering to visualize complexity for legacy modernization planning, *Proceedings of the Twenty-Second International Conference on Software Engineering and Knowledge Engineering*, 331-334.

[9] A. Agostini, S. Albolino, G. De Michelis, F. De Paoli, and R. Dondi,. “Simulating knowledge discovery and sharing,” *Proceedings of the 2003 international ACM SIGGROUP Conference on Supporting Group Work*, ACM, New York, 2003, pp. 248-257.

[10] D. L. Hansen, “Knowledge sharing, maintenance, and use in online support communities,” *CHI '06 Extended Abstracts on Human Factors in Computing Systems* ACM, New York, 2006, pp. 1751-1754.

[11] Marks, P. Polak, S. McCoy, and D. Galletta, “How Managerial Prompting, Group Identification, and Social Value Orientation Affect Knowledge-Sharing Behavior, “ *Communications of the ACM*, February 2008, pp. 60-65.

[12] J. Ezingeard, S. Leigh, and R. Chandler -Wilde. “Knowledge management at Ernst & Young UK: getting value through knowledge flows,” *Proceedings of the Twenty First international Conference on information Systems*. ACM, Atlanta, GA, 2000. pp. 807-822.

[13] Lee, S., & Shiva, S. (2009). A novel approach to knowledge sharing in software systems engineering, *Proceedings of the Fourth IEEE International Conference on Global Software Engineering*, 376-381. doi:10.1109/ICGSE.2009.59

[14] Kock, N., & Davison, R. (2003). Can lean media support knowledge sharing? Investigating a hidden advantage of process improvement. *IEEE Transactions on Engineering Management*, *50*(2), 151-163. doi:10.1109/TEM.2003.810830

[15] Jennex, M., & L. Olfman. (2004) Assessing Knowledge Management Success/Effectiveness Models. *Proceedings of the Thirty-Seventh Hawaii International Conference on System Sciences*, *8*, 236-245.

[16] Baskerville, R. (1999). Investigating information systems with action research. *Communications of the Association on Information Systems, 2.* Retrieved on March 9 2011 from http://www.cis.gsu.edu/~rbaskerv/CAIS_2_19/CAIS_2_19.html

[17] Lee, S. & Shiva, S. (2010). An approach to overcoming knowledge sharing challenges in a corporate IT environment, *Proceedings of the Fifth IEEE International Conference on Global Software Engineering*, 342-346. doi:10.1109/ICGSE.2010.47

[18] Callele, D., Neufeld, E., Schneider, K. “Requirements Engineering and the Creative Process in the Video Game Industry,” In Proceedings of the 13th IEEE International Conference on Requirements Engineering (RE '05). IEEE Computer Society, Washington, DC, USA, 240-252. DOI=10.1109/RE.2005.58

[19] Lohmann, S. & Niesenhaus, J. Towards Continuous Integration of Knowledge Management into Game Development. Proceedings of IKNOW ’08 and IMEDIA '08 Graz, Austria, September 35,2008.

Automatic Acquisition of *isA* Relationships from Web Tables

Norah Alrayes
School of Computing Science
Simon Fraser University
Burnaby, BC, Canada
naa27@sfu.ca

Wo-Shun Luk
School of Computing Science
Simon Fraser University
Burnaby, BC, Canada
woshun@sfu.ca

***Abstract*—Automatic acquisition of *isA* relationships is an important process in knowledge engineering, with significant applications in natural language processing and software engineering. Until now, most research on searches for *isA* relationships has been based on the lexical patterns in free text. While the precision of this strategy is generally high, the recall is low. This paper proposes a new source from which *isA* relationships may be acquired: web tables that come with table titles. A number of natural language processing methods are proposed for extracting the label from the table title that most accurately annotates a group of column/row headers, independent of the domain to which they belong. Experiments were conducted on summary tables from Statistics Canada and Statistics Austria, which show our extraction algorithm achieving an accuracy of 92%.**

Keywords-isA relationship; web tables; statistical tables; WordNet; Natural Language Processing

I. INTRODUCTION

An *isA* relationship is sustained between two concepts, say X and Y, if concept Y can be generalized to concept X, or equivalently concept X being a specialization of concept Y. Symbolically, it is written as (X *isA* Y). As an example, *dog isA animal* is a relationship between dog and animal such that the concept *dog* may be generalized to the concept *animal*. Linguistically, X is called a *hyponym* of Y, and Y a *hypernym* of X. An *isA* relationship is transitive, which means if X can be generalized to Y and Y to Z, then the *isA* relationship X *isA* Z holds.

WordNet [*1*] is probably the best known and most popular source of *isA* relationships. Besides applications of WordNet to many problems in natural language processing [*2*], it has become an important tool for information processing. In particular, WordNet has been used for identifying semantic similarity between keywords in matching documents in information retrieval [*3*], and between attributes in matching database schemas in schema integration [*4*][*5*].

As popular as WordNet is, it is not without shortcomings. The main concern about manually crafted semantic taxonomy, such as WordNet, is its high cost in maintaining and extending its coverage. Following a seminal paper [*6*] on automatically extracting *isA* relationships from free text, a number of papers have been published on automatic discovery of *isA* relationships based on lexical patterns [*7*][*8*]. Meanwhile, others are looking for extraction of *isA* relationships from more specialized sources such as Wikipedia, YAGO or DBpedia [*9*][*10*][*11*][*12*].

This paper investigates a new source where *isA* relationships are automatically and accurately discovered, namely tables published on the Web. A column in a table contains words, i.e., *hyponyms*, the concept of which they represent, which have an *isA* relationship with the concept represented by the column header, i.e., the *hypernym*. According to some estimates [*13*] there are roughly 100 million data-rich tables on the Web. However, the column headers are not precise enough; they might be very brief and might not lead to important knowledge. In [*12*], column headers are ignored, and the label describing the values in the column semantically is derived from web searching.

Here, the focus is on those web tables that come with table titles, which include those published by public organizations such as government agencies, particularly national statistical agencies. The objective of this research is to show how domain-independent *isA* relationships may be discovered from these tables. These tables are more complex than tables analyzed in [*12*]. Instead of multi-columns, as in most common tables, they feature multi-dimensions, where each quantitative item in the table is associated with a header from each dimension. The set of all column and row headers can be partitioned into a number of groups, each of which corresponds to a particular dimension (see the example in Section II, which gives a more details about the concept of multidimensional tables). This work builds on our previous work [*14*], which develops an algorithm for the partitioning of headers according to the dimensions they belong to. In this paper, a scheme is developed to acquire a label, mostly from the table title, that forms the *isA* relationship with a given group of dimension headers.

The rest of the paper is organized as follows. Section II explains the background of this research, in particular the terminology in table processing. Section III introduces the summarization rule to determine whether the headers of a dimension truly share a commonality for which a label can be found. A general scheme to locate a label for a group of headers is described in Section IV. Section V describes in detail how to select the most appropriate label from a set of candidate labels for a given group of headers, i.e., dimensions. Experimental results are presented in Section VI. Section VII is the conclusion and discussion of future work.

II. INTRODUCING MULTIDIMENSIONAL TABLES

Multidimensional tables are designed with a specific common layout that is standardized for readers. Fig. 1 shows the main components of a multidimensional table, including the table title, column/row headings, and data region.

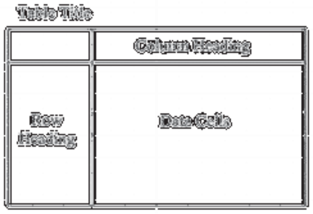


Figure 1. The main components of a multidimensional table

- The table title summarizes the information stored in the table. Typically, titles are carefully prepared by a professional organization. The table title usually consists of two components: 1) description of the data stored in the table cells, and 2) the labels for the included dimensions. The title is used to inform readers of the purpose of a particular table. According to Finland Statistics articles [*15*], a table title has four important elements: the title, which identifies the population covered; the variables described in the table and their classification; the time period of the observations; and the units of measurement. In short, the table title should provide a general description of the table.

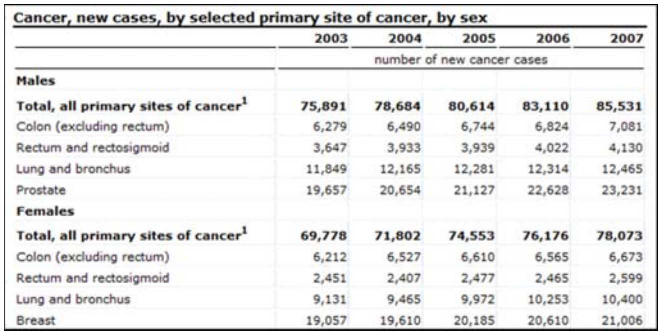
  Often the title elements are connected together with a preposition such as "by", or with punctuation such as commas. For example, in Fig. 2, "selected primary site of cancer" and "sex" both follow the preposition "by". This may not always be the case, however, such as in the table title "Workers who use an official language most often or regularly at work, by province and territory", where one of the dimensions should be labeled as "official language". Still, the majority of the dimension labels appear after the preposition.

- Multidimensional tables have two kinds of headings: the row heading and the column heading. The text string contained in a cell in either heading is called a header. The headers are displayed to the reader such that they are recognized as what they stand for. For example, Fig. 2 shows that some headers are associated with data cells, while others are not. They can be shown in different font types. These visual clues will help the reader to comprehend the meaning. For example, The "Males" header is associated with the phrase "by sex" in the table title, while the header "Total, all primary sites of cancer" is associated with another part of the title.

- The data region, i.e., the non-heading region of the table, is populated with data cells containing numeric data items. Each data cell is indexed horizontally by a row header and vertically by a column header.

A careful design of a multidimensional table will enable a human reader to associate a data item found in the data region with a number of headers in the heading regions. For example, the data item 6,279 is associated with three headers: Colon (excluding rectum), Male, and 2003. Thus, this is a 3-dimensional table, where the headers are partitioned into three groups, or dimensions, which are:

- Selected primary site of cancer: Colon (excluding rectum), Rectum and rectosigmoid, Lung and bronchus, Prostate, Breast.
- Sex: Male, Female.
- Year: 2003, 2004, 2005, 2006, 2007



Cancer, new cases, by selected primary site of cancer, by sex

	2003	2004	2005	2006	2007
	number of new cancer cases				
Males					
Total, all primary sites of cancer[1]	**75,891**	**78,684**	**80,614**	**83,110**	**85,531**
Colon (excluding rectum)	6,279	6,490	6,744	6,824	7,081
Rectum and rectosigmoid	3,647	3,933	3,939	4,022	4,130
Lung and bronchus	11,849	12,165	12,281	12,314	12,465
Prostate	19,657	20,654	21,127	22,628	23,231
Females					
Total, all primary sites of cancer[1]	**69,778**	**71,802**	**74,553**	**76,176**	**78,073**
Colon (excluding rectum)	6,212	6,527	6,610	6,565	6,673
Rectum and rectosigmoid	2,451	2,407	2,477	2,465	2,599
Lung and bronchus	9,131	9,465	9,972	10,253	10,400
Breast	19,057	19,610	20,185	20,610	21,006

Figure 2. Sample of a statistical multidimensional table

Most multidimensional tables feature 3-4 dimensions. A higher dimensional table is sometimes broken into a number of lower dimensional tables. Due to lack of space, we will not deal with table series here, although in [*16*] a scheme to handle table series is described.

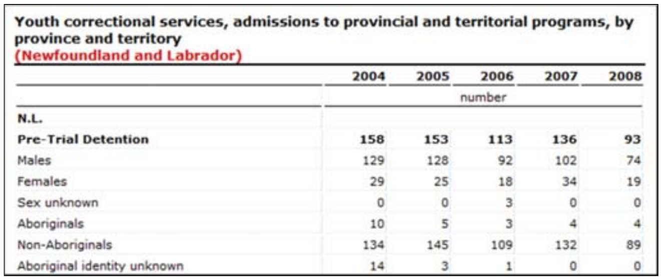
For each dimension, there is a dimension name and a number of dimension headers. Between the name and each header is an *isA* relationship. While it is a relatively easy task for humans, especially experienced readers, to partition the headers into dimensions and derive a label from the table title as the dimension name, it is a nontrivial job for a machine.

The algorithm described in [*14*] to extract the headers from the row/column headings and partition them into dimensions relies on the visual clues embedded in the multidimensional table. Also [*14*] introduces a simple technique for assigning a label to the dimensions, which is the column label associated with the dimensional headers. Clearly, this simple technique is not sufficiently accurate to uncover all *isA* relationships in the table without access to the table title. For example, in Fig. 2 the label "sex" will not be easily discovered as the dimension label, since it is not found anywhere other than the table title. Much of the present paper is to develop techniques to process the table title as a natural language entity to search for a label for a number of dimensions located by the algorithm described in Sections IV and V.

III. VALIDATING THE DIMENSION HEADERS

In [*14*] the dimensions including their headers are extracted according to visual clues. Nevertheless, our research found that the dimension headers are not always in the same category and hence should not be covered by the same label. Let us consider the example in Fig. 3. The group of dimension headers {Males, Females, Sex unknown, Aboriginals, Non-Aboriginals, Aboriginal identity unknown} have the same visual clues but are not from the same group, and no label can represent them as one collection.

To confirm that the dimension headers are related, we need to use the numeric values in the table data region. This is done by summing the numbers associated with the dimension headers and one of the columns. In our example, the sum of the numeric values under column "2004" associated with the previously mentioned dimension headers is 316. The sum exceeds the numeric value associated with the dimension summary header, which is 158. Thus, for our purpose, this proves that it is not a collection and it is not necessary to find a label for them. On the other hand, if the sum is less than the total under every column, as in the case of the table in Fig. 2, the group of the dimension headers is acceptable as they are only a proper subset. We called this *the summarization rule*.



Youth correctional services, admissions to provincial and territorial programs, by province and territory
(Newfoundland and Labrador)

	2004	2005	2006	2007	2008
			number		
N.L.					
Pre-Trial Detention	**158**	**153**	**113**	**136**	**93**
Males	129	128	92	102	74
Females	29	25	18	34	19
Sex unknown	0	0	3	0	0
Aboriginals	10	5	3	4	4
Non-Aboriginals	134	145	109	132	89
Aboriginal identity unknown	14	3	1	0	0

Figure 3. Statistics table shows the summation problem

IV. FINDING *isA* RELATIONSHIPS - GENERAL SCHEME

In this section and in Section V, the main problem of this research is tackled: given a set of headers that belongs to a dimension $\{m_1, ..., m_q\}$, a label needs to be found such that an *isA* relationship exists between the label and m_i, 1<= i <= q. A general scheme is presented in the algorithm below, followed by more detailed explanation of each step.

```
for each group of dimension headers $\{m_1, ..., m_q\}$ do
    if $\{m_1, ..., m_q\}$ are from domains (time/age) then
        merge this dimension with the domain list and assign label $L$= "time" or "age"
    quit}
    else {
    construct the set of candidates $\{c_1, \dots\dots, c_p\}$ for isA relationship
    if an isA relationship is identified for $\{m_1, ..., m_q\}$ from $\{ c_1, \dots\dots, c_p\}$ then
        assign the successful candidate as a label $L = c_i$ and merge this dimension with the domain list.
    quit}
    else {
    If the external domain list $\{d_1, ..., d_n\}$ contains a proper portion of $\{m_1, ..., m_q\}$ then
        assign the domain name as a label $L = Ld_i$ and merge this dimension with the domain list.
    quit}
    else {
    if the default candidate $\{c_1, \dots\dots, c_x\}$ is available then
        assign the candidate as a label $L = c_i$ and merge this dimension with the domain list.
    quit}
```

1) Common dimensions

Some dimensions may contain special kinds of headers; for example, time and age. These headers can easily be detected because what they represent can be understood directly without having to perform any further processing. Since these types of header may not have any linguistic meaning, being numbers or special characters, they can be identified by the use of some temporal templates. A label such as *Time* or *Age* will be assigned directly.

2) Candidate set

The candidate set consists of a number of labels, which are selected on the basis that any of them is likely to be the ultimate label we are looking for. It is constructed from two sources: the table title and the dimension summary header. Since the construction of the candidate set and the identification of the most suitable candidate from the set are very complex, the discussion is deferred to Section V.

3) External domain list

Often, dimensions share the same domain. For example, the names of provinces and territories belong to the same domain. The dimensions in many tables would contain headers drawn from this domain. Consequently, we maintain a list of domains as we process the collection of tables. Whenever a dimension is derived, it is checked against the domain list to verify whether or not it shares the same domain as the other dimensions that have been processed. Occasionally, the matching process may help to locate a label when using heuristics 1 or 2 fails to provide an appropriate label for a dimension.

The domain can also be shared with full names and abbreviations. For example, suppose the headers of a dimension are abbreviations, like NB, and a domain in the external domain list contains New Brunswick. If the two dimensions are found to share the same domain, the former header will be reverted back to its full name, since the latter is syntactically closest to the abbreviation.

To define the matching process, the two lists—the dimension headers and the domain list—are syntactically compared. First the labels are checked, and if the two lists carry the same labels, it is possible to determine syntactically whether or not the two groups of headers intersect with a low threshold to do the merge. The union of these groups is not applied directly without first confirming the label, because it cannot be guaranteed that the labels have been correctly assigned. On the other hand, if the labels are different, they can be merged by syntactically intersecting the dimension headers with a high threshold.

4) *Default*

Occasionally, the methods outlined in 1, 2, and 3 may fail to identify an appropriate label for the dimensions. Nevertheless, there may be a remaining list from the candidate list in heuristic 2 that has not been assigned as a label to any of the dimensions. The candidates derived from the dimension summary headers have the highest priority to be assigned as a label, because of the nature of this location in regards to the dimension. If a dimension summary header was not found, then the non-assigned candidate from the table title is assigned as the label without any processing.

V. IDENTIFYING *isA* RELATIONSHIPS FROM A CANDIDATE SET

In this section, the candidate set is constructed using the table title and the dimension summary header. One of the candidates in this set can be the dimension label. This paper introduces a special algorithm to reach this goal. The algorithm is run in three steps.

In short, there should be a candidate set, i.e., $\{c_1, \dots, c_p\}$, and the dimension headers; i.e., $\{m_1,...,m_q\}$. The problem is to select a c_i, 1<=i<=p, which is the most appropriate label for the set of headers $\{m_1,...,m_q\}$. For the pre-processing (step A), a list of ancestors is identified from the WordNet taxonomy that is deemed to be semantically close to the dimension headers as a whole, i.e., $\{m_1, ..., m_q\}$. Each ancestor in the list is essentially a proxy for the dimension headers. In step B, the candidate set is constructed in three different methods. The last step C is to compute the semantic similarity between each ancestor in the list and each candidate in the candidate set, i.e., $\{c_1, \dots, c_p\}$, to select the candidate with the highest similarity score to be the label for the dimension. These steps are explained in detail below.

A. *Pre-Processing*

The headers $\{m_1,...,m_q\}$ in the dimensions could be phrases or short sentences and could be quite large. They could also be too diverse semantically. When the headers are phrases or short sentences, it is not known which word of the header is more dominant. Therefore, it is not appropriate to directly match the headers to the candidate set. This decision is justified by the experimental results described in Section VI. By this pre-processing, the set of the dimension headers $\{m_1,...,m_q\}$ is replaced with another set of words, i.e., common ancestors $\{a_1,...,a_z\}$, that semantically represents this set of headers, such that a common ancestor a_i represents the majority of or all the headers. This set of common ancestors is smaller and semantically closer to the candidate set, to be constructed in step B.

To proceed, each header in the dimension must first be represented within the WordNet taxonomy. It is reasonable to assume that headers in the same dimension should share a common ancestor. Two important factors must be considered when choosing a common ancestor for the dimension headers: the size of the coverage (c), where c is the set of headers in the dimension that retrieves this ancestor, and height (h), which is the median distance between the common ancestors of the headers in c. The greater the height, the more tenuous is the ancestor-descendant relationship. Conversely, the larger the coverage, the more relevant is the ancestor to the header set as a whole, as demonstrated in Fig. 4. For each ancestor matching at least 25% of the headers in the set, the common ancestor score is defined as:

$$\text{Common ancestor score} = |c|/h \tag{1}$$

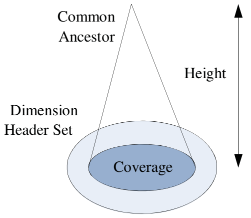


Figure 4. Height and Coverage of Common Ancestor

The list of ancestors is sorted according to their common ancestor score, and the low scoring ancestors will be dropped from the list. Because some headers in the dimension are phrases or short sentences, it can be difficult to collect the common ancestor for these headers, since one cannot be sure which word is most closely related to the other headers in the set. WordNet contains single words and some general purpose/ common phrases. Since the headers are mostly phrases defined by the table designer and are not common or general purpose WordNet synset, therefore, for WordNet to retrieve them, each dimension header needs to be divided to the appropriate WordNet synset. Also some ancestors may be retrieved more than once for the same group, for the reason given above. Therefore, the shortest height between them is chosen. The following algorithm is a summary of the pre-processing step.

```
for each group of dimension headers $\{m_1, ..., m_q\}$ do
    for each $m_i$ collect ancestor group $\{a_1, ..., a_{zi}\}$
    for each Ancestors $a_i$ in $\{a_1,..., a_z\}$ collect the
    ancestor score
        $a_i$ score = $|c|/h$
    sort the Ancestors according to its high score
    return ancestors and their score $\{a_1, ..., a_z\}$
```

The pre-processing step is not needed in one situation, when the dimension headers are listed in the table title without any common words to represent them. This can happen when the dimension headers are less than or equal to 3. In such cases, the list of headers is compared directly with the candidate set.

B. *Construction of the Candidate Set*

The candidate set is formed with labels from the two main sources: the table title, which was introduced in Section II, and the dimension summary header for the dimension header set. The latter is located right above the headers in either column or row headings. It is distinguished from the headers by font type/size. This summary header is often just a single word, but occasionally it contains the label that appropriately describes the dimension header set, e.g., the phrase “total, all primary sites of cancer” in Fig. 2. The dimension summary header is ignored when it is an aggregate word only, such as “total”.

The table title and the dimension summary header are most unlikely full sentences. Many resemble noun phrases. Therefore, they need to be partitioned to appropriate sizes in order to find a label to be matched with the ancestor set. Those labels could be phrases or words. This study has developed three different ways of partitioning in order to find the best method to choose the candidate set. The candidates can be partitioned into NLP chunks, or they can be partitioned to the WordNet synset and the full chunk that would serve as a label can then be retrieved. The third method is a combination of the previous two.

1) Case #1: NLP chunks

Chunking any sentence/phrase, in this case the table title and the dimension summary header, depends on the part of speech tagging of words in the sentence. These chunks are considered as candidates for the label that covers the dimension headers. We refer to "Statistical parsing of English sentences" [*17*] to divide the candidates into appropriate noun phrases, verb phrases and words. For example, for the table title in Fig. 2, the chunks are {Cancer, new cases, selected primary site of cancer, sex}, which become candidates in the candidate set.

2) Case #2: single words

Each candidate in the candidate set will be a single WordNet synset that is found in the table title and the dimension summary header.

3) Case #3: sliding window chunks

A sliding window technique is formed on the table title and the dimension summary header. The candidates in the candidate set are of size 1, 2 or 3 WordNet synsets picked from this technique.

C. Assigning Label from the Candidate Set

This step computes the semantic similarity between the list of ancestors, derived in step A, and the candidates in the candidate set, derived in step B. This is done using the semantic score presented in "WordNet-based semantic similarity measurement" [*18*]. The semantic similarities are computed between the ancestors in the ancestor set and the candidate set, beginning with the highest scoring ancestor and the candidates that appear after the prepositions in the table title. The candidate that matches one of the ancestors is selected as a label. This procedure is done in two steps, depending on the type of candidates in the candidate set. The steps are presented in the following algorithm.

```
for each $a_i$ in $\{a_1, ..., a_z\}$
    for each candidate in $\{c_1, ......, c_p\}$
        divide each $c_i$ to each WordNet synset $\{w_1, ......, w_g\}$
        for each $\{w_1, ......, w_g\}$
            identify $LCA$ between $a_i$ and $w_i$, the depth $d$ for $a_i$ and $w_i$,
                $Sim= LCA / (d_{a_i} + d_{w_i})$
        end
    end
    if Case #1 or Case#3 then{
            compute the similarity for the ancestor $a_i$ and the full chunk
                $p_1 =\sum Sim (\{w_1, ......, w_g\}$and $a_i)$
                $p_2 = Max\ Sim (\{w_1, ......, w_g\}$and $a_i)$
                $T= (p_1+p_2)/ (l_1+l_2)$
            if $T \geq$ certain threshold then{
                $L = c_i$
                quit loop}
    else if Case#2 then{
            if (Max Sim $\geq$ certain threshold) then{
                $L = c_i$
                quit loop}
```

When the candidates are single WordNet synsets, the semantic similarity between each ancestor and each candidate is measured according to Wu and Palmer [*19*], as shown in (2), where *LCA* is the least common ancestor depth in WordNet taxonomy between two WordNet synsets; i.e., the ancestor and the candidate, d_{a_i} is the depth for the ancestor, and d_{w_i} is the depth for the candidate word.

$$Sim= LCA / (d_{a_i} + d_{w_i}) \qquad (2)$$

When the candidates are chunks, the chunks must be divided into single WordNet synset and their semantic similarity computed with the ancestors. This is followed with a computing of the full similarity of the full chunk, since any semantic similarity function cannot compute the similarity directly between the phrase and words, because the chunk contains more than one WordNet synset and the ancestor is one WordNet synset. Therefore, it is necessary to ensure that the semantic similarity measure considers comparisons of a group of WordNet synsets taken together.

Since (2) computes the similarity between two WordNet synsets, the similarity scores for the case when the candidate is a phrase must be collected. According to [*18*] the total similarity, *T,* is equal to the summation of the similarities between the phrase and the ancestor, *p1,* and *p2* is the highest similarity score between the ancestor and a word in the phrase. This is divided by the summation of their length *l1* and *l2*.

$$T= (p1+p2)/ (l1+l2) \qquad (3)$$

VI. Experimental Results

Our system was tested with multidimensional tables from Statistics Canada's website of summary tables [*20*] and from Statistics Austria website [*21*]. The tables are freely accessible and belong to a few national statistical agencies that continue to publish in HTML. They contain invaluable quantities of *isA* relationships. More importantly, the tables are valuable because they cover a wide range of topics. Our test dataset contained some 305 randomly selected tables, containing 781 dimensions, which were domain-independent and covering such topics as education, construction, household, travel, and languages. By implementing the algorithms, we found that we are able to extract 92% of *isA* relationships successfully.

Different techniques were implemented to reach the goal of this research. This section of the paper looks deeper into the results to determine their effectiveness. First, the pre-processing step described in Section V was found crucial to identification of the most accurate label. Alternative experiments were conducted by directly matching between the candidate set and the dimension headers, while skipping the pre-processing step, and the success rate dropped by a wide margin, to 69%.

The experimental results also demonstrated how essential and critical the table title is. It was the most important source for labeling all the major dimensions. It was used to successfully label almost 51% of 92% of the total successfully labeled dimensions. On the other hand, 1.6% of the successfully labeled major dimensions were labeled by the dimension summary header. The majority of the other 44% labeled dimensions were common dimensions such as time, since this kind of dimension is very common in the multidimensional tables.

Experimental data also compare the effectiveness of three different ways of analyzing the table titles, resulting in three different candidate sets. The success rates for these cases are shown in Table I.

TABLE I: EXPERIMENTAL RESULTS

Candidate set	Success rate
Case #1	87%
Case #2	92%
Case#3	92%

We found that the best case for successfully computing the semantic similarities between the ancestors and the candidates was by representing the candidates as a single WordNet synset (Case #2).

By using Case #3 type of candidates, the sliding window size 1 always had the highest scores, which is also represented in the candidates in Case #2. The similarity between the ancestors and the candidates as chunks usually had lower scores, since the ancestors are a single WordNet synset, and comparing them to a chunk means unnecessary words were included the score. The results were ambiguous when comparing a chunk of size 1 with a chunk of size 3.

Case #1 had the lower success rate, since the candidates were fixed chunk size. In addition, these experiments depended on the success in chunking the candidate set. The procedure might not always retrieve a good list of chunks since some of the chunks would contain two words, each of which would label one of the dimensions.

VII. CONCLUSION AND FUTURE WORK

This paper has demonstrated that web tables collectively are a valuable source for acquiring *isA* relationships. It proposed a rule of summarization to verify whether or not a group of table headers share some commonalities, so that a label can be found that describes them collectively. Different ways were examined for processing the table title as a natural language entity to extract a label for a group of table dimensions. Our work can be extended to millions of web tables where table titles are not located in the immediate neighborhoods of the tables. Strategies for locating a proper table title in a web page where a table is found are currently being developed.

REFERENCES

[1] C. Fellbaum, "WordNet: an electronic lexical database," *Cambridge, MA: MIT Press.*, 1998.

[2] "Usage of WordNet in Natural Language Processing systems," in *Proceedings of COLING-ACL Workshop*, Montreal, Canada., 1998.

[3] R. Mandala, T. Takenobu, and T. Hozumi, "The use of WordNet in information retrieval," in *Proc. Conf. Use of WordNet in Natural Language Processing Systems*, 1998, pp. 31-37.

[4] F. Hakimpour and A. Geppert, "Resolving semantic heterogeneity in schema integration: an ontology based approach," in *Proc. of the Intl. Conf. On Formal Ontologies in Information Systems*, Ogunquit, Maine, USA, 2001, pp. 297--308.

[5] N. F. Noy, "Semantic integration: A survey of ontology-based approaches," *SIGMOD Record*, vol. 33, no. 4, 2004.

[6] M. A. Hearst, "Automatic acquisition of hyponyms from large text corpora," in *In Proc. of the 14th Intl. Conf. on Computational Linguistics*, Nantes, France, 1992, pp. 539--545.

[7] R. Snow, D. Jurafsky, and A.Y. Ng, "Learning syntactic patterns for automatic hypernym discovery," in *Advances in Neural Information Processing Systems*, 2005.

[8] A. Ritter, S. Soderland, and O. Etzioni, "What is this, anyway: Automatic hypernym discovery," in *Proc. of AAAI Spring Symposium on Learning by Reading and Learning to Read*, 2009, pp. 88-93.

[9] A. Sumida and K. Torisawa, "Hacking Wikipedia for hyponymy relation acquisition," in *Proc. of the Third Intl. Joint Conf. on Natural Language Processing*, 2008.

[10] S. Sarawagi and S. Chakrabarti G. Limaye, "Annotating and searching web tables using entities, types and relationships," in *36th Intl. Conf. on Very Large Data Bases*, Singapore, 2010.

[11] A.Bhattacharjee and H. Jamil M. Amin, "Wikipedia driven autonomous label assignment in wrapper induced tables with missing column names," in *Proc. of the 2010 ACM Symposium on Applied Computing*, Switzerland, 2010.

[12] A. Halevy, J. Madhavan, M. Paşca, W. Shen, F. Wu, G. Miao and C. Wu P. Venetis, "Recovering semantics of tables on the Web," in *The 37th Intl. Conf. on Very Large Data Bases*, Seattle, Washington, 2011.

[13] M.J. Cafarella, A. Halevy, D.Z. Wang, E. Wu and Y. Zhang, "Uncovering the relational web," in *Proc. of the 11th Intl. Workshop on Web and Databases*, Vancouver, Canada, 2008.

[14] W. Luk and P. Leung, "Extraction of semantics from web statistical tables," in *IEEE/WIC/ACM Intl. Workshop on Semantic Web Mining and Reasonin*, Beijing, China, 2004.

[15] Statistics Finland. (2010, September) Statistics Finland - Online statistics course - How to read and use statistics - Ways of displaying statistics tables. [Online]. http://www.stat.fi/tup/verkkokoulu/data/tlkt/03/01/index_en.html

[16] Norah Alrayes, "Extracting data cubes from multidimensional tables on the Web," M.Sc. Thesis, 2011.

[17] R. Northedge. (2011) Code project: Statistical parsing of English sentences. [Online]. http://www.codeproject.com/Articles/12109/Statistical-parsing-of-English-sentences

[18] T. Simpson and T. Dao. (2010, Jan) Code project: WordNet-based semantic similarity measurement. [Online]. http://www.codeproject.com/KB/string/semanticsimilaritywordnet.aspx?msg=2776502

[19] Z. Wu and M. Palmer, "Verb semantics and lexical selection," in *Proc. of the 32nd Annual Meeting of the Associations for Computational Linguistics*, 1994, pp. 133-138.

[20] Statistics Canada. [Online]. http://www.statcan.gc.ca

[21] Statistics Austria. [Online]. http://www.statistik.at/

A light weight alternative for OLAP

Hugo Cordeiro
Infoway - Teresina - PI - Brazil
hugo@infoway-pi.com.br

Jackson Cassimiro
Infoway - Teresina - PI - Brazil
jackson@infoway-pi.com.br

Erick Passos
IFPI - Teresina - PI - Brazil
erickpassos@ifpi.edu.br

Abstract

OLAP applications allow end-users to view and analyze enterprise data based on its multidimensional nature and has become an important tool for business management. Although an OLAP solution may bring strategic advantage, its adoption is complex given that building and maintaining data warehouses is not an easy-to-perform task. We propose an alternative approach to perform OLAP by using production rules to compute data cubes. The discussed method incrementally builds the multidimensional database by confronting rules with events sent by operational information systems. The solution described in this paper is currently integrated to a claim processing system in a government health care insurance service, from which we describe a case study. Our method has some advantages in comparison to traditional OLAP systems, such as seamless distribution of processing over time and simpler rules-oriented language for cube generation.

1. Introduction

The term OLAP (Online Analytical Processing) was coined in a 1993 white paper [1], which defines requirements of systems where users can perform multidimensional data analysis. An OLAP application transforms data, retrieved from a data warehouse, into strategic information, enabling users to perform ad-hoc analysis through a view of aggregate data. This type of analysis has become one of the important requirements for managing enterprises [1].

It is importance to notice that performing this kind of analysis directly on operational databases may not be feasible, due to the fact that data may be persisted on logically separated instances. To provide OLAP features properly, it becomes necessary to build and maintain a data warehouse, where data from operational databases will be cleaned, transformed, loaded and summarized. This is a hard, error-prone process where mistakes can create irrecoverable problems.

This paper proposes an alternative approach to represent multidimensional queries with **production rules**, avoiding the creation data warehouses, at the same time simplifying query specification. The goal is achieved by confronting data received from enterprise information system with rules, which define filters and a set of actions to compute **business measures** and persist them over a timeline.

This new approach for performing OLAP brings the following contributions:

- Simplifies query specification by replacing the dimensional-based SQL with simpler production rules;
- Avoids the creation and maintenance of complex data warehouses;
- Seamless distribution of load processing over time;
- A run-time engine and plugin architecture to enable live monitoring of events from many distributed enterprise systems at the same time;

The rest of the paper is organized as follows: next section covers related works in production rules and applications in OLAP systems. The following section presents foundations on the proposed approach, and describes the cycle of performing queries over data warehouses. It also explains production rules as knowledge representation. In Section 4, our proposal is presented, including the inversion of OLAP and the run-time engine architecture. Section 5 show a case study, and Section 6 brings final remarks and future work considerations.

2. Related Work

As mentioned in the previous section, the original paper on OLAP [1] defines it as a category of database processing to solve the most notably lacking feature of relational databases, which is the ability to consolidate (synthesizing pieces of information into single blocks of knowledges), view and analyze data according to multiple dimensions at any given point in time. Since then, many works have been made to support and enhance this approach.

Gray et al. [2] present the problems of the standard SQL *group by* operator and defines the data cube operator, which generalizes histogram, cross-tabulation, roll-up, drill down, and sub-total constructs. An N-dimensional cube is a set of points, which are the aggregate of a particular set of attribute values. And given that computing multidimensional aggregates is the performance bottleneck for OLAP, Agarwal et al. [3] proposed faster algorithms for computing sets of *group bys*, specially the cube operator. Extending sort-based and hash-based grouping methods with optimizations like using pre-computed group by for computing others group-by.

Harinarayan et al. [4] investigated the issue of which cells from the data cube to materialize when working with large data cubes. They used a lattice framework to express dependencies among views, and a greedy algorithm to determine a good set of these views to be materialized, based on the tradeoffs between the space used and the average time to answer a query on a proposed benchmarking database.

The standard method for optimizing OLAP queries execution is often precomputing some of the queries into subcubes, and then to build indexes on theses summary tables. Grupta et al. [5] were pioneers in proposing automation of the selection of summary tables and its indexes with near-optimal algorithms. Going further, and using a logical reconstruction of multidimensional schema design, multidimensional forms were proposed by [6] to ensure summarizability within the whole application schema, achieving sparsity reduction of the underlying data cube and reasoning about the quality of conceptual data warehouse schema.

The application of production rules in OLAP systems have not been deeply explored, with only a few recent papers being published. Vasilecas et al. [7] proposes a backward chaining approach to transform production rules into executable MDX instructions [1], representing rules in XML format and automating decisions and decision support according to the internal and external influences. Prat et al. [8] argue that multidimensional models poorly represent aggregation knowledge, and propose production rules to better represent this knowledge of how aggregation may be performed on a given data cube based on the additive nature (non-, semi-, fully-additive) of the attributes involved, and minimizing the risk of introducing errors during aggregation, since this errors may accumulate consequently leading into awful analysis.

3. Foundations

The proposed method relies on using production rules to replace query languages, anticipating the summarization of data into a dimensional structure, providing a subset of all OLAP features. OLAP was created as a database processing solution to solve issues not tackled by relational database systems. Production rules are a way to represent knowledge about reasoning on data. In this section, we present a general introduction to these two foundational concepts, before discussing our solution.

3.1. OLAP

OLAP applications help analysts and executives to gain insight into the performance of an enterprise through fast access to a wide variety of data, organized to reflect the multidimensional nature of the information [9]. However, this type of analysis can not be done directly from operational databases [1], given that they are prepared only for on-line transactional process. Performing OLAP operations requires detaching data from database, transforming, integrating and loading it into a multidimensional one.

Multidimensional databases are used in data warehousing to support OLAP operations and separate structural aspects and contents. Relational database technology is better suited for transaction management and ad hoc querying [10]. Data warehouses maintain data from operational databases, transformed and integrated accordingly to the demands of analysts. The data is usually organized into a relational model of multidimensional data, called a star schema [11], shown in Figure 1, where tables are separated into dimension tables, which contains identifying information about the dimensions themselves, and a fact table, that correlates dimension and information of interest.

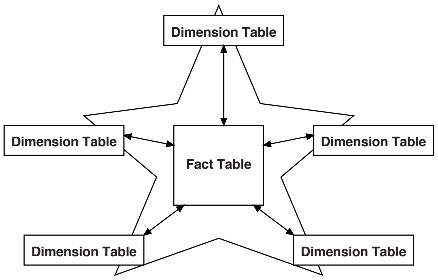


Figure 1. In the star schema dimension tables contains identifying information about the dimensions itself and the fact table correlates them and informations of interest to analysts

The conversion of data from a relational database into a multidimensional schema is not an easy task, and the procedure normally has to be assisted by an expert in business modeling. Some organizations opt for data marts instead,

[1] MDX is a standard query language for OLAP.

i.d. subsets of cubes focused on selected objects, if a complete business model is not fully developed [11].

Queries over the warehouse are made through *XML for Analysis*, a specification for a set of XML message interfaces that defines data access interaction between a client application and an analytical data provider. XMLA provides access to multi-dimensional data from varied data sources through web services that are supported by multiple vendors.

3.2. Production rule engines

A production rule engine is a forward-chaining based system which uses production rules to perform reasoning [12]. A production rule is a list of structured statements comprising a set of conditions (Left Hand Side or LHS) and a correspondent set of actions/consequences (Right Hand Side or RHS).

A production system has a centralized knowledge base known as *Production Memory*, which manage all available rules, an ongoing memory of assertions called *Working Memory*, which contains available facts, and an *Inference Engine*, which is based on a pattern matcher responsible for comparing the current state of the Working Memory with the Rules from the Production Memory, as shown in Figure 2. The forward-chaining logic goes as follows:

1. Find which rules have a LHS that satisfies the current working memory;
2. Among the found rules, choose which of them should get a chance to execute (priority policy);
3. Perform the RHS of all the selected rules.

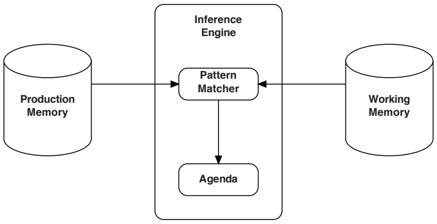


Figure 2. A production rule system architecture, composed by a production memory to manage rules, a working memory to manage facts, and a inference engine responsible for matching rules with facts and executing the consequences.

The application of production rules into information systems have many advantages over traditional approaches. By using production rules, it is possible to separate business logic from conventional boilerplate code used to implement it, maintaining all business knowledge centralized. All decisions made by production rules are readable by a specialist, due to the fact that rules are expressed in a declarative manner, usually even a domain specific one. Modern rule engines are backed by robust algorithms, like RETE [13], that make them faster and scalable than ad-hoc solutions.

However, maintaining a knowledge base involves some issues. Firstly, to ensure reliability, it is important that these systems are checked and validated before being implanted. This process can include checks for anomalies like redundant, contradictory or missing rules that appear after refactoring or because of a communication mismatch between a business expert and a knowledge engineer as explained in [14].

Some efforts on tools developing for verification and validation of knowledge bases are Drools Verifier [14] and VALENS [15]. Drools Verifier check for redundancy, subsumption and equivalence. VALENS focuses on logical verification of knowledge bases using meta-rules, a inference engine and meta information provided by users.

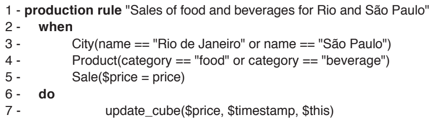
In the next section, we explain our approach of using a production rule engine to provide for the replacement of a relational to dimensional database conversion in OLAP systems.

4. OLAP cube construction using production rules

As cited in previous sections, traditional OLAP solutions involve managing complex data warehouses, where the transformation of data into a dimensional model is difficult and error-prone. In fact, the cost of maintain a multidimensional database can be higher that of maintaining a relational database.

Our proposal is to build each histogram (normally computed by a query over a cube) incrementally through application of production rules on live *events* (data received from an enterprise system, associated with the exact time it was created). Each production rule has a set of conditions that define a filter over data and a set of actions that compute the associated magnitude and persist them associated to a timestamp. With this approach, production rules provide for a robust way of building OLAP cubes, reducing the possibility of errors. Furthermore, building OLAP cubes incrementally divides the processing cost along time. Additionally, storing all measures into relational databases simplifies and reduces the cost of maintaining the multidimensional schema.

To understand the role of production rules in our proposal, the following example should be considered: in a sales management system, all products are grouped by categories such as food, personal care, beverages, etc., and all sales of these products have a location and a price associated. In this context, a manager wants to know the sum of all sales of food and beverage in the cities of Rio de Janeiro and Sao Paulo, grouped by date. With a traditional approach, this query must be performed over an OLAP cube where the dimensions are category, city and date. In our system, this whole operation can be replaced by the production rule shown in Figure 3.



```
1 - production rule "Sales of food and beverages for Rio and São Paulo"
2 -     when
3 -             City(name == "Rio de Janeiro" or name == "São Paulo")
4 -             Product(category == "food" or category == "beverage")
5 -             Sale($price = price)
6 -     do
7 -                     update_cube($price, $timestamp, $this)
```

Figure 3. A simple production rule for a sales management system

In this figure, the rule are expressed in pseudocode. Line one identify the rule to represent the cube where all measures are stored. Lines two to five define the set of conditions (LHS of rule) that filter only desired data to be included into the particular cube. Specifically, line three filters data allowing only two cities (Rio de Janeiro and Sao Paulo), also defining city as a dimension. Line four filters only products from categories food and beverage, also defining the category dimension. Line five is used to temporarily store the price of the sale into a variable ($price) that will be used to increment data into the cube. Lines six and seven define the set of actions (RHS of rule) where the measures are updated using auxiliary functions (these functions are generalized and can be used for any domain). In line seven, the sum of prices is updated using a function called update_cube, a timestamp (the $timestamp variable) and a reference to current rule (the $this variable).

The timestamp is used to explicitly associate the computed measure with the time dimension, and the two other dimensions, city and category, are defined implicitly through the set of conditions.

Considering that this rule has filtered some data from sales registries, computed measures are persisted as shown in table 1.

In table 1, each rule represents a cube, the timestamp represents the time dimension and the measure column represents all business measures computed by the production rules. In the sales management example, the *measure* column contains the sum of prices of all sales of food and beverage sold in the cities of Rio de Janeiro and Sao Paulo in a specific date.

In Figure 4 a generic representation of the OLAP cube generated for the sales management example is shown. In this figure, all sales of food and beverage are presented as one single measure because of the manner in which the rule's LHS was written, applying a disjunction on the product category: measures will be updated when category is ”food” or ”beverage”. It was applied in the same manner to cities of Rio de Janeiro and Sao Paulo, according to the rule's LHS shown in line three of Figure 3. Timestamp and measures are stored together as shown in 1.

Id	Rule	Measure	Timestamp
1	sales_example	10	2012-06-10
2	other_rule_01	20	2012-06-11
3	sales_example	37	2012-06-12
4	other_rule_02	25	2012-06-13

Table 1. Each rule identifies a particular cube. The timestamp represents the time dimension and the other ones are inferred from the LHS of the rule

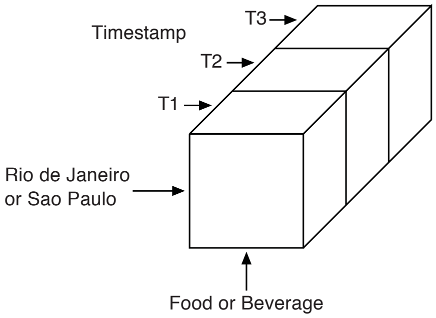


Figure 4. Generic representation of the OLAP cube generated for the sales management example

4.1. Run-time engine architecture

To support our approach of dynamically building OLAP cubes by applying production rules to enterprise data, we developed a runtime architecture that is comprised of five modules: connector, adapter, engine, knowledge base and database.

The *connector* is integrated into the enterprise system, and is responsible for collecting relevant data as events, that will be passed to the *adapter*. The later, for instance, splits all events into smaller unities, called facts, to be confronted with the rules stored in the *knowledge base*. The *engine* itself is responsible for matching the production rules with the facts generated by the *adapter*. All measures computed

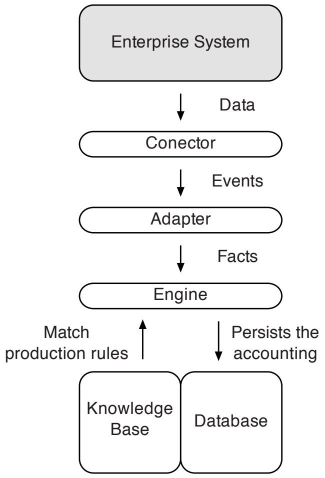


Figure 5. Run-time Engine

by the *production rules* are stored in a relational *database* through auxiliary functions provided by a simple API.

Dynamically, all these five components work together as a consumer of data streams from an enterprise system. When a new event arrives in the stream, it is splited in facts to be confronted with the production rules from the knowledge base, producing measures that will compose the OLAP cube. This process is shown in Figure 6.

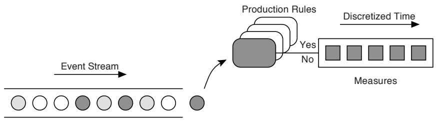


Figure 6. Stream to business measures

In the next section we describe how this system was integrated into a large governmental health care management system.

5. Case Study

The proposed architecture is part of a real claim processing system of a government health insurance service in Brazil. This system is currently being used to smartly monitor health service spending for some medical specialities, for instance cardiological procedures on people older than 50 years old. With this kind of data, it is possible to efficiently manage financial resources and create prevention campaigns.

This goal is implemented by creating rules that filter only the specific medical events, monitoring, accounting for and storing the associated appointments. For instance, a possible query to be executed in this scenario could be: ”How much did cost the cardiology exams made in july for persons above 50 years old?”. In our system, the whole conversion into a cube, and the query over a traditional OLAP system can be replaced by the simpler production rule shown in Figure 7.

Id	Rule	Count Measure	Timestamp
1	Cardiology cost age $>$ 50	10	2012-06-31
2	Cardiology cost age $>$ 50	20	2012-07-01
4	Other_Rule	71	2012-07-02
5	Other_Rule	12	2012-07-02

Table 2. Table that illustrates how our system integrates all matched occurrences of the LRS of a rule over time, using day as the smaller unit of time. All events that match a given rule are summed into the count measure.

```
production rule "CARDIOLOGY exams' costs for USER with age > 50"
    when
        User(age > 50)
        Speciality(name = "cardiology", isExam = true)
        Claim($cost = totalCost, $date = date)
    do
        update_cube($cost, $date, $this)
```

Figure 7. Production rule that filters only the events of interest and integrates them over time.

As this rule is applied to an event stream, the business measures are persisted into a database as shown in Table 2, characterizing a cube with the well-defined time dimension in column ”timestamp” and aditional dimensions (User and Speciality) implicitly defined trough the rule’s LHS. With this data, the query from example can be executed by filtering only the desired date interval.

6. Conclusions

OLAP applications allow the execution of complex queries over summarized data, also know as measures, which aids decision-making for managers. Normally, the creation and maintenance of a data warehouse separated from normal system database is required, where the data is persisted in a multidimensional schema.

However, maintaining a multidimensional database is a complex task and costs more than traditional relational solutions. This paper proposes a model where the measures are accounted and organized in dimensions by production rules, where the rule's LHS filters the information that is used to integrate the measures, and the RHS persists the measures into a relational database discretized over the time dimension.

The most important advantages of the proposed approach are the declarative construction of OLAP queries, the load distribution over time and the relative lower costs when confronted with the traditional method. However, our system is still not able to compute all OLAP operations such as roll-up and drill down in a aggregate set of cubes.

It is important to notice that in our system, a single production rule applied over time results in the same data as an OLAP query that requires a full-stack multi-dimensional conversion before being performed. However, to extend the set of supported OLAP features over the resulting datamart, we plan to keep the computed data in a multidimensional schema.

We also plan to augment the system with reasoning about the time-series computed in our engine. We are interested in prediction algorithms over these time-series. All the proposed features are part of a roadmap for the information system on which we integrated the engine, and we will submit further developments to the academic community.

References

[1] E. F. Codd, S. B. Codd, and C. T. Salley, "Providing olap to user-analysts: An it mandate," *Ann Arbor Michigan*, 1993.

[2] J. Gray, A. Bosworth, A. Layman, D. Reichart, and H. Pirahesh, "Data cube: A relational aggregation operator generalizing group-by, cross-tab, and subtotals," 1996, pp. 152–159.

[3] S. Agarwal, R. Agrawal, P. Deshpande, A. Gupta, J. F. Naughton, R. Ramakrishnan, and S. Sarawagi, "On the computation of multidimensional aggregates," in *VLDB 96, Proceedings of 22th International Conference on Very Large Data Bases, September 3-6, 1996, Mumbai (Bombay), India*, T. M. Vijayaraman, A. P. Buchmann, C. Mohan, and N. L. Sarda, Eds. Morgan Kaufmann, 1996, pp. 506–521.

[4] V. Harinarayan, A. Rajaraman, and J. D. Ullman, "Implementing data cubes efficiently," *SIGMOD Rec.*, vol. 25, pp. 205–216, June 1996.

[5] H. Gupta, V. Harinarayan, A. Rajaraman, and J. D. Ullman, "Index selection for olap," in *ICDE '97: Proceedings of the Thirteenth International Conference on Data Engineering*. Washington, DC, USA: IEEE Computer Society, 1997, pp. 208–219.

[6] W. Lehner, J. Albrecht, and H. Wedekind, "Normal forms for multidimensional databases," *Proceedings. Tenth International Conference on Scientific and Statistical Database Management (Cat. No.98TB100243)*, pp. 63–72, 1998.

[7] O. Vasilecas and A. Smaizys, "Business rule based data analysis for decision support and automation," 2006.

[8] N. Prat, I. Comyn-Wattiau, and J. Akoka, "Representation of aggregation knowledge in olap systems," in *ECIS*, 2010.

[9] G. Colliat, "Olap, relational, and multidimensional database systems," *SIGMOD Rec.*, vol. 25, pp. 64–69, September 1996.

[10] M. Gyssens and L. V. S. Lakshmanan, "A foundation for multi-dimensional databases," in *VLDB '97: Proceedings of the 23rd International Conference on Very Large Data Bases*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1997, pp. 106–115.

[11] S. Chaudhuri and U. Dayal, "An overview of data warehousing and OLAP technology," *ACM Sigmod record*, vol. 26, no. 1, pp. 65–74, 1997.

[12] R. Brachman and H. Levesque, *Knowledge Representation and Reasoning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2004.

[13] C. L. Forgy, "Expert systems," P. G. Raeth, Ed. Los Alamitos, CA, USA: IEEE Computer Society Press, 1990, ch. Rete: a fast algorithm for the many pattern/many object pattern match problem, pp. 324–341.

[14] A. Giurca, G. Nalepa, and G. Wagner, Eds., *Proceedings of the 3rd East European Workshop on Rule-Based Applications (RuleApps 2009) Cottbus, Germany, September 21, 2009*, ser. CEUR Workshop Proceedings. CEUR-WS.org, 2009.

[15] R. Gerrits and S. Spreeuwenberg, "Valens: A knowledge based tool to validate and verify an aion knowledge base." in *ECAI*, W. Horn, Ed. IOS Press, 2000, pp. 731–738.

A Tool for Visualization of a Knowledge Model

Simon Suigen Guo, Christine W. Chan, Qing Zhou
Energy Informatics Laboratory, Faculty of Engineering and Applied Science
University of Regina, Regina, Saskatchewan, Canada S4S 0A

Abstract - A survey of existing ontology visualization tools has revealed that existing tools are inadequate as they lack support for (1) a knowledge modeling methodology, (2) dynamic knowledge representation and visualization, and (3) visualizing a large amount of information due to limitations of the 2D graphical space. This paper presents the design and implementation of an ontology visualization tool called Onto3DViz with the objective of addressing these three weaknesses. Onto3DViz supports visualization of both static and dynamic knowledge models that have been developed based on the Inferential Modeling Technique. By employing 3D computer graphics, Onto3DViz presents a solution for organizing and manipulating complex and related domain concepts and relationships among concepts in a 3D model. Onto3DViz accepts knowledge sources from both XML and OWL files, hence, it is designed to not only visualize an application ontology developed based on the Inferential Modeling Technique, but also supports visualizing application ontologies that have been developed using other tools or techniques. The application of Onto3DViz for visualization of knowledge models in the carbon dioxide capture process was also illustrated in this paper.

Keywords- Knowledge Engineering; Ontology Visualization; 3D graphic; CO_2 Capture Process System

1. INTRODUTION

Ontological Engineering aims to model concepts, axioms and facts which reflect a real world phenomenon of interest. In the modeling process, ontological engineering software tools are often used for creating and editing application ontologies. However, tools that can simultaneous support an ontological engineering methodology and dynamic knowledge modeling are lacking. A recent exception is the dynamic knowledge modeler of Dyna [1], which is based on the Inferential Modeling Technique (IMT) [2] and can be applied for modeling dynamic knowledge. The knowledge model generated with Dyan can be stored in XML based languages such as RDF[1] and OWL[2]. By using XML[3] based languages, Dyna enables knowledge re-use and sharing on the Semantic Web [3]. While Dyna can simultaneously support the IMT and dynamic knowledge modeling, its capability for supporting ontology visualization is limited. In the area of ontology visualization, most major ontology visualization tools use 2-dimentional or 2D graphics for representing concepts and relationships among concepts. 2D graphics are often employed for representing static knowledge in hierarchical or frame like structures. However, 2D graphics are not adequate for representing complex and related information because there is insufficient space on a bi-dimensional plane, where complex and related visual objects tend to squeeze together or even overlap with each other. In an effort to accommodate a model of complex related objects in a 2D plane, existing visualization tools either collapse lower priority visual objects into high level objects, or implement different representations that offer multi-perspectives on an organization of knowledge elements. While these approaches solve the problem of related objects being overcrowded in a shared space, it sacrifices clarity of the display. Moreover, the ontology of many industrial applications often requires visualization of not only static but also dynamic knowledge models. Adding dynamic knowledge on top of an overcrowded 2D graphics model of static knowledge would make the display confusing for the user. Hence, we aim for 3D visualization of knowledge models that consist of both static and dynamic knowledge.

The objective of this paper is to present a tool for ontology 3D visualization called Onto3DViz, 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 IMT, hence Onto3DViz simultaneously supports a developed knowledge engineering method and ontology visualization. Compared to a 2D graphical representation, 3D graphics supports visualization solutions that represent complex and related

[1] Resource Description Framework (RDF), http://www.w3.org/RDF/

[2] OWL Web Ontology Language, http://www.w3.org/TR/owl-features/

[3] Extensible Markup Language, http://www.w3.org/XML/

information in a clear visual model that can be manipulated for multiple views.

This paper is organized as follows. Section 2 presents relevant background literature about the field of ontology editors, ontology visualization tools and the structure of 3D graphic rendering. Section 3 describes the development of the proposed ontology 3D visualization tool. Section 4 presents the application of Onto3DViz on ontology visualization of the CO_2 capture process system, and section 5 discusses some strengths and weaknesses of the tool for ontology visualizing application. Section 6 provides a conclusion and some discussion of future work.

II. BACKGROUND LITERATURE

A. Inferential Modeling Technique (IMT)

The IMT supports "an iterative-refinement of knowledge elements in a problem-domain that provides top-down guidance on the knowledge types required for problem solving" [2]. Typically, the resulting inferential model consists of the four levels of knowledge: domain knowledge, inference knowledge, task knowledge and strategy knowledge. The domain knowledge, also referred as static knowledge, consists of concepts, attributes, values and their relationships; the task knowledge, also referred as 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 [2].

B. Ontological Engineering Modeling Tools

There are various modeling tools available that support ontological engineering. Some examples include KAON [4], OntoStudio [5], Protégé [6] and Dyna [1]. KAON is an open source ontology management infrastructure, which supports creation, storage, retrieval and maintenance of an application ontology. KANO is based on the RDFS file format. The successor of KANO, called KANO2, supports OWL and F-Logic. OntoStudio is a commercial ontology modeling tool, which supports the creation and maintenance of application ontologies. OntoStudio supports a variety of file formats, such as F-Logic, OWL, RDF(S) and OXML (OntoStudio's own XML format). Protégé is another open source ontology editor and knowledge-based framework, which supports frame based (F-Logic) and OWL based ontology modeling. Protégé ontologies can be exported into a variety of formats including OWL, RDF(S), and XML schema. Protégé has been designed so that more features can be added to it by means of implemented plug-ins. For example, Dyna has been implemented as a Protégé-OWL editor plug-in that addresses the need for dynamic knowledge modeling. Dyna supports ontologies represented in the XML or OWL languages format.

C. Ontology Visualization Tools

There are many ontology visualization tools. Although Protégé has some degree of visualization capabilities, the actual ontology visualization applications are implemented in its plug-ins. Some examples of these ontology visualization plug-ins include Jambalaya [7] and OntoSphere [8].

Jambalaya provides several viewing perspectives for the ontology model and supports operations such as filtering and searching, so that the user can examine and interact with the knowledge elements at different levels of abstraction and details. However, Jambalaya only visualizes the static knowledge of classes and instances of an application ontology, it does not support dynamic knowledge visualization. Furthermore, since Jambalaya is based on 2D graphics, the space it supports is insufficient for rendering complex knowledge. In its representation, text labels and symbols tend to overlap when the domain ontology includes a hierarchy of many levels of concepts. This deficiency means it is difficult for users to view and understand the concepts and the relationships among concepts when the domain ontology is complex.

OntoSphere is the only existing ontology visualization tool that adopts the 3D view, thereby extending the volume of space available for visualizing overcrowded concepts. 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. For this purpose, OntoSphere provides four scenes so that the user can observe a visualized application ontology from multiple perspectives. However, OnotoShere had not been developed based on any ontological engineering methodology, and it does not support visualization of dynamic knowledge. Although the employment of 3D graphics enlarges the space available for OntoShpere in rendering images, the problem of overlapping concepts and labels can still occur when the application ontology to be visualized is complex.

III. DESIGN AND IMPLEMENTATION OF ONTO3DVIZ

A. Overview

Onto3DViz has been developed for visualizing an application ontology in 3D graphics. It is written in Java™ language and its 3D visualization engine is implemented in java 3D™. The main difference between Onto3DViz and other ontology visualization tools is that Onto3DViz is a visualization tool developed based on the IMT [2]. Hence it supports visualization of both static and dynamic knowledge models. Onto3DViz also supports knowledge sharing and re-use, by requiring ontology documents represented in Web Ontology Language (OWL) as the input. Unlike Dyna which has been implemented as a Protégé plug-in, Onto3DViz is developed as an independent software application. Onto3DViz is designed to not only visualize an application ontology developed based on the IMT, but also supports visualizing application ontologies that has been developed by other tools or techniques, as long as the ontology is stored in the OWL format.

The design of Onto3DViz is shown in Figure 1, it consists of the three major components of (1) Graphical User Interfaces (GUI), (2) ontology document processor and (3) 3D graphic rendering engine. Each of the three components contains sub packages. The GUI package is used to create windows for displaying the output model generated by Onto3DViz, it also displays text and responses to user inputs. The GUI package consists with three sub components: which are (1) the User Input Handler component, (2) the Text Output component and (3) the Graphic Output component.

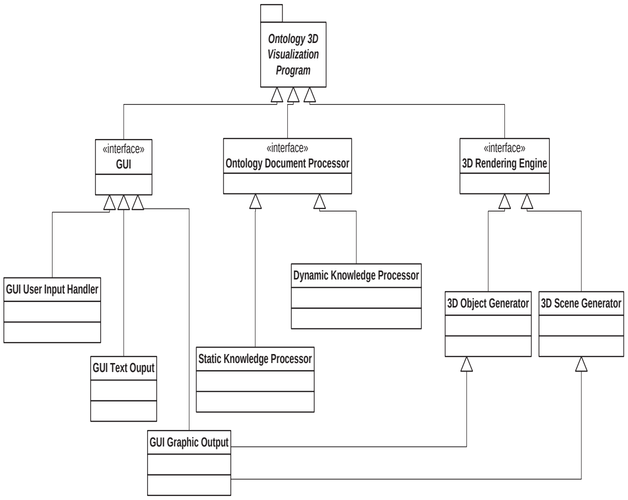


Figure 1. Structural diagram of Onto3DViz

B. Knowledge Extraction and Interpolation

After an application ontology has been constructed using a tool like Dyna [1], both the static and dynamic knowledge elements are stored in the XML files, which can be shared and re-used by other systems. Instead of being connected to an ontology editor that generates an application ontology, Onto3DViz is a separate stand-alone system that uses OWL and XML documents as inputs for 3D ontology visualization. The benefit of this approach is that Onto3Dviz is not restricted by other systems. It can produce 3D ontology visualization as long as it receives a copy of ontology document from another system or via the network. Since the formats of OWL and XML are well standardized and recognizable, the only requirement for using Onto3DViz is that a valid ontology document is available to the system in either format.

C. Visual Object Creation and Display

The objective of Onto3DViz is to visually represent knowledge elements specified in the IMT. The tool employs three techniques for accomplishing this objective and these techniques enable Onto3DViz to represent an ontology with minimal literal description while enhancing visual effects of the representation. Two techniques are discussed in detail as follows; they are (1) representation of concepts by the shape of visual objects, (2) size scaling of visual objects. Onto3DViz uses the following symbols to represent the following concepts:

- Class : Sphere ●
- Objective : Cylinder ⛁
- Instance : Box ▮
- Task : Cone ▲
- Relationship between concepts : Line ——

For the purpose of showing hierarchical structure among concepts of an ontology, Onto3DViz decreases the size of the visual objects that occupy the lower levels in the hierarchy. For example, if the class of *Thing* is the parent class of all classes, then it is scaled to be the largest sized object. A subclass of the *Thing* class is class, the visual object of this class is scaled to be 70 percent of the size of the *Thing* class. Similarly, subsequent children classes are scaled to be 70 percent of the associated parent class. Hence the classes in the default hierarchical view of an application ontology represents the visual objects that are decreasing in size from top to bottom. By employing this technique, the hierarchical relationships as reflected in a parent-and-child relationship are clearly represented among concepts.

D. Hierarchical Visual Object Rendering Algorithm

To represent the hierarchical graph structure among concepts of an ontology in Onto3DViz, a layered rendering algorithm has been implemented in three steps:

1) Pre-process information for rendering: for rendering relevant information on each visual object, e.g. total number of subclasses associated with the object, are

collected in preparation for the drawing algorithm in the next.

2) Recursive drawing algorithm: for each v isual object. According to th e information that is collected from the previous step, the drawing algorithm assigns coordinate for each visual object so as to optimize the graphical space. In such a way that the children classes are placed un der their parent classes with suitable length of distance.

3) Post-process rendering information: t he visual representation of the visualized ontology is further optimized so as to minimize the visual nodes crossing each other, separate nodes colliding together, and eli minate redundant visual objects.

E. User Interaction

Interactions between the 3D model generated by Onto3DViz and t he user are effected using the physical devices of computer mouse and keyboard. Both the mouse and keyboard can be u sed for controlling operation of the 3D model in Onto3DViz. By combining these user control actions, users can manipulate the 3D ontology model and obtain multiple perspectives of the 3D model of an application ontology.

IV. APPLICATION CASE STUDY

In order to demonstrate functionalities of Onto3DViz, the tool was applied for visualization of CO_2 capture process system ontology model. The ontology of carbon dioxide capture process system was developed and implemented on Protégé and D yna at th e Energy Informatics Laboratory at the University of Regina, Canada. The knowledge modeling process was originally conducted based on the IMT, and th e knowledge model consists of both static a nd dynamic knowledge. The static kn owledge includes the information on constructive components of the reaction instruments, fluids, and th e 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 th e control strategies for dealing with 25 critical process parameters when a fault condition emerges. Onto3DViz was applied for visualizing this ontology so as to test and verify the visualization capability of Onto3DViz in a co mplex industrial domain. An overview hierarchical representation of the concepts in the CO_2 capture ontology is shown in Figure 2.

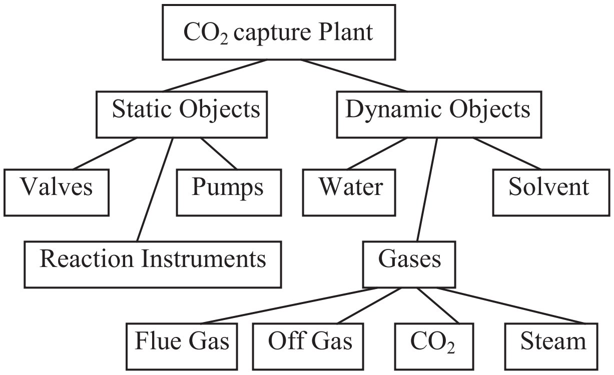


Figure 2. Concepts in CO_2 capture plant [9]

The ontology of the CO_2 capture process generated by Protégé and Dyna is stored into two files, an OWL and an XML file. After loading these files into Onto3DViz, a 3D ontology visualization is generated in Onto3DViz, as shown in Figure 3. Figure 3 shows a co mplete visualized model of carbon dioxide capture ontology, which consists of both static and dynamic knowledge. Static knowledge is represented by the process param eters shown in the foreground. The spherical objects represent the class hierarchy of the knowledge structure. Dynamic knowledge is displayed in the background and is rep resented by cylinders and cones. To further explore the carbon dioxide capture ontology in this model, a u ser can choose to f ilter out some context.

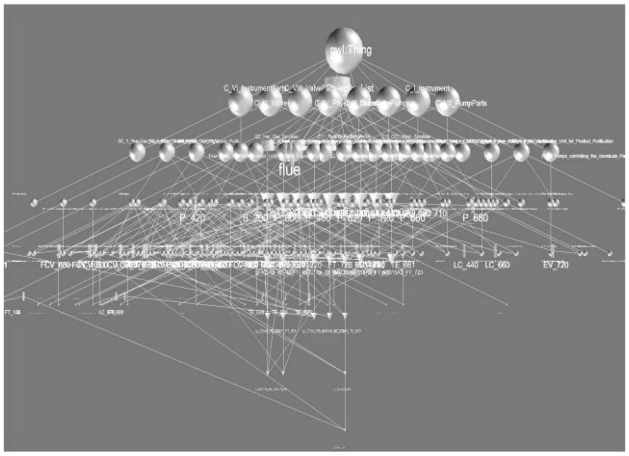


Figure 3. 3D Visualized model of CO_2 capture ontology

By filtering out the dynamic knowledge, a visualized static knowledge model of carbon dioxide capture ontology is shown in Figure 4. Classes are sh own as spheres in the foreground, and insta nces are shown as box es in the background. To obtain a clearer view, the user may need to perform the actions of zooming, rotation, and translation. By filtering out the static knowledge, a visualized dynamic knowledge model of carbon diox ide capture ontology is shown at Figure 5. Objectives are shown as cylinders under the node called ObjectiveList. Tasks are represented as cones and placed under the objectives they are associated with. The relationships among objectives and tasks are linked by lines. The tasks are sorted by the task priority, which means a task located at a higher position has a higher priority than the task or tasks located at the lower positions. Moreover, the priority of a task is correlated to the size of the represented node. A task that has a higher priority is larger than the task that has a low priority.

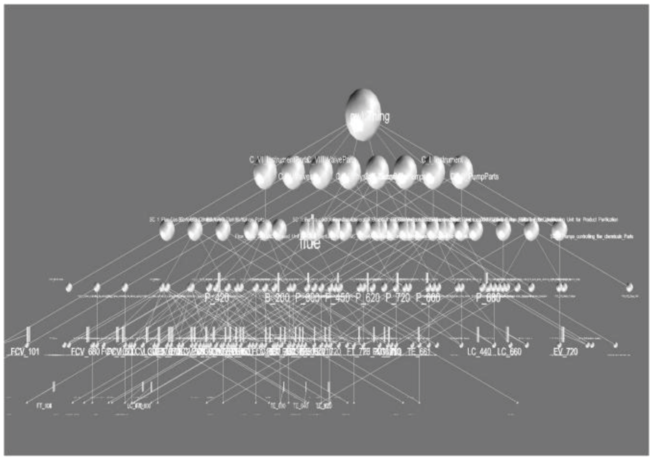


Figure 4. Visualized static knowledge model of CO_2 capture ontology

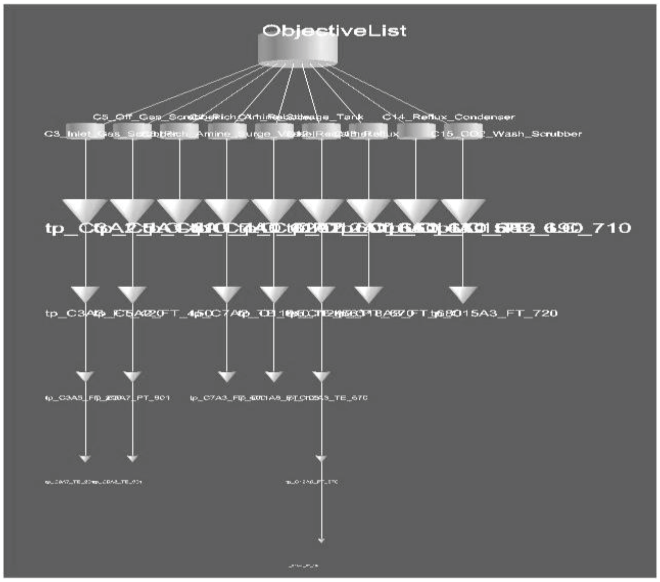


Figure 5. Visualized dynamic knowledge model of CO_2 capture ontology

V. DISCUSSION

A survey of Onto3DViz has been conducted with three users, who completed a questionnaire with 20 questions about the tool's support for knowledge engineering, user experience, and user recommendations. The results suggested that this tool with the developed visualized conceptual models is helpful for enhancing understanding of domain knowledge in the process of knowledge acquisition.

Some strengths of Onto3DViz are summarized as follows. Onto3DViz supports the IMT during the knowledge acquisition process and enables visualizing complex domain knowledge in a 3D model. The visualized model of the domain knowledge supports design of the KBS because the visualized knowledge model can help the knowledge engineer gain quicker understanding of the concepts and relationships among the concepts. It supports communication between the domain expert and knowledge engineer so that it is not limited to oral discussion and textual recording. Hence, it can be used for bridging the knowledge gap between the domain expert and knowledge engineer in the knowledge acquisition process. Moreover, the preliminary design of the knowledge model can be visualized in Onto3DViz. Then, the domain expert and knowledge engineer can discuss and fine tune the visualized model to become the final design.

Several weaknesses of the current version of Onto3DViz are also noted. First, the visualized concepts are not easy to identify in the 3D model, especially when the knowledge model is complex. The control actions of rotating, zooming, and translating of the 3D model requires some learning time. Secondly, the visual objects and labels are too close together in some places, reducing the expressiveness of the model. Furthermore, the nodes and lines overlap at some spots in the 3D visualized model; this problem needs to be addressed in the next version of Onto3DViz. Thirdly, Onto3DViz lacks a search function for concepts.

VI. CONCLUSIONS AND FUTURE WORK

The survey of existing ontology visualization tools revealed that they are inadequate in dynamic knowledge visualization and visualization of a large amount of information due to limitations of the 2D graphical space. Onto3DViz is a new ontology visualization tool that is designed to address these weaknesses. As Onto3DViz is developed according to the knowledge engineering technique of the IMT methodology, it supports both static and dynamic knowledge visualization. By allocating visual objects to 3 different planes in a 3D space, Onto3DViz can support visualization of a large amount of information.

Onto3DViz can represent a complex knowledge model by rendering the concepts using objects of different shapes and placing them at designated 3D positions. Visualization of the CO_2 capture ontology demonstrates Onto3DViz's capabilities in generating a 3D model of static knowledge and dynamic knowledge. We believe that

Onto3DViz can provide strong support for ontological engineers in visualizing application ontologies.

To address the weaknesses noted in the current version of Onto3DViz, some improvements to the tool can be made in the future. Firstly, the current graphical layout of the Onto3DViz is a tree lik e structure generated from top to bottom. It is dif ficult to exten d the rendering algorithm because it i s tightly coupled. In the future, Onto3DViz can be im plemented in a loosely coupled architecture, which enables various graphic layouts to be ported into Onto3DViz. Secondly, to address the difficulty of finding concepts in a model, a f eature that supports automatic identification of a required concept can be added to Onto3DViz. Such a search fu nction can be ad ded by providing a graphical area for the user to in put the exact text name of the concept. After th e name is accepted by Onto3DViz, it will look through the model and locate th e concept, then highlight the 3D co ordinate of the searched concept in the visualized model by adding color and lighting. These improvements will be le ft for future research.

ACKNOWLEDGEMENT

We are grateful for the generous support from the Natural Sciences and Engineering Research Council (NSERC) and the Canada Research Chair Program.

REFERENCES

[1] R. Harrison and C. W. C han. “A dynamic knowledge modeler”, Artificial Intelligence for Engineering Design, Analysis and Manufacturing , Publisher, Volume 23 , Issue 1, pp 53-69, April 2009.

[2] C. W. Chan. (2004, Dec). “From Knowledge Modeling to Ontology Construction”, Int. Journal of Software Engineering and Knowledge Engineering, 14(6), pp. 603-24.

[3] T. Berners-Lee, J. Hendler and O. L assila."The Semantic Web", Scientific American. May 2001.

[4] K. Bauknecht, A. M.Tjoa and Gerald Quirchmayr. E-Commerce and Web Technologies, Third International Conference, EC-Web 2002, Aix-en-Provence, France, September 2-6, 2002, Proceedings, Volume: 2455, Series: LNCS, Seiten: 304-313, Verlag: Springer. 2002.

[5] Onto-Studio. (n.d.). ontoprise GmbH. Retrieved from: http://www.ontoprise.de/en/home/products/ontostudio

[6] Protégé. (n.d.). Stanford Center for Biomedical Informatics Research. Retrieved from: http://protege.stanford.edu

[7] Jambalaya. (n.d.). Computer Human Interaction & Software Engineering Lab (CHISEL) and its members. Retrieved from: http://www.thechiselgroup.com/jambalaya

[8] Alessio Bosca, Dario Bonino, and P aolo Pellegrino. (2005) OntoSphere: more than a 3D ontology visualization tool, SWAP.

[9] Zhou Q, Ch an CW, Tontiwachwuthikul P, Idem R, Gelowitz D, A statistical analysis of the carbon dioxide capture process. Greenhouse Gas Control, 3, 535–544 (2009).

Rendering UML Activity Diagrams as a Domain Specific Language— ADL

Charoensak Narkngam
Department of Computer Engineering
Chulalongkorn University
Bangkok 10330, Thailand
Charoensak.N@student.chula.ac.th

Yachai Limpiyakorn
Department of Computer Engineering
Chulalongkorn University
Bangkok 10330, Thailand
Yachai.L@chula.ac.th

Abstract—**Activity diagrams are extensively used in software engineering for modeling the behaviors of systems. However, for large and complex systems, manually creating activity diagrams with graphic notation is error-prone and may cause data and behavior inconsistency. This paper thus proposes a preventive approach to generating proper activity diagrams with a textual notation that can be considered as a domain specific language, called action description language (ADL). The design of ADL contains both grammars and lexical analyzer that can explain nodes and coordinate behaviors in activity diagrams. Parsing an ADL script results in a semantic model that can be eventually transformed to the target activity diagram, using the methodology described in this research. An example is used to demonstrate how to transform the ADL script to the semantic model of the target activity diagram containing decision, nested decision, and concurrent controls, without loss of information.**

Keywords-process modeling; unified modeling language; activity diagram; domain specific language

I. INTRODUCTION

Activity diagrams are extensively used in various domains for modeling behaviors of systems. The diagram shows the workflow from a s tart point to the finish point detailing the many decision paths that exist in the progression of events contained in the activity. They may be used to detail situations where parallel processing may occur in the execution of some activities. Although activity diagrams are useful for describing procedural logic, business processes, and workflows, it is difficult to manually create the activity diagrams of large and complex systems that accurately describe the behaviors of th e systems, and conform to UML specification.

Several research and so ftware tools have attempted to create activity diagrams using textual notation languages. Flater et al. [1] proposed the activity diagram linear form (ADLF) which is a plain text representation to support the rendering of activity diagrams as h uman-readable text. PlantUML [2], and yUML [3] are so ftware tools that facilitate creating UML diagrams with the simple and i ntuitive language. However, they still lack the features of verification and validation to ensure consistency and understanding of th e activity diagrams generated. This paper thus presents an approach to rendering UML activity diagrams as a domain specific language, called action description language (ADL). ADL is designed to explain relations between objects such as seq uence, parallel, and alternative flows. An example is provided to demonstrate how to transform the semantic model of ADL to th at of the target activity diagram. The resulting semantic model will then be used for visualizing the target activity diagram without loss of information, and the data can be exported for the verification and validation process.

II. BACKGROUND

A. UML Activity Diagram

In UML [4], an activity diagram is used to display a sequence of activities. The diagram describes the flow of operations and data represented by a set of nodes and edges. A node denotes an action, an object, or a control, while edges represent either an object flow or a control flow. Fig. 1 s hows the architecture or metamodel of UML activ ity diagrams 2.3 proposed in this work.

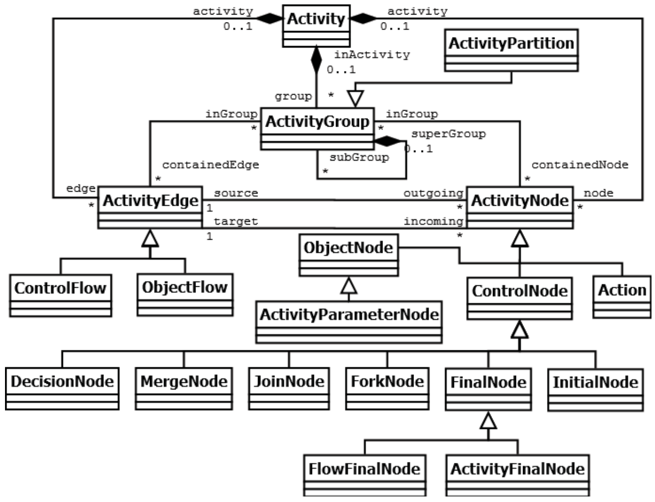


Figure 1. Activity diagram metamodel

B. Domain Specific Languages

A domain specific language (DSL) is a formal specification language which contains the syntax and semantics that model a concept at the same level of abstraction provided by problem domain [5]. Four elements are required to constitute a D SL: structure, constraints, representation, and behavior, where structure defines the abstract syntax; constraints describe additional restrictions on the structure of the language; representation can be the description of a graphical or textual concrete language syntax; and behavior can be a mapping or transformation into another language [6,7].

III. Action Description Language

Fig. 2 illustrates the ADL metamodel considered as the source model that corresponds to the target model or the activity diagram metamodel presented in Fig. 1. Meta Object Facility [8] is used to develop ADL on the basis of the objects or tokens residing the activity diagram. Rather than focusing on actions that do not enable the detection of objects between two actions, this research opts to focus on objects as the key concept to link between two actions. In other words, an action is regarded as the relation of two objects. Therefore, an action must have at least one input object and one output object.

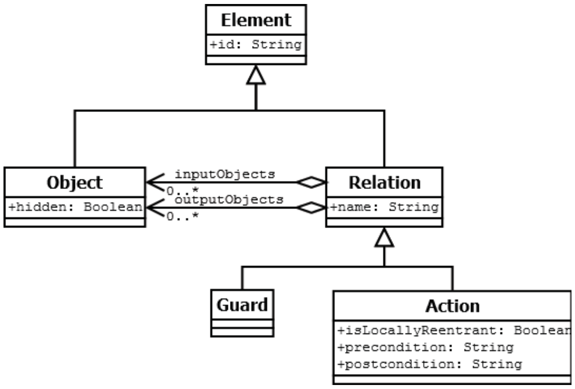


Figure 2. ADL metamodel

Since the intent of ADL is to explain relations between objects, or actions, the textual notation language is developed in terms of actions and relations of actions. The language contains the keywords:

```
action, end, decision, if, then, else, endif,
initiate, break, terminate
```

And the textual syntax grammar built on EBNF [9] is defined as following:

```
<diagram> ::= 'diagram' <name> (<action> |
    <sequence> | <decision>)* 'end'
<action> ::= 'action' <action_identifier>
    <action_attrs>* 'end'
<action_attrs> ::= 'name' <name>
    | '<-' <object_identifier> (','
        <object_identifier>)*
    | '->' <object_identifier> (','
        <object_identifier>)*
    | 'precondition' <condition>
    | 'postcondition' <condition>
<sequence> ::= <action_identifier> '->'
    <action_identifiers> ['[' <label> ']'] ('->'
    <action_identifiers> ['[' <label> ']'])*
<decision> ::= 'decision' 'from' <action_identifier>
    <guard> <guard>* 'end'
<guard> ::= 'if' <expression> 'then' <statement>
    (<statement>)* ('else' <statement>
    (<statement>)*)* 'endif'
<statement> ::= <action_identifiers> | <guard>
<action_identifiers> ::= <action_identifier> (','
    <action_identifier>)*
```

A. Defining Input and Output of An Action

Since the approach uses an object to create a connection between two actions, the grammar of an action is defined as follows:

```
action a
    <- O1 /* an input object O1 */
    -> O2 /* an output object O2 */
end
```

Hence, an action can be described as a path in directed graph, as expressed in (1).

$$a = O1 \rightarrow a \rightarrow O2 \quad (1)$$

B. Defining Sequence of Actions

A sequence of actions can be defined by two methods: 1) use explicit objects to create a connection; and 2) use implicit objects to create a connection.

The former method uses the input and output objects explicitly defined in an action to create a connection between two actions. For example,

```
action a
    <- O1
    -> O2
end
action b
    <- O2
    -> O3
end
```

Hence, a sequence of actions can be derived as stated in (2):

$$a = O1 \rightarrow a \rightarrow O2$$
$$b = O2 \rightarrow b \rightarrow O3$$

then $$a \,.\, b = O1 \rightarrow a \rightarrow O2 \rightarrow b \rightarrow O3 \quad (2)$$

The latter method provides the alternative grammar to reduce the complexity of ADL scripts by defining a sequence of actions directly without specifying the binding object, or the implicit object is used to create a connection. For example,

```
01 action a
02     <- O1
03 end
04 action b
05     -> O3
06 end
07 a -> b
```

The line 07 defines a sequence of actions with the implicit object that is created and bound to the output of action `a` and the input of action `b`. The result is tantamount to (2).

C. Defining Concurrent of Actions

Since the ADL grammar cannot directly describe parallel behaviors, process algebra [10] is used to replace them with sequential behaviors as expressed in (3).

$$(a \,.\, b) \parallel (a \,.\, c) = a \,.\, (b \parallel c) \quad (3)$$

Given a script containing concurrent of actions as follows:

```
action a
    <- O1
    -> O2
end
action b
    <- O2
    -> O3
end
action c
    <- O2
    -> O4
end
```

According to (1), actions can be defined as follows:

$$a = O1 \rightarrow a \rightarrow O2$$
$$b = O2 \rightarrow b \rightarrow O3$$
$$c = O2 \rightarrow c \rightarrow O4$$

Next, the sequences of actions can be derived as:

$$a\,.\,b = O1 \rightarrow a \rightarrow O2 \rightarrow b \rightarrow O3$$
$$a\,.\,c = O1 \rightarrow a \rightarrow O2 \rightarrow c \rightarrow O4$$

According to (3), the parallel behaviors with object relations can be expressed as (4).

$$a\,.\,(b \parallel c) = O1 \rightarrow a \rightarrow O2 \rightarrow (b \rightarrow O3 \parallel c \rightarrow O4) \qquad (4)$$

D. Defining Condition

Activity diagrams use a decision node and guard conditions to denote alternative composition or choices. That is, when a decision node accepts a token, it will ev aluate if th e token satisfies which guard condition annotated in square brackets on branches, and th en deliver the token to th e outgoing edge satisfying the condition.

Example activity diagram containing a decision and the associated ADL script are illustrated in Fig. 3.

```
action receiveOrder end
action fillOrder end
action closeOrder end

receiveOrder -> fillOrder ['order accepted']
receiveOrder -> closeOrder ['order rejected']
```

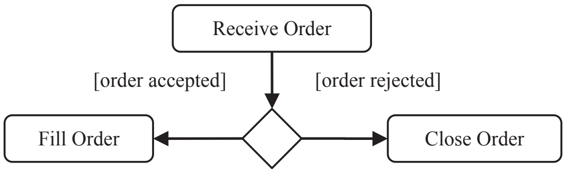


Figure 3. Example for defining condition

E. Defining Nested Condition

For complex decisions or nested decisions, additional information of guard conditions will be added to the sequence of actions. A guard condition also owns input and output objects to create a co nnection between actions related to it. Given the following chunk of ADL script,

```
action a end
action b end
decision from a
  if 'guard1'
  then
     if 'guard2'
     then b
     endif
   endif
end
```

the sequence of actions a . b can be expressed as (5).

Suppose
$$a = O1 \rightarrow a \rightarrow O2$$
$$b = O2 \rightarrow b \rightarrow O3$$

then
$$a\,.\,guard1\ \&\ guard2{:}\rightarrow b \;= O1 \rightarrow a \rightarrow O \rightarrow G1 \rightarrow G2 \rightarrow b \rightarrow O3 \qquad (5)$$

where G1, G2 denote output objects produced from `'guard1'` and `'guard2'`, respectively. The values of G1 and G2 a re O2 attached with the evaluated condition.

IV. Generating Activity Diagram From ADL

Parsing ADL scripts results in ADL semantic models, which cannot be directly transformed to activity diagrams because they still lack the information of controls, namely fork, join, decision, and merge. However, the particular pattern exists for each co ntrol and can be detected from the details of edges (incoming/ outgoing), labeled edges, and flow detail contained in ADL scripts. Steps to generate activity diagrams from ADL scripts are illustrated in Fig. 4.

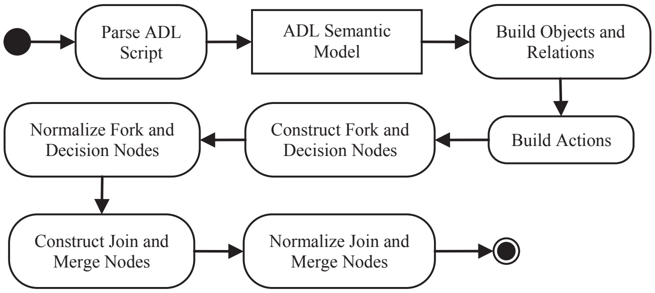

Parse ADL Script
ADL Semantic Model
Build Objects and Relations
Build Actions
Construct Fork and Decision Nodes
Normalize Fork and Decision Nodes
Construct Join and Merge Nodes
Normalize Join and Merge Nodes

Figure 4. Steps to generate activity diagrams from ADL scripts

A constructor or a m odel is required for retriev ing loss information of controls and generating the activity diagram. To accomplish model-to-model transformation, QVT or query/ view/ transformation specification [11] is applied to describe the transformation from the semantic model of ADL metamodel to the semantic model of the metamodel for constructing activity diagrams, as sh own in Fig. 5. Starting from defining top-level relations of model transformation that specify the mappings between the two models as follows,

```
transformation adl (adl : ADL, constructor :
ADConstructor) {
  top relation ObjectToObjectNode {...}
  top relation RelationToEdge {...}
  top relation ActionToActivityNode {...}
  relation ActionToAction {...}
  relation ActionToControlNode {...}
}
```

the semantic model for constructing activity diagrams is then created. Next, it will be transformed to the semantic model of the metamodel for constructing activity diagrams using the mappings of top-level relations defined as follows:

```
transformation ad (constructor : ADConstructor, ad :
AD) {
  top relation ObjectNodeToObjectNode {
    checkOnly domain constructor left:ObjectNode {
      hidden=false, name=on}
    enforce domain ad right:ObjectNode {name=on}
    ...
  }
  top relation ActionToAction {...}
  top relation EdgeToActivityEdge {...}
  relation ActivityEdgeToControlFlow {...}
  relation ActivityEdgeToObjectFlow {...}
  top relation InitialNodeToInitialNode {...}
  top relation FlowFinalNodeToFlowFinalNode {...}
  top relation ActivityFinalNodeToActivityFinalNode
    {...}
  top relation ActivityNodeToEdgeDetail {...}
  relation OutgoingEdgeToForkAndDecisionNode {...}
  relation IncomingEdgeToJoinAndMergeNode {...}
}
```

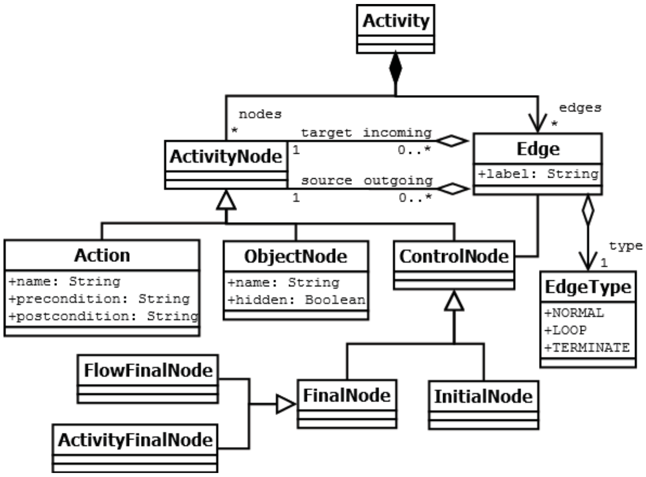


Figure 5. Metamodel for contructing activity diagrams

V. Example of Research Method

An example of the trouble ticket scenario is selected to demonstrate how to compose the underlying ADL script and how to insert control nodes based on the ADL semantic model. Fig.6 illustrates the target activity diagram [2] associated with the example scenario.

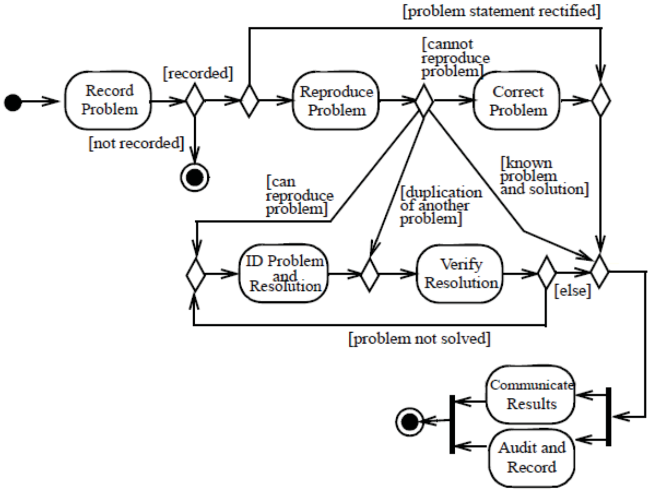


Figure 6. Target activity diagram of trouble ticket scenario [2]

The ADL script representing the trouble ticket scenario can be written as follows:

```
diagram 'ticket trouble scenario'
  action recordProblem end
  action reproduceProblem end
  action correctProblem end
  action idProblemAndResolution end
  action verifyResolution end
  action auditAndRecord end
  action communicateResult end

  decision from recordProblem
    if 'recorded'
    then
    reproduceProblem
      if 'problem statement rectified'
      then auditAndRecord,
           communicateResult
      endif
    else
      if 'not recorded' then terminate endif
    endif
  end

  decision from reproduceProblem
    if 'cannot reproduce problem'
    then correctProblem
    else
      if 'can reproduce problem'
      then idProblemAndResolution
      endif
    else
      if 'duplication of another problem'
      then verifyResolution
      endif
    else
      if 'known problem and solution'
      then auditAndRecord,
           communicateResult
      endif
    endif
  end

  correctProblem -> auditAndRecord
  correctProblem -> communicateResult

  decision from verifyResolution
    if 'problem not solved'
    then idProblemAndResolution
    else auditAndRecord,
         communicateResult
    endif
  end
end
```

Fig. 7 illustrates all the actions and their object evidences as implicit objects are used to create all action sequences here.

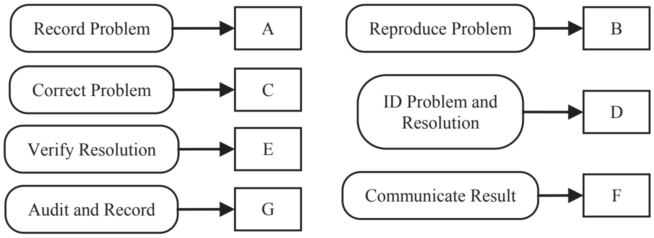


Figure 7. Actions and their object evidences

Once all sequences of actions have been created, the start point and the termination point are identified. Actions without an input object and those without an output object are candidates for connecting to the initial node and the activity final node, respectively. Therefore, recordProblem is connected to the initial node, while auditAndRecord and communicateResult are connected to the activity final node. The directed connections can be expressed as follows:

initiate → INITIAL → recordProblem → A
auditAndRecord → G → terminate → TERMINATE
communicateResult → F → terminate → TERMINATE

Next, (5) is applied to build guard condition objects, resulting in the directed connections as follows:

recordProblem → A → reproduceProblem => recordProblem → A → [recorded] → reproduceProblem; recordProblem → A → auditAndRecord => recordProblem → A → [recorded] → [program statement rectified] → auditAndRecord; recordProblem → A → communicateResult => recordProblem → A → [recorded] → [program statement rectified] → communicateResult; recordProblem → A → terminate => recordProblem → A → [not recorded] → terminate.

A digraph [12] is deployed to visualize all the objects and action relations constructed, as shown in Fig. 8.

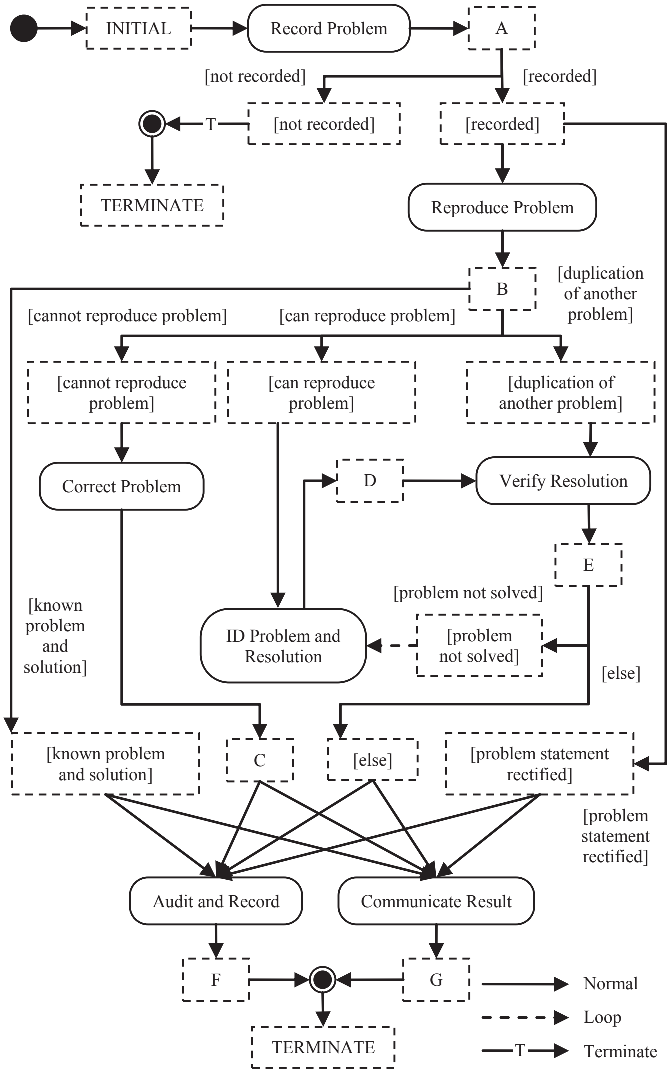


Figure 8. Digraph of objects and action relations

When an edge has the ancestor action node as the target of the edge, it is considered as a loop-type edge. Example loop-type edge existing in Fig. 8 is the edge, which has [problem not solved] as source and idProblemAndResolution as target. It is observed that idProblemAndResolution is simultaneously the ancestor node and the target node on the looping path: idProblemAndResolution → D → verify Resolution → E → [problem not solved] → idProblemAndResolution → ...

Next, the controls will be created by detecting the patterns as described in [13], whether they appear in the digraph. For example, if a node has more than one outgoing edges, this will signify the insertion of a fork, a decision, or both a fork and a decision node. A fork node will be inserted to the node which has all outgoing edges that are normal-type without label, while a decision node will be inserted to the node which has all outgoing edges that are normal-type with label. In Fig. 8, a decision node will be inserted to object A, B, and E. Based on Fig. 8, the insertion of a decision node to object [recorded], as shown in Fig. 9, results from applying the following process algebra (6) to reduce the complexity.

Refer to [10] $$a . b + a. c = a . (b + c) \qquad (6)$$

then [recorded] . reproduceProblem
+ [recorded] . [problem statement rectified] =
[recorded] . (reproduceProblem+[problem statement rectified])

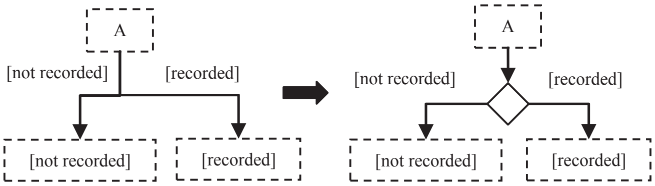


Figure 9. Example of mapping decision pattern

Considering auditAndRecord and communicateResult, they have quite complex incoming edges providing tokens from four nodes. The result of using the simple fork pattern is illustrated in Fig. 10a. Process algebra is further applied to reduce the complexity or the number of edges. That is, from Fig. 8, let s1 be C, s2 be [known problem and solution], s3 be [else], s4 be [problem statement rectified], t1 be auditAndRecord, and t2 be communicateResult, then s1 . (t1 || t2) + s2 . (t1 || t2) + s3 . (t1 || t2) + s4 . (t1 || t2) = (s1 + s2 + s3 + s4) . (t1 || t2). The result is shown in Fig. 10b.

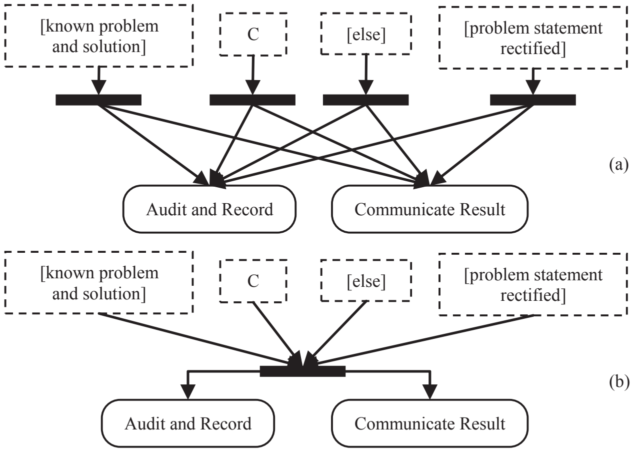

[known problem and solution]
C
[else]
[problem statement rectified]
(a)
Audit and Record
Communicate Result
[known problem and solution]
C
[else]
[problem statement rectified]
(b)
Audit and Record
Communicate Result

Figure 10. Example of mapping fork pattern (a) without combined fork nodes (b) with combined fork nodes

Once all fork and decision nodes have been created, the details of incoming edges and flow types can be used for the insertion of a join or merge node. The algorithm of tree traversal is used to determine the type and level of flows where the root node is INITIAL object node. Abstract State Machines or ASM [14] is used to describe the algorithm as follows:

```
TraverseTree(node, flow) =
  let f = new(Flow) in
    info:= flow.info
  if IsJoinNode(node) or IsMergeNode(node) then
    f.info:= CalculateFlow(node)
  else
    if IsInitialNode(node) then
      f.info:= f.info + "N"
    if IsForkNode(node) then
      f.info:= f.info + "F"
    if IsDecisionNode(node) then
      f.info:= f.info + "D"

  let c = true

  if count(node.incomingEdges) > 1
    and IsJoinNode(node) = false
    and IsMergeNode(node) = false then
      forall e in node.incomingEdges do
        if IsNull(e.flow) then c:= false
      if c = true then
        TraverseTree(CreateJoinMerge(node), f)
  else
    forall e in node.outgoingEdges do
      e.flow:= f
      TraverseTree(e.target,f)
```

To reduce complexity when inserting join and merge nodes, it is suggested that the flows with the deepest level and the same type should be determined first. Fig. 11 shows the result of the insertion of two merge nodes into Fig. 10b.

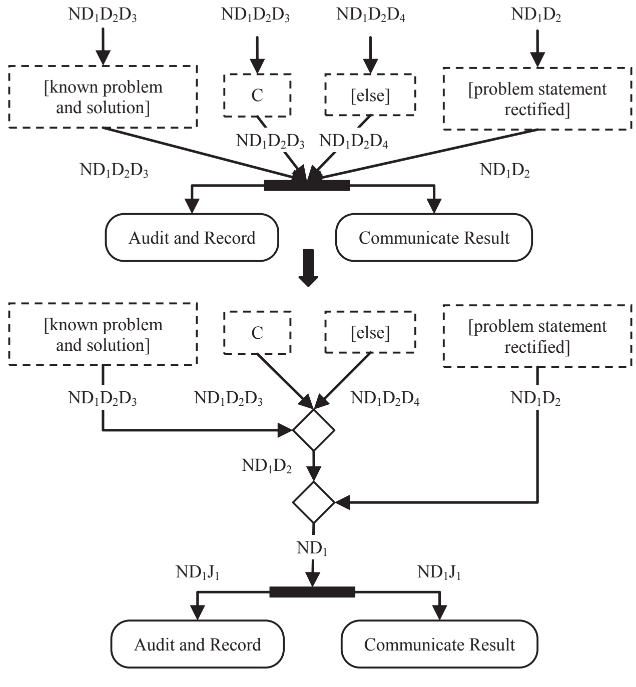


Figure 11. Example of mapping merge pattern

VI. CONCLUSION

The design of ADL, a domain specific language for UML activity diagrams presented in this paper, covers four elements required to constitute a DSL: structure, constraints, representation, and behavior. The ADL metamodel illustrates the language structure. Constraints can be defined as validation and verification rules described in [13], serving the purposes of preventing data inconsistency, and fortifying conformance to UML specification, respectively. Representation can be visualized with a digraph. Behavior is accomplished by means of QVT applied for model-to-model transformation.

Process algebra and metamodel-based technology are applied to develop the language, which is unambiguous and has a sufficiently high-level abstraction to describe general activity diagrams. Compared to previous research and existing tools mentioned earlier, the proposed approach could reduce the complexity of scripts, and source lines of code. Compared to manual method, the proposed method could reduce resource consumption, in addition to prevent modeling mistakes and incorrect notation usages.

The trouble ticket scenario is selected as an example to demonstrate how to transform the underlying ADL script to the semantic model with control nodes that can be further processed to visualize the target activity diagram, eventually.

The method presented in this paper is capable of generating intermediate activity diagrams. Future research work could be the enhancement of the method to support structural activity diagrams which can describe interruptible regions, events, and features related to the programming languages.

REFERENCES

[1] D. Flater, P. A. Martin, and M. L. Crane, Rendering UML Activity Diagrams as Human-Readable Text, National Institute of Standards and Technology, 2007.

[2] PlantUML, http://plantuml.sourceforge.net, January 2012.

[3] yUML, http://yuml.me, Febuary 2012.

[4] OMG, Unified Modeling Language^{TM} (OMG UML): Superstructure Version 2.3, Object Management Group, Inc., 2010.

[5] D. Ghosh, DSLs in Action, Manning Publications Co., 2011.

[6] A. Prinz, M. Scheidgen, and M. S. Tveit, "A Model-Based Standard for SDL", Springer-Verlag Heidelberg, SDL 2007, LNCS 4745, pp. 1-18, 2007.

[7] T. Gjøsaeter, I. F. Isfeldt, and A. Prinz, "Sudoku – A Language Description Case Study", Springer-Verlag Berlin Heidelberg, SLE 2008, LNCS 5452, pp. 305-321, 2009.

[8] OMG, Meta Object Facility (MOF) Core Specification Version 2.0, Object Management Group, Inc., 2006.

[9] D. Grune and C. Jacobs, Parsing Techniques: A Practical Guide, 2nd ed, Springer, 2008.

[10] J. C. M Baeten, T. Basten, and M. A. Reniers , Process Algebra: Equational Theories of Commnicating Processes, Cambridge University Press, 2010.

[11] OMG, Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification Version 1.1, Object Management Group, Inc., 2011.

[12] J. Bang-Jensen and G. Gutin, Digraphs: Theory, Algorithms and Applications, Springer-Verlag, 2007.

[13] C. Narkngam and Y. Limpiyakorn, "Domain Specific Language for Activity Diagram", in Ramkhamhaeng Journal of Engineering, vol. I, 2012.

[14] E. Börger and R. Stärk, Abstract State Machines: A Method for High-Level System Design and Analysis, Springer-Verlag Heidelberg, 2003.

umlTUowl - A Both Generic and Vendor-Specific Approach for UML to OWL Transformation

Andreas Grünwald

Vienna University of Technology
Karlsplatz 13, 1040 Wien, Austria
Email: a.gruenw@gmail.com

Thomas Moser
Christian Doppler Laboratory CDL-Flex
Vienna University of Technology
Taubstummengasse 11, 1040 Vienna, Austria
Email: thomas.moser@tuwien.ac.at

***Abstract*—The extraction of knowledge from UML class diagrams into ontologies is a typically manual thus time-consuming and error-prone task in software and systems engineering. To support an automated UML to OWL transformation approach, purebred and hybrid tool solutions have been researched and evaluated. Since no approach met the defined requirements, a new UML to OWL tool, called *umlTUowl* was designed and realized. *umlTUowl* supports the transformation of Visual Paradigm, MS Visio and ArgoUML UML class diagrams into valid OWL2 DL and is available as open source software. The tool provides a novel approach, resolving issues of preceding approaches through an extensible architecture dealing with the fragility of XML Metadata Interchange (XMI) by providing traceability and an automated testing framework for vendor-specific UML tools. In addition, this work presents an industrial use case, in which *umlTUowl* is applied to models from the automation systems engineering domain. The tool successfully passed all test cases completely, including the presented industry-specific use case.**

***Index Terms*—Semantic Web; Automated Ontology Creation; Knowledge Modeling; UML to OWL transformation; Industrial Case Study.**

I. Introduction

Industrial projects, such as performed by the Christian Doppler Laboratory ”Software Engineering Integration for Flexible Automation Systems” (CDL-Flex) at Vienna University of Technology (TU Vienna), often require the collaboration of people with different domain expertise and methodological approaches that should provide their heterogeneous knowledge into an overall solution. The CDL-Flex tackles this kind of issues by leveraging OWL ontologies as explicit data model specifications, which enable the intercommunication of heterogeneous tools based on a semantically level within CDL-Flexs integration framework called Engineering Knowledge Base Engineering Knowledge Base (EKB) [12].

However, there is still an important issue to cover, concerning the common lack of experience regarding ontologies, i.e. OWL, by project members. Engineers do not only often have insufficient knowledge about ontologies; furthermore, the maintenance of sophisticated OWL ontologies by itself is challenging, error-prone and may lead to unrecognized consequences during runtime. Common modeling components (e.g. used for software technology, automation systems engineering and industrial automation) are scattered across team roles thus it may be unreasonable, not to say impossible, to expect that team members make a shift from their prefered modeling language towards OWL.

CDL-Flexs project experience shows that most of the engineers are familiar with simple concepts using UML class diagram notation. Hence, UML modeling tools, such as Visual Paradigm (VP)'s UML editor[1] can be established to collect and share domain knowledge between project partners, primarily using UML's logical data model notation. An UML to OWL tool may be leveraged to replace ontology experts who have to transfer this UML diagrams into adequate OWL knowledge bases.

In this work, existing UML to OWL transformation solutions will be evaluated with respect to their practical applicability. After discussing pros and cons of even more persuasive solutions, i.e. Eclipse's (ATLAS Transformation Language (ATL) and TwoUse Toolkit, as well as common issues regarding XMI transformations, the lack and necessity of a flexible, easy-adaptable model to model solution becomes obvious. Thus this work introduces the developed transformation tool *umlTUowl*. The application of the tool, which is capable of transforming VP XMI 2.1, MS Visio XMI 1.0 or ArgoUML XMI 2.1 class diagrams into OWL ontologies, is then illustrated by presenting an industry-specific use case.

The innovation of *umlTUowl* lies in the combination of the benefits of classical meta models with a vendor-specific but flexible implementation approach. While conventional transformation tools either fail by trying to provide a solution that is capable to deal with all XMI standards or simply assume that a model-specific implementation implies that the tool is compatible with all other XMI versions of all different vendors, *umlTUowl* considers this grievances by providing a traceable, testable, integrable, extensible framework approach that might also be of scientific value.

II. Evaluation of Existing Transformation Tools

Because large vendors tend to come up with new ontology modeling solutions currently, existing tools have been grouped into *purebred* UML to OWL transformation and *hybrid* modeling tools during evaluation phase. Latter are often sophisticated graphical tools, which do not support

[1] http://www.visual-paradigm.com

transformation of UML diagrams directly, but offer ontology modeling utilizing UML profiles (compare [4][pg.2ff]). In this alternative approach users have to explicitly specify OWL elements using UML syntax[1], while conventional tools typically try to extract implicit information from UML elements[2]. Only the latter approach is relevant for CDL-Flex, hence only one representative, i. e., *TwoUseToolkit*[2], will be outlined in this paper. Other examples for hybrid tools are *Altova Semantic Works*,*UMLtoOWL* by Gasevic *et al.* which is described in [3] and *VisioOWL*.

Purebred transformation tools have been compared and evaluated based on portability, traceability, their transformation approach, up-to-dateness, availability of documentation, support, usability and surplus value for CDL-Flex. The major prerequisite was that the UML to OWL transformation feature is executable or that, if the tool/feature is not available, it is somehow justified, why the tool is capable to transform UML into OWL.

A. Evaluation Results

None of the tools transformed VP XMI 1.2 or 2.1 into valid OWL 2 DL. All scripts completed within a reasonable time (in other words: none of the programs crashed), but also none of the tools correctly transformed a single UML element into its valid OWL counterpart. The tools either terminated before an OWL file was created or the root element of the resulting OWL XML/RDF file was empty.

Most of the tools passed their own reference model tests, which means that they successfully translated the examples (XMI files) they provided into valid OWL ontologies, which has been validated using OWL Validator by University of Manchester (ManchesterValidator)[3]. Transformation of Leinhos' UML2OWL[4] reference models failed when the models were opened in Poseidon for UML SE 6.0.2 and re-exported into XMI 1.2. *Dia*[5] successfully imported MS Visio XSD files, but crashed during OWL export, because the Visio file could not be saved as a valid Dia file (tested with Ubuntu 11.04). OntoStudio[6] seemed quite interesting and provides a pliable GUI (Eclipse based). However, UML 2 is only available for ObjectLogics. *NeOn Toolkit*[7], which is a derivate of OntoStudio claims to provide transformation functionality as well, but version 2.5.1 misses any feature. Eclipse offers an entire UML modeling environment (EMF; ATL; UML2 or Papyrus project) and ATL transformation performed well for the distributed XMI 2.1 reference examples, but the flip side of the coin is that most of the Eclipse modeling tools are still in incubation. Detailed evaluation results and a comprehensive comparison of available tools in tabular form are provided in [4][p.2ff].

B. ODM-ATL Implementation (Bridging UML - OWL)

ATL[8] is an Eclipse-based model transformation language that provides ways to produce a set of target models out of a set of source models. It originally has been initiated by Object Management Group (OMG) and is part of Eclipse's Model To Model Transformation (M2M) project. One of the transformation use cases for ATL is the Ontology Defnition Meta Model (ODM) implementation for bridging UML and OWL. Hillairet[9] provides the most current project files (including samples) as well as the source files for reverse transformation (OWL2UML) of OWL. Evaluation showed that the tool transforms XMI 2.1 UML models accurate, if they have been designed using Eclipse's UML2 plug-in or Papyrus, which both implement OMG's XMI 2.1 standard. Transformation of VP XMI 2.1 caused exceptions and resulted in an empty OWL file.

Eclipse ATL's transformation approach is based on different meta levels, as defined by OMG (compare [5]). ATL is used to transform UML 2.0 (XMI 2.1) input files into OWL metadata, based on OWL.ecore. Eclipse ecore is a meta language inspired by MOF 1.4 and is used to define platform independent models [6]. Once the XMI input file has been transformed into OWL ecore format, the AM3 plugin serializes OWL resulting in valid OWL RDF/XML output.

The ATL approach is elegant, though transformation rules are defined in a single, transparent file. ATL supports syntax highlighting and debugging. Transformation rules are intuitive to interpret and may be easier to read than XSLT, because the rules do not follow an XML sytanx. Helper functions can be defined and allow more freedom for the developer. ATL is easy adaptable. Drawbacks are that the Eclipse environment is hard to setup. ATL UML2OWL project requires Eclipse Eclipse Modeling Framework (EMF), UML2 and AM3. Some of those components are still in incubation phase and incompatible either among themselves or with some versions of Eclipse. Furthermore, ecore/ATL training effort is high. Developers have to dive into the EMF concept to gain a deeper understanding and learn ATL, which can be time consuming.

C. TwoUse Toolkit

The TwoUse Toolkit originated from an European project. It implements the current OMG and W3C standards for software design, code generation and OWL ontology engineering.[7] All components are implemented as Eclipse plug-ins. The aim of TwoUse toolkit is to enable Semantic Web Software Engineering and Model Driven Semantic Web. Thus, besides extensive browsing and querying support (e.g. SPARQL), it offers different graphically design tools for modeling of OWL ontologies:

- UML editor for modelling and transformation of UML profiles into OWL ontology functional syntax.
- TwoUse Graphical Editor for directly creating ontologies (some design elements have been adopted from Protégé).

[2] http://code.google.com/p/twouse

[3] http://owl.cs.manchester.ac.uk/validator

[4] http://diplom.ooyoo.de/index.php?page=download

[5] http://projects.gnome.org/dia/

[6] http://www.ontoprise.de/de/produkte/ontostudio/

[7] http://neon-toolkit.org/wiki/Main_Page

[8] http://www.eclipse.org/atl

[9] http://perso.univ-lr.fr/ghillair/projects.html

The tool's ability to deal with SWRL rules based on an UML profile based approach as discussed in literature [8] has been successfully demonstrated in [4][p.7,8].

D. A Word about XML Metadata Interchange

Evaluation showed that the transformation proccess heavily depends on the format of the provided XMI file. Although most of the leading UML modeling tool vendors support XMI, their formats are highly incompatible and implementations often have a lack of quality. Often export and import of simple diagrams within the same modeling tool is doomed to fail.

One of the reasons for this is that XMI attempts to solve a technical problem far more difficult than exchanging UML models; it attempts to provide a mechanism for facilitating the exchange of any language defined by the OMG's MetaObject Facility (MOF). Furthermore, the UML 2.* Diagram Interchange specification lacks sufficient detail to facilitate reliable interchange of UML 2.* notations between modeling tools. Since UML is a visual modeling language, this shortcoming is a show-stopper for many modelers who don't want to redraw their diagrams[10].

There are several reasons why transformation of VP XMI 2.1 format into valid OWL fails using existing UML2OWL tools. First of all, VP takes use of different XML namespaces to distinguish between different types of elements (i.e. classes, associations, packages) which do not have been considered by most of the transformation tools. Secondly, the XML structure is nested and data for particular elements have to be grabbed using cross references. This turned out to be fragile according to XSLT scripts implementations and tools who try to tackle a joint solution for all XMI dialects. Furthermore, the structure of the resulting VP XMI file depends on the state of the UML editor view during the export process (e.g. if a particular package/part of the diagram is selected, the XML hierarchy changes). Finally there are even incompatibilities between different versions of VP (i.e. between VP UML 7.2 and VP UML 8.2).

The conclusion is that an UML2OWL transformation process always should be tailored to a specific vendor's UML solution. UML2OWL tool developers are required to specify vendor, product name and version of the UML editor they support as well as XMI versions and instructions, how to export that XMI version from the respective UML editor. Furthermore they must define, which UML elements their tool supports (and which not) and provide adequate reference models, including test cases (automated?) and documentation.

III. umlTUowl Transformation Tool

A. Overview

umlTUowl has been developed to overcome the problems that arise during the transformation process of evaluated UML2OWL tools. It is not only optimized to transform UML class data models, as used by partners of CDL-Flex (*Visual Paradigm for UML V7.2, 8.2; XMI 2.1*, *Microsoft Visio 2010 XMI 1.0*) into OWL 2 DL ontologies, but also attempts to be a best practice lightweight framework for engineers and thus is hosted as an open source project.[11] Both, executable and source code are available.

Supported UML elements	Not supported UML elements
- classes	- multiplicity of attribute value
- abstract classes	- attribute values except primitives
- interfaces	- package elem. inside a diagram
- generalization	- data constraints
- multiple packages (diagrams)	- class operations
- attributes	- association classes
- visibility of attributes (partly)	- n-ary associations
- attrs. with primitive data types	- overlapping/disjoint classes
- attrs. with XML data types	- roles (VP, ArgoUML)
- associations	- XOR annotation
- navigable associations	- redefnition of derived attributes
- multiplicity of associations	- subset annotation
- aggregations	- ordering+uniqueness attr. annot.
- compositions	- datatype meta annotation
- labelled endpoints (MS Visio)	- enumerations
- comments / note elements	- stereotypes

TABLE I: Comparison of supported and not supported UML elements

umlTUowl eases the integration of new transformation scripts, e.g. support for *ArgoUML 0.32.2 XMI 2.1* (freeware) has been already implemented. Its maxim is: don't try to provide an overall transformation solution for all vendors, because parsing of XMI is too fragile. Be prepared for variations and UML-model-vendor updates that affect the structure of the resulting XMI code by providing traceability of supported UML tools, as well as providing high testability, modifiability and extensibility.

Traceability is reached by keeping record of version numbers, export settings and supported UML elements for each vendor-specific UML modeling tool. This approach has several advantages. It provides end user documentation and guarantees designated transformation results, which at all times are OWL2 DL compatible and thus should be also compatible with current versions of Protégé (¿=4.1). Furthermore, by leveraging automated testing those artifacts also ensure that whenever a vendor adapts a modeling tool, these changes will be detected. *umlTUowl* comes with a bulk of unit tests and three implemented reference model tools[4][pg.37ff.], hence providing comprehensive templates to developers, who want to implement transformations for additional models. By utilizing a meta model, either a new input format (e.g. Entity Relationship Diagram (ERD) instead of UML) may be added, or the output converter (e.g. DAML instead of OWL) may be replaced.

Supported UML elements are outlined in Table I. VP uses the term *package* for *diagram* synonymously. *umlTUowl* allows to define, if all packages should be merged into a single ontology, or if an own ontology file should be created for each package. The elements which are not supported by *umlTUowl* typically are not commonly used or only well-understood by UML experts. They have not been implemented

[10] http://www.uml-forum.com/FAQ.htm

[11] http://sourceforge.net/projects/uml2owl

within the *umlTUowl* prototype because these elements have not been used by CDL-Flex customers and engineers in the past years and also may not be important for the populace of UML modelers outside CDL-Flex projects. Other reasons are that some elements are hard to implement because they can't be expressed naturally in OWL. For instance, how can someone express a class operation such as `sum(int a, int b): return int;`, without introducing additional semantics? However, this case is not relevant for the CDL-Flex.

The tool should be seen as a recommendation as well. It attempts to suggest, which informations should be provided when publishing an UML transformation tool, based on the experiences collected during the evaluation phase.

B. *Software Architecture*

umlTUowl naturally accepts UML diagrams in the specified XMI format. As illustrated in figure 1, at first the XML/XMI files are loaded into memory. Depending on the XMI input-format a particular converter is selected, which transforms the UML model into a simplified meta model (`umltometamodel`). The meta model basically consists of modules that represent and facilitate all relevant UML model elements in a convenient way. It has been defined analogue to Eclipse MOF, Netbeans MDR or as discussed in [9]. The usage of a meta model has numerous benefits. On the one hand, decoupling of XMI parsing process (`umltometamodel`) and OWL serialization (`metamodeltoowl`) eases the adoption of additional converters (e.g. *Poseidon* XMI format or even ER-diagram conversion) and enables automated testing approaches. On the other hand, a meta model can facilitate access to specific meta data elements, which implies that searching, manipulating and referencing of entities is much more comfortable than directly accessing XML. This advantage is also leveraged by the `harmonizer` component, which ensures that all classes, attributes and associations are labeled with a unique and OWL 2 DL compatible name, before they are finally serialized into an OWL 2 DL ontology utilizing the OWL API by University of Manchester (OWL API). Decoupling of input (parsing) and output (serialization) process furthermore has the advantage that not only the parser might be replaced, but also the OWL serialization component.

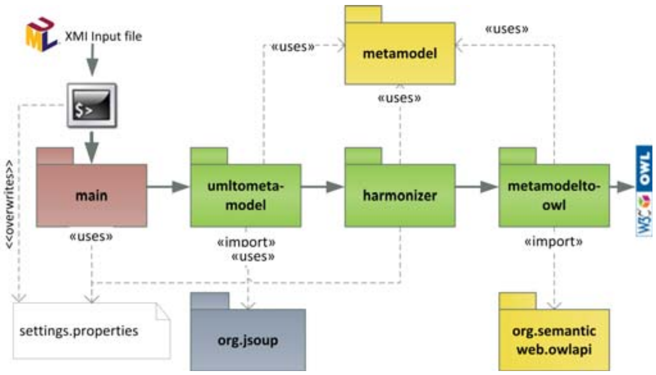


Fig. 1: Workflow and software architecture of umlTUowl

1) Harmonizer: The harmonizer component ensures the uniqueness of element names and prepares them for the OWL export. According to the configured strategy all entity names in the meta model are unified, so that each name only occurs once. Furthermore, attribute and association names are converted into common OWL styles.

Table II illustrates some of the most considerable harmonizing techniques and how they are applied (the complete example can be found in [4][p.22]).Two strategys have been implemented to handle packages: Depending on the configuration either all meta packages are merged into a single meta model during the harmonizing phase, or all packages remain and are serialized into separated ontologies, subsequently. If all classes are merged into a single ontology, more harmonizing effort is required, since it can happen that semantically inequivalent classes with same class name exist in different packages. These classes have to be prefixed with their package name.

Duplicate attributes and association names frequently occur even within a package. They appear, when packages are merged into a single ontology or when the same attribute (or association) name is contained in different UML classes (e.g. age might exist for a class named "Student", but also for a class named "Building"). Therefore, the `harmonizer` component offers a set of naming strategys, which can be customized.

Thus, attribute names are always renamed (pre- or suffixing class and/or package name), if they occur more than once, although different strategys exist therefore. Associations often exist without label. Even if they are named labels are useless, except the association is navigable. Hence, all associations are renamed, depending on the chosen strategy (`settings.properties`). For instance, as illustrated in table II, aggregations and compositions can be renamed using supplementary patterns. If only a conventional association is established between two classes, e.g. "Building" contains "Rooms", this results in two OWL object properties. The first one is named "buildingHasRoomAssociation" and the second (inverse) object property is named "roomHasBuildingAssociation"; if the association is bidirectional navigable, there is no way, to find out, if "Building" is contained in "Room" or vice versa.

2) Metamodeltoowl: The OWL API by the University of Manchester has been utilized to serialize *umlTUowl*'s meta model into valid OWL2 DL. Not all UML elements have a matching counterpart. For instance, an abstract UML class will be transformed into an `owl:Class` entity with probably specific naming conventions. Depending on the chosen transformation strategy either all UML packages are merged into a single file, or each package is separated into a single OWL ontology. This approach is inspired by previous works, such as [2] and [9].

Attributes are transformed into data properties. XML built-in data types (e.g. `anyType`) are supported as well as OWL built-in data types. UML associations result in OWL ob-

[12] X: harmonizing carried out using default settings; blank: adapted configuration settings; S: blank + special case

Elem.	UML Example	D[12]	Harmonizing result	Relevant settings

class	Two classes named *Student* exist within two different packages.	X	Student
			Package2_Student
				merge-packages=true
				merge-disable-fixing=true
				merge-class-prefix={package}_

class	Two classes named *Student* exist within two different packages.		Package1_Student
			Package2_Student
				merge-packages=true
				merge-disable-fixing=false
				merge-class-prefix={package}_

class	Two classes named *Student* exist within two different packages.		Student in out_Package1.owl Student in out_Package2.owl	merge-packages=false
abstract class	{*abstract*} *Student*	X	Abstract_Student	abstract-prefix=Abstract_ abstract-postfix=

attribute	Two classes, namely *Student* and *Building* exist within the same package. Both contain attribute *name*.	X	Building: hasName
			Student: hasPackage1_StudentName
				merge-packages=true
				data-property-prefix=has
				merge-attribute-strategy=duplicates
				merge-attribute-prefix={package-class}_

attribute	Two classes, namely *Student* and *Building* exist within the same package. Both contain attribute *name*.		Building: hasName
			Student: hasStudentName
				merge-packages=true
				data-property-prefix=has
				merge-attribute-strategy=duplicates
				merge-attribute-prefix={class}

attribute	Two classes, namely *Student* and *Building* exist within the same package. Both contain attribute *name*.		Student: hasStudentName
			Building: hasBuildingName
				merge-packages=true
				data-property-prefix=has
				merge-attribute-strategy=all
				merge-attribute-prefix={class}

association	Unidirectional association named *based on* between Tree and Trunk.	X	hasTreeTrunkRelation
			hasTrunkTreeRelation (inverse). Comment "Relation originally named 'based on'" added.
				relation-strategy-relation=Relation
				relation-strategy-1=
				has{from}{to}{dependingRelation}

composition	Each *Tire* belongs to exactly 1 *Car* at one time.	X	hasCarTireComposition
			isTireOfCar (inverse - part of)
				relation-strategy-1=has{from}{to}{dependRel}
				relation-strategy-composition-part-1=is{from}Of{to};
				relation-strategy-composition=Composition

aggregation	*University* has *Researchers*.		hasUniversityResearcher
			IsResearcherOfUniversity (inverse)
				relation-strategy-1=has{from}{to}{dependRel}
				relation-strategy-aggregation-part-1=is{from}Of{to};
				relation-strategy-aggregation=

association	Three associations: *Tank* contains *Temperature Sensors*, *Level Sensors* and *Heaters* (class names). All assoc. are labelled with *contains*.	S	All associations are still labelled with contains. One OWL object property named `contains` with domain `Tank` and range `Temperature_Sensor or Level_Sensor or Heater` will be created.	allow-multiple-labelled-names=true
				relation-strategy-1={name}
				relation-strategy-2=has{from}{to}
				{dependRel}

TABLE II: Examples of implemented harmonizing techniques, depending on settings

ject properties, which are restricted by domain and range through their corresponding OWL classes (formally UML classes or interfaces). Equally named associations as well are supported. Thus, domain/range expressions contain concatenated OWL entities, separated by `OR`. Example: UML associations `Professor holds Lecture` and `Professor holds Seminar` will be transformed into object property `holds` with domain `Professor` and range `Lecture OR Seminar`. If an association is navigable in both directions, object properties will be linked as *Inverse properties* one another. All object properties are added as subclass axioms to their related OWL classes, concerning cardinality. Assuming that a Professor can hold between 0 and 500 lectures, the example above would result in two subclass axioms: `holds min 0 Lectures and max 500 Lectures` and `holds some Seminar`.

The mapping mechanism of *umlTUowl* is detailed in figure 2. All transformed ontologies have been validated manually, by applying ManchesterValidator for OWL2 DL format and by comparing the resulting ontology in Protégé (using the FaCT++ reasoner) [10]). The verification of the ontologies showed that all UML reference models were transformed correctly and completely for all three UML modeling tools and with respect to different configuration settings of *umlTUowl*.

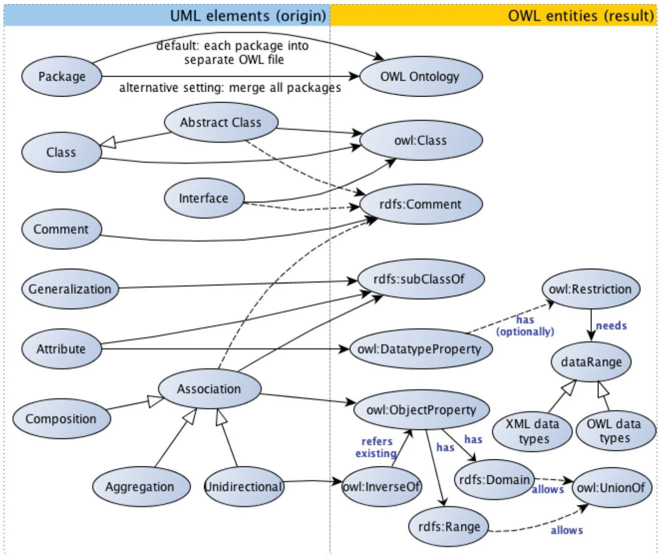


Fig. 2: Mapping of UML elements and OWL entities

Due to *umlTUowl*'s architecture the test cases are transparently available and repeatable.

C. Time Complexity

Time complexity of the transformation implementation is

$$\begin{aligned} T = O(\#N) * [O(\#G) + O(\#Assoc) \\ + O(\#Attr) + O(\#Com) * O(\#ComC)] \end{aligned} \quad (1)$$

where *#N* is the number of XML elements (depending on the XML serialization), *#G* is the number of class generalizations, *#Assoc* the number of associations, *#Attr* the number of attributes, *#Com* the number of comments and *#ComC* the number of connections between classes/associations and comments.

The comparison with Xu et al.'s solution[11] shows that time complexity is slightly higher than *O(#N)*. The main reason is that the DOM structure of VP's XMI 2.1 in some cases has to be traversed, e.g. to map attributes and data range, or to link comments to their corresponding entity. The algorithm may be optimized (e.g. extracting data types into an array before processing attributes), but the current algorithm is broadly satisfactory.

D. Automated Testing

umlTUowl uses an elegant testing approach based on executable test cases (jUnit): The tool is shipped with a variety of test cases, which are linked with elaborate UML reference models to cover common UML class diagram constructs as defined in Table I. The unit tests are applied to particular elements of the reference model's diagrams, whereupon identical diagrams have been created for each of the supported modeling tools. For each converter implementation, an adequate XMI file, images of the model's diagrams created by the specific modeling tool, and the related project files are provided. To implement a new UML modeling editor's XMI format, one just has to extend an existing XMI converter, redraw the reference model in the particular UML modeling tool and document the export process of the newly implemented XMI version. Due to leveraging, the intermediate meta model coding of test cases is no longer necessary! Additional reference models can be placed to support additional features for a specific UML editor's XMI file format. Adaption of not yet implemented transformation features can be carried out across all existing converters, hence the approach helps to gain high-quality converters. Even UML-related models, such as MS Visio ERD, can be implemented in adequate time.

E. Use Case: Transformation of Industrial Tank Model

Practitioners, especially designers and Quality Assurance (QA) personnel, want to make complex industrial automation systems more robust against normally hard to identify runtime failures. Challenges to detect and locate defects at runtime come from the different focus points of models. Without an integrated view on relevant parts of both design-time and runtime models inconsistencies from changes and their impact are harder to evaluate and resolve. Better integrated engineering knowledge can improve the quality of decisions for runtime changes to the system, e.g., better handling severe failures with predictable recovery procedures, lower level of avoidable downtime, and better visibility of risks before damage occurs [12][13][14]. As shown in [15], these problems can be addressed with the help of ontologies and reasoning.

The CDL-Flex developed, domain-specific EKB provides a better integrated view on relevant engineering knowledge in typical design-time and runtime models, which were originally not designed for machine-understandable integration. It contains schemes on all levels and instances, data, and allows reasoning to evaluate rules that involve information from several models. *umlTUowl* in combination with the EKB enables practitioners from different fields to create ontologies out of previous or novel created UML (or similar) models on the fly, and thus helps to gain better integrated and shared knowledge across teams. It not only supports iterative development of ontologies and hence helps to improve consistency and compliance between design- and code/runtime- elements, but also unburdens EKB ontology engineers so that they can focus on modeling of complex logical relationships and axioms.*umlTUowl* incorporates and centers domain experts and helps improving the CDL-Flex EKB workflow transiently.

To demonstrate the integration of *umlTUowl* into CDL-Flex's EKB and to dig into a use case, the tool has been validated against a previously-used, domain specific tank model (compare [15]). Both, VP class data diagram and OWL ontology that had been created in manual work by CDL-Flex, have been provided beforehand. The UML diagram models a piped tank that is connected to several control elements, which ensure that the tank provides enough fluid (eg. water or liquid chemical substances) and the content is heated, dependent on the measures of different sensors. The existing, manually created ontology and the *umlTUowl* transformation result have been compared in Protégé and were validated using ManchesterValidator. The results are outlined next.

1) Class Transformation and Disjointness: The tool transformed all 19 classes and considered all hierarchy levels correctly. For instance, actuator is the superclass of heater, pump and valves, while valve itself has been subdivided into magnetic, manual and pneumatic classes. Unlike the manually created ontology, disjointness of classes is not defined in the *umlTUowl* created ontology. By default, UML classes are defined as overlapping, so the ontology engineer has to explicitly define disjoint classes, using Protégé.

2) Equivalent Classes: Of course the automated *umlTUowl* transformation approach cannot distinguish equivalent and defined classes. While the CDL-Flex ontology engineer defined some equivalent classes in OWL, the tool created defined classes with subclass axioms, instead.

3) Attributes: Because for none of the attributes a data type had been specified in the UML diagram, the tool assigned data range `anyType`. The ontology engineer has to add missing ranges to all data properties manually, e.g. set the data range of `hasCapacity` (Domain = `Tank`) to `float`.

4) Associations: In the first test run, the tool created 24 object properties, while the CDL-Flex ontology contained only seven object properties. The reason was that the transformation tool created an object property and its inverse for each UML association. For instance, one sensor measures an actuator. The resulting object properties were `hasActuatorSensorRelation` and `hasSensorActuatorRelation`, which is the inverse object property (compare Table II). For avoidance of inverse relations, either the object properties have to be removed in Protégé, or the UML designer has to define unidirectional associations (add navigation arrows). After the UML diagram had been reworked, 24/2=12 object properties remained. The CDL-Flex ontology still contained seven object properties. The reason is that in the UML model six associations have been labelled with `contains` and three are named `connectedTo`. *umlTUowl* cannot determine, if all associations with same label are semantically equal and therefore, creates a new object property for each association.

So *umlTUowl* had been extended, to support transformation of equally named associations into a single object property (compare Table II). By adding the lines `allow-multiple-labelled-names=true` and `relation-strategy-1={name}` in `settings.properties` one can define that equally named associations are allowed and that their name are used to label OWL object properties. After re-execution of *umlTUowl* the number of object properties in the result was reduced to a number of five (the manually created ontology contained two additional properties, which were not modeled in the UML file). The result has been equivalent to the manually created ontology. The transformation succeeded for all three UML modeling tools with respect to different configuration settings.

IV. Conclusion

Traditional UML to OWL transformation tools are doomed to fail transforming VP UML2.0 or MS Visio 2010 diagrams into valid OWL, because they heavily rely on XMI standards, which turned out to be fragile and vary by vendors' tools. Eclipse's ATL framework offers a transparent transformation approach, but is, in combination with OWL, still half-baked. *umlTUowl* is not only tailor-made for the CDL-Flex, but provides a framework that supports high extensibility due to precise documentation of supported UML modeling tools and an automated test approach. It rests on a well-thought-out lightweight architecture, which enables developers to add high-quality converter scripts rapidly. Future work may be carried out regarding the implementation of less common UML constructs, such as disjointness, multiplicity of data properties or support of constraints.

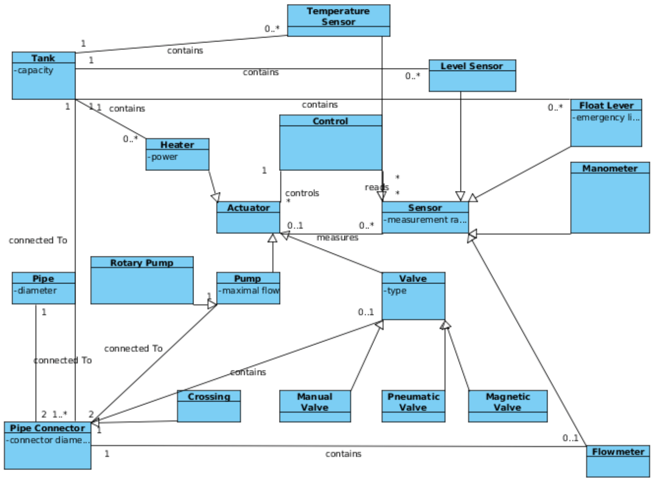


Fig. 3: Tank modeled in VP for UML 8.2

References

[1] K. Baclawski, M. M. Kokar, P. A. Kogut, L. Hart, J. Smith, J. Letkowski, and P. Emery, "Extending the Unified Modeling Language for Ontology Development," *Software and Systems Modeling*, vol. 1, 2, 2002.

[2] K. Falkovych, M. Sabou, and H. Stuckenschmidt, "Uml for the semantic web: Transformation-based approaches," *Knowledge Transformation for the Semantic Web*, vol. 95, pp. 92–107, 2003.

[3] D. Gasevic, D. Djuric, V. Devedzic, and V. Damjanovi, "Converting UML to OWL ontologies," in *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, ser. WWW Alt. '04. New York, NY, USA: ACM, 2004, pp. 488–489.

[4] A. Grünwald, "Bachelor thesis. evaluation of uml to owl approaches and implementation of a transformation tool for visual paradigm and ms visio," 2011. [Online]. Available: http://cdl.ifs.tuwien.ac.at/files/bachelor_thesis_uml2owl.pdf

[5] J. Nytun and A. Prinnz, "Metalevel representation and philosophical ontology," in *Workshop on Philosophy, Ontology, and Information Systems (held as part of the Eighteenth European Conference on Object Oriented Programming, ECOOP-04), Oslo*, 2004, p. 2.

[6] V. Bacvanski and P. Graff, "Mastering eclipse modeling framework," *EclipseCon. Eclipse Foundation*, 2005.

[7] F. Silva Parreiras, S. Staab, and A. Winter, "Twouse: Integrating uml models and owl ontologies," Institut für Informatik, Universität Koblenz-Landau, Tech. Rep. 16/2007, 2007.

[8] S. Brockmans, P. Haase, P. Hitzler, and R. Studer, "A metamodel and uml profile for rule-extended owl dl ontologies," *The Semantic Web: Research and Applications*, pp. 303–316, 2006.

[9] Kiko and Atkinson, "A Detailed Comparison of UML and OWL," *REIHE INFORMATIK TR-2008-004*, 2008.

[10] M. Horridge and S. Bechhofer, "The OWL API: a Java API for working with OWL 2 ontologies," in *Proc. of the 5th Int. Workshop on OWL: Experiences and Directions (OWLED 2009), CEUR Workshop Proceedings, Chantilly, VA, United States, October*. Citeseer.

[11] Z. Xu, Y. Ni, L. Lin, and H. Gu, "A Semantics-Preserving Approach for Extracting OWL Ontologies from UML Class Diagrams," *Database Theory and Application*, pp. 122–136, 2009.

[12] T. Moser and S. Biffl, "Semantic tool interoperability for engineering manufacturing systems," in *Emerging Technologies and Factory Automation (ETFA), 2010 IEEE Conference on*. IEEE, 2010, pp. 1–8.

[13] F. Waltersdorfer, T. Moser, A. Zoitl, and S. Biffl, "Version management and conflict detection across heterogeneous engineering data models," in *Industrial Informatics (INDIN), 2010 8th IEEE International Conference on*. IEEE, 2010, pp. 928–935.

[14] T. Moser, S. Biffl, W. Sunindyo, and D. Winkler, "Integrating production automation expert knowledge across engineering stakeholder domains," in *Complex, Intelligent and Software Intensive Systems (CISIS), 2010 International Conference on*. IEEE, 2010, pp. 352–359.

[15] M. Melik-Merkumians, T. Moser, A. Schatten, A. Zoitl, and S. Biffl, "Knowledgebased runtime failure detection for industrial automation systems," in *Workshop Models@ run. time*, 2010, pp. 108–119.

A Framework for Class Diagram Retrieval Using Genetic Algorithm

Hamza Onoruoiza Salami, Moataz A. Ahmed
Information and Computer Science Department,
King Fahd University of Petroleum and Minerals,
Dhahran, Saudi Arabia
{hosalami, moataz}@kfupm.edu.sa

***Abstract*—In this research, we propose the use of genetic algorithm (GA) for retrieving class diagrams from a software repository. This technique will prove useful in the reuse of existing Unified Modeling Language (UML) artifacts. Our proposed similarity metric for matching query class diagrams to repository class diagrams is based on name (semantic) similarity and structure similarity. Our preliminary results on structure similarity show that the proposed method is effective in retrieving class diagrams from the repository that are similar to query class diagrams.**

Keywords-component; UML Class Diagrams; Software Reuse; Similarity; Genetic Algorithm

I. INTRODUCTION

Software reuse is the process of creating software systems from existing software rather than building software systems from scratch [1]. Some of the advantages of reuse include reduced overall development cost, increased reliability and accelerated development [2]. One of the most important activities in software reuse is retrieval. During retrieval, an input query is compared with existing software artifacts to determine those artifacts that are similar enough to the query. The emphasis on similarity of query artifacts to stored software artifacts is important, to ensure that adapting retrieved artifacts is more beneficial than building a new software system from scratch.

Software artifacts that can be reused include domain models, requirement specifications, design, documentation, test data and code [3]. The first three types of artifacts are referred to as early-stage reusable artifacts, while the remaining three are referred to as later-stage reusable artifacts [3]. Clearly, it is more beneficial to reuse early-stage artifacts than later-stage artifacts because once a matching early-stage artifact is found, all later-stage artifacts related to the matched artifact can also be reused.

This research focuses on the retrieval of early-stage artifacts that are represented using the Unified Modeling Language (UML). In particular, we concentrate on class diagrams since they are the de facto standard in the design stage of the software development process [4].

The rest of this paper is organized in the following manner. We discuss related work in Section II. Section III describes a graphical representation for UML class diagrams. A similarity metric for comparing class diagrams is presented in Section IV. Section V explains the use of Genetic Algorithm (GA) in matching of class diagrams, while our experiments are described in Section VI. Finally, we present our conclusions and future work in Section VII.

II. LITERATURE SURVEY

This research focuses on the use of GA for retrieving UML class diagrams from the repository for reuse. Retrieving or locating reusable artifacts is a search problem involving a comparison of query artifacts and repository artifacts [3]. In this section, we review research regarding software artifact reuse. In particular, because the UML is the de facto modeling language for software systems, we describe previous work regarding retrieval of software artifacts represented using UML.

Robinson et al [5] developed a CASE tool that automatically retrieves similar sequence diagrams from a repository using a graph matching algorithm called SUBDUE. Ahmed [6] has applied GA for matching sequence diagrams based on their structural and semantic relatedness.

Rufai [3] proposed a set of similarity metrics to measure class diagram similarity based on semantic relatedness of class names, attributes and methods. Gomes et al [7] combine Wordnet and Case Based Reasoning (CBR) for retrieving UML models. Because they represented cases as UML class diagrams, their work involved retrieval of UML class diagrams. Robles et al [4] have used domain and application ontologies for class diagram retrieval. They use query expansion to match class diagrams, while we employ GA.

Blok et al [8] have matched Use Cases by computing a similarity measure of their event flow vectors.

Our method is similar to that of Gupta et al [9] who use inexact graph matching to detect design patterns. However, while we GA, the authors use an iterative method to match graphs. Their algorithm has a computational complexity of $O(n^2K^n)$ where n is the number of nodes and K is the number of phases in the algorithm, specified by the user.

Park et al [10] have presented a two stage method for matching UML specifications. In the first stage, Class Diagrams are compared using analogy. In the second stage, Sequence Diagrams are compared based on graph similarity.

This research was supported by NSTIP Project Grant 11-INF1633-04

III. GRAPH REPRESENTATION OF CLASS DIAGRAMS

UML class diagrams can be converted to labeled directed graphs in which the classes are represented by nodes, and the relationships between the classes are represented as edges of the graph. In addition, edges contain extra information which specify whether they represent dependencies, generalization, association, and so on. With this representation in mind, the problem of matching a query class diagram to another class diagram in the repository becomes that of graph matching. In particular, since the graphs to be compared usually have different numbers of nodes and edges, the problem is referred to as inexact graph matching [11]. Fig. 1 shows how a class diagram is converted to a directed graph. An adjacency matrix representation of the graph is also shown in Table I. Rather than containing zeros and ones, the entries of the matrix show the types of Class relationships represented by the edges of the graph.

IV. SIMILARITY METRIC

We propose a similarity metric which is composed of two parts; name (semantic) similarity and structure (topology) similarity [4], [12]. Name similarity measures the semantic relatedness of the concepts (classes) in the class diagrams to be compared, while structure similarity measures how closely the relationships between classes match one another. In other words, name similarity determines how the corresponding nodes of two graphs are related, while structure similarity measures the level of similarity between corresponding edges [13], [9].

A. Name Similarity

One way of measuring the semantic relatedness of class names is to use domain ontology [4]. Another possibility is to utilize a lexical database such as WordNet.

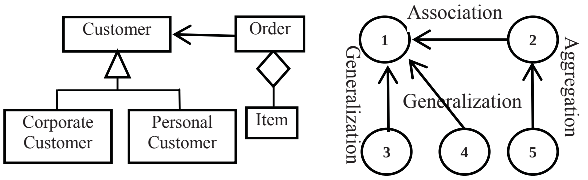


Figure 1. A Class Diagram and its corresponding directed graph. Nodes 1, 2, 3, 4 and 5 in the graph represent the Customer, Order, Corporate Customer, Personal Customer and Item Classes

TABLE I. ADJACENCY MATRIX

	1	2	3	4	5
1	None	None	None	None	None
2	Association	None	None	None	None
3	Generalization	None	None	None	None
4	Generalization	None	None	None	None
5	None	Aggregation	None	None	None

We propose using WordNet as is done in [3]. The Name Similarity of two class diagrams A and B having equal number of classes is given in (1).

$$NS(A,B) = \frac{\sum_{i=1}^{n} cns(A_i, B_i)}{n} \qquad (1)$$

A_i is the name of the i th class in class diagram A, B_i is the name of the i th class in class diagram B while n is the number of classes contained in both diagrams. *cns* (Class Name Similarity) is a function that returns a semantic relatedness value between zero and one. Zero and one denote maximal relatedness and un-relatedness of concepts, respectively. The division by n in equation 1 ensures that the value of name similarity (NS) always lies between 0 and 1.

B. Structure Similarity

In order to measure the structure similarity between two matrices, we define a square matrix Diff, whose entries represent the level of dissimilarity between the various types of class relationships. The (i, j)th entry of the matrix is a measure of the dissimilarity between the i th type of relationship and the j th type of relationship. A value of 1 indicates that the two relationships are extremely dissimilar, while 0 indicates that the relationships are the same (hence the diagonal entries of the matrix are all zeros). The entries of this matrix can be filled by gathering information from UML experts, or by applying ontology as in [4]. Table II (adapted from [4]) shows a sample Difference matrix (Diff). Since the main objective of retrieving class diagrams is to reuse them, the entries in Diff should be proportional to the amount of effort required to convert one type of relationship to another, after retrieving a class diagram from the repository. The last row labeled 'None' shows the level of dissimilarity between having no relationship between two classes (that is no edge connecting the vertices) and having a relationship between the two classes.

Let A and B be two directed graphs (representing class diagrams) each having n nodes. In addition, let the n x n matrices AdjA and AdjB be the adjacency matrices of A and B, respectively. The Structure Similarity (SS) between A and B is computed as shown in (2). nm is the number of times the edges in both graphs match exactly, while nu is the number of times the edges do not match.

$$SS(A,B) = \frac{\sum_{i,j} Diff\,(AdjA(i,j), AdjB(i,j))}{nm + nu} \qquad (2)$$

The overall similarity metric between two class diagrams represented by graphs A and B is a weighted sum of the Name Similarity and Structure Similarity as shown in (3). α is a value between 0 and 1 that determines the relative importance of SS and NS.

$$S(A,B) = \alpha * SS(A,B) + (1-\alpha) * NS(A,B) \qquad (3)$$

V. APPLICATION OF GENETIC ALGORITHM

As previously mentioned, in practice the class diagrams in the repository would usually have different number of classes than the number of classes in the query class diagram. While discussing the Name Similarity and Structure Similarity measures in the last section, we have assumed that the class diagrams have equal number of classes. In this section we explain how GA can be used to select an equal number of classes from both diagrams. Wang et al [14] used GA to match 2 graphs having the same number of nodes. In contrast, our method can match graphs having different number of nodes.

A. Chromosome Design

Let A and B be two class diagrams having na and nb classes respectively such that na ⩽ nb. The task of choosing how all the na classes of A will be mapped to na classes in B is a combinatorial optimization problem. GA has been used to solve combinatorial optimization problems such as timetabling, scheduling, Travelling Salesman Problem, Eight Queens Chess problem and so on. Fig. 2 shows a suitable encoding of a chromosome to determine the mapping of all the na classes from A to na classes in B.

Each gene in the chromosome represents a class number in B. For example, from Fig. 2 we observe that the 1st class in A is mapped to the 5th class in B, the 2nd class in A is mapped to the nbth class in B, the 3rd class in A is mapped to the nath class in B and so on. In this way, we map the classes in A to a subset of the classes in B. The selected classes in B maintain their class relationships only if both classes involved in the relationship were chosen as part of the chromosome.

B. Fitness Function

We use the similarity measure S given in equation 3 as our fitness function. Since the value of S always ranges from 0 to 1 (where 0 implies the highest level of similarity), successive generations of GA should produces less values of S compared to previous generations. This results in GA selecting (near) optimal mappings from classes in the query class diagram to those in repository class diagrams.

VI. EXPERIMENTS

In this section, we present the results of experiments carried out using our proposed method. Our experiments focused mainly on Structure Similarity (SS), hence the weight α in equation 3 was set to 1. Our objective was to determine the efficiency of the proposed method in retrieving matching class diagrams from the repository. Fig. 3 shows two class diagrams; a query class diagram Q and a class diagram R from the repository. As shown in the figure, Q is isomorphic to a sub-graph of R.

We used the proposed method to determine how many times the classes in Q were correctly mapped to the classes in R. Maximum similarity is obtained when the value of the fitness function is zero as described in Section V. Fig. 4 shows the mean and standard deviation of the fitness value over 500 successive generations. The experiment was repeated 100 times. After a few generations, the mean and standard deviations of the fitness function stabilizes, indicating that there is no additional benefit in running GA further.

TABLE II. DIFF MATRIX

	AS	AG	CO	DE	GE	RE	IR	NO
AS	0	0.11	0.11	0.45	0.45	0.66	0.77	1
AG	0.11	0	0.11	0.45	0.45	0.66	0.77	1
CO	0.11	0.11	0	0.45	0.45	0.66	0.77	1
DE	0.49	0.49	0.49	0	0.28	0.21	0.32	1
GE	0.49	0.49	0.49	0.28	0	0.49	0.6	1
RE	0.83	0.83	0.83	0.34	0.62	0	0.11	1
IR	1	1	1	0.51	0.79	0.17	0	1
NO	1	1	1	1	1	1	1	0

AS = ASSOCIATION, AG = AGGREGATION, CO = COMPOSITION, DE = DEPENDENCY, GE = GENERALIZATION, RE = REALIZATION, IR = INTERFACE REALIZATION, NO = NO RELATION

In addition, the mean value of the fitness function is often sufficiently close to zero because the proposed algorithm finds exact matches most of the time. However, the standard deviation of the fitness value is higher than the mean in most cases. This is because the algorithm usually finds optimal solutions, but obtains near optimal solutions at other times.

In another set of experiments, we studied the impact of using GA in our algorithm. We replaced the GA component of our algorithm with a random matching of query class diagrams to repository class diagrams. The experiment was repeated 1,000 times. For the GA-based experiments, there were 100 individuals in each generation, and the maximum number of generations was 50. Thus, the maximum number of iterations was 5,000 across all generations. In the case of the experiments based on random matching of classes, the matchings were based on randomly generated permutations. The random permutations were generated and tested up to 5,000 times. As in the case of GA, the search was abandoned as soon as an optimal matching was found. The results shown in Fig. 5 indicate that our GA-based algorithm usually finds optimal class matchings in less number of iterations compared with random matching of classes. The figure shows that in 700 out of the 1000 cases, our GA-based algorithm finds optimal matchings after a few generations (≤ 1,500 iterations or alternatively ≤ 15 generations). However, 30% of the time, GA executes all the 5,000 iterations (50 generations) and determines near-optimal class matchings.

VII. CONCLUSION AND FUTURE WORK

We have described a method for retrieving UML class diagrams from a software repository by using GA. Few experiments were carried out to measure the effectiveness of the proposed algorithm in detecting structural similarity. No experiment was carried out regarding name similarity. Thus, it is necessary to perform many more experiments to evaluate the performance of the proposed method in terms of precision, recall, execution time and so on.

We have considered only class names and class topology. In the future, we hope to include class attributes and operations in the similarity measure. In addition, the technique will be extended to determine similarity measures of other UML diagrams such as sequence diagrams and state chart diagrams. The development of a tool to integrate our proposed technique as an add-in to popular UML modeling software like Rational Rose and Enterprise Architect is also being considered.

1	2	3	4 .		.	na
5	nb	na	2			3

Figure 2. Chromosome Encoding for GA

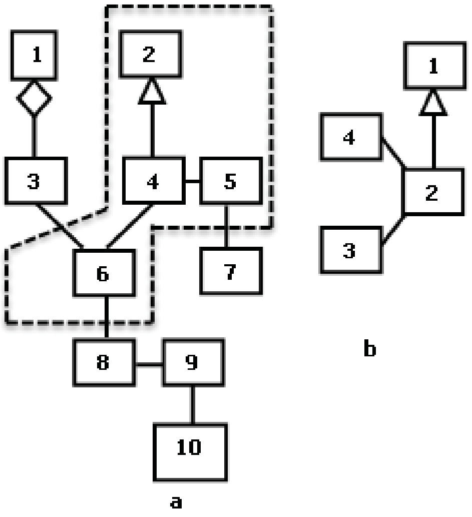


Figure 3. Repository and Query Class Diagrams. a) Repository Class Diagram b) Query Class Diagram. Classes 1, 2, 3 and 4 from the Query Diagram can be mapped to Classes 2, 4, 6 and 5 in the Repository Diagram respectively.

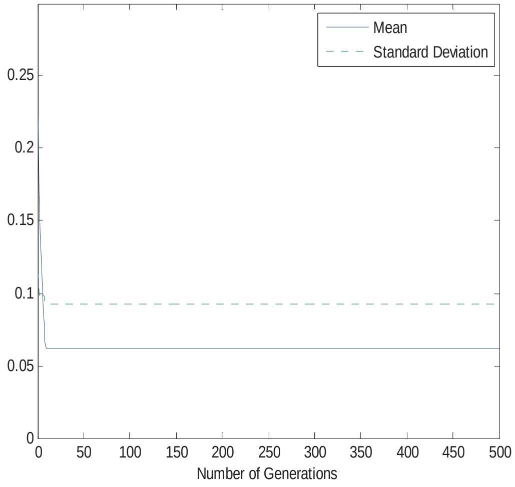


Figure 4. Mean and Standard Deviation of similarity value over 500 generations

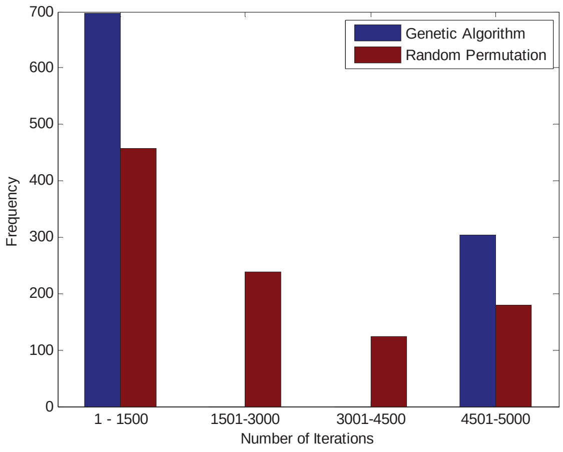


Figure 5. Number of required iterations in GA-based matching and random matching of classes.

ACKNOWLEDGMENT

The authors would like to acknowledge the support provided by the Deanship of Scientific Research at King Fahd University of Petroleum and Minerals (KFUPM) under Research Grant 11-INF1633-04.

REFERENCES

[1] Krueger C. W, “Software reuse,” ACM Comput. Surv. vol.24, 2, pp 131-183, 1992.

[2] I. Sommerville, “Software engineering,” 9th ed., Addison-Wesley, 2010.

[3] R. Rufai, “New structural similarity metrics for the UML, ” MS Thesis, King Fahd University of Petroleum & Minerals, Dhahran, Saudi Arabia, 2003.

[4] K.. Robles, A. Fraga, J. Morat and J. Llorens, “Towards an ontology - based retrieval of UML class diagrams,” Inf. Softw. Technol. vol. 54, no. 1, pp. 72-86, 2012

[5] W. N. Robinson and H. G. Woo, "Finding reusable UML sequence diagrams automatically," IEEE Software, vol. 21, pp. 60-67, 2004.

[6] A. Ahmed, “Functional similarity metric for UML Models, ” MS Thesis King Fahd University of Petroleum & Minerals, Dhahran, Saudi Arabia, 2006.

[7] P. Gomes et al, “Using WordNet for case-based retrieval of UML models”. AI Commun. vol. 17, no. 1, pp 13-23, 2004.

[8] M.C. Blok and J. L. Cybulski, J.L, "Reusing UML specifications in a constrained application domain," in Software Engineering Conference, 1998. Proceedings. 1998 Asia Pacific, 1998, pp.196-202.

[9] M. Gupta, R. Singh, A. Tripathi, “Design pattern detection using inexact graph matching,” in International Conference on Communication and Computational Intelligence (INCOCCI), 2010, pp.211-217.

[10] W. Park and D. Bae, “A two-stage framework for UML specification matching,” Inf. Softw. Technol., vol. 53, no. 3, pp. 230-244, March 2011.

[11] E. Bengoetxea, “Inexact graph matching using estimation of distribution algorithms,” Phd Dissertation, University of the Basque Country, 2002.

[12] Z. Xing, E. Stroulia, “UMLDiff: an algorithm for object-oriented design differencing,” in Proceedings of the 20th IEEE/ACM international Conference on Automated Software Engineering, 2005, pp. 54-65.

[13] A. Hlaoui, S. Wang, “A new algorithm for inexact graph matching,” in 16th International Conference on Pattern Recognition, vol. 4, 2002, pp. 180- 183.

[14] Y. Wang, K. Fan, and J. Horng, “Genetic-based search for error-correcting graph isomorphism,” IEEE Trans. on SMC, vol. 27, no. 4, pp.588-597, Aug 1997.

Managing Linear Hash in a Closed Space

Satoshi NARATA Takao MIURA
Dept.of Elect. and Elect. Engineering, HOSEI University
Kajinocho 3-7-2, Koganei, Tokyo, Japan
E-mail: satoshi.narata.ze@stu.hosei.ac.jp miurat@k.hosei.ac.jp

Abstract

In this investigation, we propose a new approach for overflow management of linear hash organization by which we can reduce excessive access of dynamic hash techniques. We propose Closed Linear Hash organization to relieve the situation. It manages all data without an overflow Area while keeping all the aspects of linear hash.

Keywords: *Closed Linear Hash, Linear Hash, Overflow Management, Data Structure*

1. Motivation

In modern computer processing, *hash approach* provides us with excellent data management such as access in O(1), though there exist fundamental deficiencies such as *data collision*, *spill-out* and *space limitation* caused by *static size* of hash spaces, as described in every textbook. By *Linear Hash* (LH)[1], hash space grows *smoothly* (i.e., space grows in an one by one manner) for relieving overflow situation, and we expect good efficiency (small overhead of I/O) while keeping *density*[1] constant.

However in LH there is no feature for collective updates such as bulk inserts[5, 4], and we are forced to split a bucket whenever *insert* operation happens to maintain the density close to our specified density factor σ, called *load factor*. Since there is no explicit condition like load-factor against bucket-splitting, we can't improve this issue within a general framework. More serious is that bucket-splitting doesn't always relieve overflow buckets and the splitting may not improve efficiency. Especially we face to severe deficiencies to bulk insert/delete operations, because the successive operations cause heavy manipulation (one extension at each operation) so that huge amount of I/O access happen to the secondary storage devices, called *thrashing*.

The authors have discussed and proposed a new approach for bulk insert (delete) to relieve the thrashing situation and to reduce total I/O access, say 40 %. The basic idea comes from bulk extension of the space, we accept many records (for insert) at once, reorganize hash-space as well as new records and generate new *image* of the space[3]. On the other hand, we got a problem of density reduction, about 50% worse. In this investigation, we propose a *closed* linear hash, which means no overflow area in hash space but only primary area. Our experimental results show the approach is really promising.

In section 2 we review Linear Hash and describe outstanding aspects of this technique. In section 3, we discuss a new approach of Closed Linear Hash (CLH) Section 4 contains some experiments, several analysis and the comparison with other approach. We conclude our investigation in section 5.

2. Linear Hashing

In this section, we review Linear Hash very quickly. Fore more detail, see the literatures[1],[2]. Given a linear hash (LH) space consisting of a set of buckets, a possible domain C of keys, we assume a record d, key $c \in C$ of d and a hash function $h : C \rightarrow H$, then we can identify a bucket position of d which contains d in the hash space by an address $h(c)$. So we can go to the position by $h(c)$ directly (on the secondary storage). Each bucket in a primary hash space may contain several records.

Given two non-negative integers L (called a *level* of LH space) and p (called a *growth position*) where $0 \leq p < 2^L$, we assume a hash function h_L where $h_L(c) \leq 2^L$ for any key c in such a way that $h_{k+1}(c)$ is identical to $h_k(c)$ on the low k bits for any k and c so that $h_{k+1}(c)$ might be different from $h_k(c)$ only at $(k+1)$-th bit position. A trivial example is $h_k(c) = c \mod 2^k$ considered c as bit sequence.

For given a key c, we can retrieve a record containing c as a key by the algorithm:

```
a := $h_L(c)$;
a := $h_{L+1}(c)$ if a < p
```

That is, we obtain the value $a = h_L(c)$ first, then if $h_L(c) <$

[1] *Density* means the ratio of the number of stored records to the space size.

p, we calculate a again by $a = h_{L+1}(c)$. Once we get the position a, we access the bucket and examine whether this bucket and the overflows contain c or not.

To insert a record d with a key c, we apply the algorithm illustrated above to obtain a. In this case, when we insert d, we might put it into an overflow area if a-th bucket is full. Each overflow bucket may contain several records. After inserting d, we examine the status of the LH space whether there exist too many records or not, by checking density, for example. If the density goes beyond σ, we should split some bucket and add new buckets to relieve the status. In LH, we select the bucket of the growth point n and distribute (or *split*) all the records into the two buckets of p and of $p + 2^L$ according to their h_{L+1} values[2]. Finally we increment p. Note that we should generate $p + 2^L$-th bucket as a new one and this means the LH space grows smoothly.

Also note that the growth point p should be $p \leq 2^L$. We adjust this condition by the condition:

```
$p := 0$ if $p + 1 \geq 2^L$
else $p := p + 1, L := L + 1$
```

Let us note that a bucket in the primary area may be splitted only if there are too many records to keep them in the bucket, or if the global density goes beyond a threshold by this insertion. To relieve the situation, an overflow arises in the former case while a split happens in the latter case.

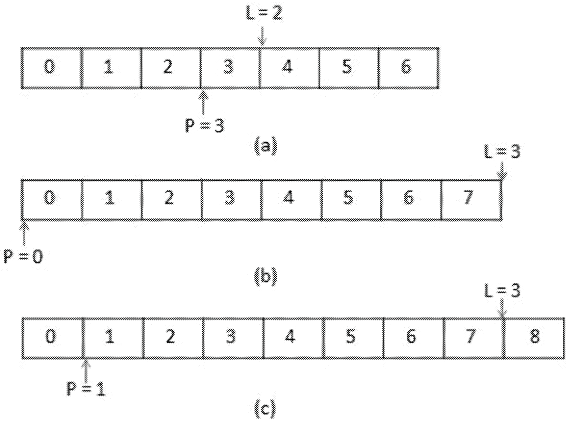


Figure 1. Linear Hashing

In figure 1 (a), assume we have the global level $L = 2$ and a growth position $P = 3$. The buckets 0,1 and 2 have already been split thus we have the buckets 4,5 and 6 respectively. Then we split a bucket 3 whenever the density goes over the threshold σ and we have $P = 4$. Once P becomes 2^L, we restart with $P = 0$ but $L = 3$ as shown in the figure (b). We repeat the whole process. We may split the bucket 0 into the two of 0 and 8 and set $P = 1$ as shown in the figure (c).

By *Linear Hash* (LH), we split buckets and append new ones in an one by one manner to keep density constant which allows us to relieve overflow situation, and we don't need to allocate hash space in advance while achieving good efficiency (small overhead of I/O). However in LH, bucket-splitting doesn't always relieve overflow buckets and a new bucket is appended to the end of the space because of *linear* property. We may get severe deficiencies to bulk insert/delete operations, because the successive operations cause heavy manipulation (one extension at each operation) so that huge amount of I/O access happen to the secondary storage devices, called *thrashing*.

Rafiei et al. examines the situation and proposes how to allocate LH space distribution in advance only for LH creation[4]. Yasuda et al. has proposed *Tree Hash* to recover overflow problem[5] where they don't manage hash spaces linearly but extend the space over tree-structure spaces so that some additional management mechanism has to be introduced.

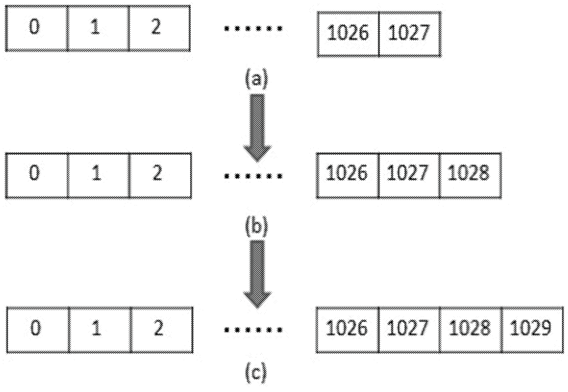


Figure 2. I/O Thrashing

Let us explain this problem by a small example. The figure 2 (a) shows a hash space consisting of 1028 buckets with the global density threshold $\sigma = 0.8$, assuming the space containing 822 records thus the current density = $822/1028 = 0.7996$. When we insert *one* record, the density becomes 0.8007 thus bucket splitting happens as in the figure (b). Note we have 1029 buckets and the density $823/1029 = 0.7998$. When we add *one more* record, the density is $824/1029 = 0.8008$ and the splitting arises again as in the figure (c).

As this example says, once the splitting happens, we have much amount of splitting for bulk insert of many records. The problem is called *I/O thrashing*[5].

[2] For any key c' in the bucket, we have $n = h_L(c')$, and $h_{L+1}(c') = p$ or $h_{L+1}(c') = p + 2^L$ by definition.

3. Closed Linear Hash

3.1. Describing Data

As said previously, in LH, bucket-splitting doesn't always relieve overflow buckets because other buckets might be split to improve density. In this work we introduce LH *without* overflow area but the overflow records within the primary area. The approach is called *Closed Linear Hash* (CLH). In CLH, a bucket may be split according to dynamic growth (the bucket is located at the growth position) when density goes beyond threshold, *or* when a bucket has too many records to keep them within this bucket. That is, we can split buckets of the problem only if these records can be distributed well over several buckets (otherwise we put them in a list.). Thus there might be unallocated buckets in the primary hash space so that we can improve space efficiency and avoid I/O overhead caused by overflow.

Let us discuss how to manage Closed Linear Hash space. As in conventional LH, there are 3 parameters L (global level), P (growth position) and σ (global data density). Here in CLH, we also maintain ℓ in each bucket to describe a local level ℓ of this bucket ($0 \leq \ell \leq L$) or some status, *free* ($\ell = -2$) or *overflow* ($\ell = -1$). There are some other information such as number of records in this bucket, denoted by thr. We manage a list of free buckets on hash space and we assume the list is not empty (we expand the space by generating free buckets if necessary). Let us note that the growth position P can't go beyond the last position, $Last$, of the space; if in the case we expand the space by free buckets.

3.2. Retrieving Data

Let us discuss how to retrieve data in CLH space. Basically there is no difference between LH and CLH except one point: the bucket to be accessed might be assigned as an overflow or a free bucket so that we should examine the bucket validity by inspecting local levels/status. If the bucket is valid, we examine the bucket and its overflow buckets to obtain the record. If it is not valid (a free bucket or an overflow bucket), we decrease the level value until we get to the valid one. We apply retrieval of a key X:
(0) Let k be a global level L. We like to find X.
(1) We calculate a hash value v of X by $H[k](X) = X \bmod 2^k$. If $v < P$, we obtain a re-hashed value v by $H[k+1](X)$.
(2) We get the bucket at v and examine the local level ℓ. If it is a free or overflow bucket, we decrease k and repeat the process until we get the valid ℓ ($\ell \geq 0$). This step goes L times at most.
(3) Once we get the valid bucket, we examine whether the bucket contains X or not. If not, we go to the overflow buckets and repeat the process. There is no X if the buckets don't contain X.

Given a global level $L = 3$, the global threshold $THR = 0.90$, the local threshold $thr = 0.80$ and the growth position $P = 1$, we assume each bucket contains 5 records at most. Let us start with retrieving X where the hash-value is assumed 2. We go to the bucket 2 because $2 < P$ and check the local level ℓ whether $\ell \geq 0$ or not. Then we examine the bucket (and the overflows) may contain X.

3.3. Inserting Data

To insert (add) data X in CLH space, we first get to the bucket just same as retrieval. We insert the data X into the bucket if there exist some room, otherwise we go to overflow buckets to see some room. In principle, we examine whether the bucket can be split or not. That is, if $0 \leq \ell < L$ and all the data in the buckets can be split into two groups, we do splitting. In other cases (i.e., they can't be split or $\ell = L$), we manage overflow data including X by overflow buckets newly from a list of free buckets. Let us summarize an algorithm for CLH insertion of a key X:
(0) Assume $K = L$ and let us insert X.
(1) First of all, we come to a bucket to be accessed according to the retrieval process. If the bucket contains X, we stop the process.
(2) We examine whether the bucket can be split or not. That is, if $0 \leq \ell < L$ and all the data in the buckets can be split into two non-empty groups, we do splitting as shown in the next subsection.
(3) In other cases (i.e., they can't be split or $\ell = L$), we add X to the bucket or manage X by overflow buckets; we follow overflow buckets to examine room or obtain a new (overflow) bucket from a free list of buckets (See figure 3).

Let us see how the insertion works in CLH. Assume figure 3 shows CLH space where the global level $L = 2$, the global threshold $THR = 0.90$, the local threshold $thr = 0.80$ and the growth position $P = 1$, containing 24 records. Let us insert a new record of the hash value 0. Since $0 < P$, we rehash and assume the rehash value is 4, and the $\ell \geq 0$ but full of records. Then we obtain a free bucket at 6 and link the new bucket to the 4-th. The new bucket now contain "overflow" records and $\ell = -1$.

3.4. Splitting Buckets

The heart of LH approach is *splitting* buckets to maintain hash space linearly. The situation is same as CLH but the difference comes from a fact that splitting may arise when inserting in CLH. Let us discuss how to split buckets in two cases, *space growth* and *insertion.*

First we show space growth in CLH. There is a global threshold P. Triggering arises whenever both the data den-

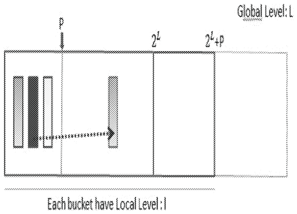


Figure 3. Inserting Records into Free Buckets

sity goes beyond this value and the local data density goes beyond a local threshold. Remember that each bucket keeps local level ℓ and the local density. Once a trigger arises, we examine a bucket at the growth position P. No split arises if new local density is below thr or we can't two non-empty groups with level $\ell+1$. We split the bucket only if the local density goes beyond thr thr, then all the relevant records maybe split into two non-empty groups. Then we move P to the next (or $P = 0, L = L+1$ if $P = 2^L$). See the situation in figure 4.

We split the bucket (of the local level ℓ) by rehashing all the data in the buckets and the overflow buckets by the level $\ell+1$. Note $\ell \leq L$ and we need a new bucket at $P + 2^\ell$ ($< 2^{L+1}$). We put the level $\ell+1$ to the bucket at P. The new bucket is either *free* or assigned as an *overflow* already. We put the level $\ell+1$ into the new bucket if if free. Otherwise, we get a new (free) bucket and move all the content of the overflow bucket at $P + 2^\ell$ to the new bucket, then we put the last half of the splitting bucket.

Here is an algorithm for splitting by space growth:

(1) Splitting by space growth may arise whenever the data density goes beyond a given global threshold THR and the local data density at the bucket goes beyond a local threshold thr.

(2) We examine the bucket and all the buckets linked to this bucket as overflows, and split the bucket only if the local density goes beyond and all the relevant data can be split into two non-empty groups.

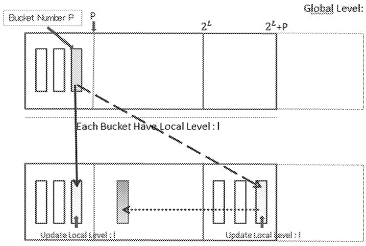


Figure 4. Splitting Bucket at Growth Position

(3) To split the bucket (of local level ℓ) at P, we increment the local level at the bucket P, and move the last half of data to the new bucket at $P + 2^\ell$ with the new local level $\ell+1$. If the bucket at $P + 2^\ell$ has been assigned as an overflow, we get a new (free) bucket and move all the content of the overflow bucket at $P + 2^\ell$ to the new bucket, then we put the last half of the splitting bucket.

(4) Finally we move P to the next (or $P = 0, L = L+1$ if $P = 2^L$).

Another splitting may happen just when inserting data. When inserting data X, we come to the bucket (at k) that should contain X, and examine $\ell < L$ or $k < P$. Then we examine whether the local density goes beyond a local threshold or not, and also whether all the relevant data can be split into two non-empty groups or not. If so, we do splitting this bucket. The process for splitting is just same as space growth case. Note that, in conventional LH, no splitting arises at inserting.

Let us summarize splitting at inserting.

(1) We come to a bucket at k to insert X, examine the local density goes beyond a local threshold and see the local level $\ell < L$ or the position $k < P$. We stop the process if not.

(2) We split the bucket as (3) of space growth.

Let us show some example of splitting at insertion in figure 5. Here we assume CLH space where the global level $L = 2$, the global threshold $THR = 0.90$, the local threshold $thr = 0.80$ and the growth position $P = 3$, containing 27 records. At inserting a record at a bucket $v < P$, we assume the data density of v goes beyond thr and $\ell \leq L$. Then we split a bucket v into v and $v + 2^L$. Unfortunately, the new bucket may be allocated as an overflow bucket so that we obtain a free bucket and move the overflow records to the free bucket. Then we can go back to the split process since the bucket at $v + 2^L$ is now available. Both buckets at v and $v + 2^L$ have $\ell+1$ as local level.

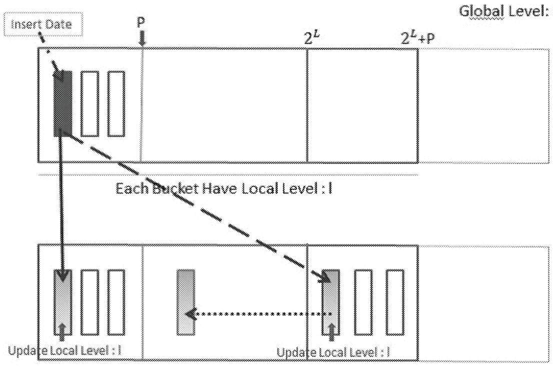


Figure 5. Splitting at Insertion

4. Experimental Results

Let us discuss the experimental results to see how well the proposed CLH algorithms work. We prepare experimental data for our experiments, consisting of 5 files containing 5000, 10000, 50000 and 60000 records respectively coming from 123593 postal addresses (points) which represent three metropolitan areas (New York, Philadelphia and Boston) in the US, abbreviated by "ZIP". It is known to be three clusters here in the form of non-uniformly distributed rural areas and smaller population centers[3].

We examine load/insert these records. We also examine all the frequencies of I/O requests to 10000 records queries where half of them fail.

During our experiments we assume one bucket can keep 5 records of LH and CLH spaces in primary/overflow area. We examine all the frequencies of I/O requests within the Primary and overflow area, the number of overflow records, as well as LH levels, densities and load factors.

For LH space, we put $0.7, 0.8, 0.9, 0.95$ as global threshold values THR (or load factors), and we manage overflow records by means of a linear list to each primary bucket. For CLH space, there is no overflow area, we we put $0.7, 0.8, 0.9, 0.95$ as global threshold THR values and 0.8 as a local threshold value thr.

In this experiment, we create LH and CLH spaces of 5000, 10000, 50000 and 600000 records. We add 5000, 10000, 50000 and 600000 records to a space containing 50000 records in LH and CLH. To these LH spaces, we examine 10000 queries where half of them have no answers and take counts of bucket access (called *touch count*). As the baseline, we examine conventional LH files and we compare our CLH to the LH[4].

We show the results in the following tables 1 to 4. First of all, let us discuss I/O frequencies as shown in tables 1 and 3 of space creation. For example, let us see 60000 records with a global threshold 0.95, and we see CLH Read frequency is 0.57 times better than LH while CLH Write frequency is 1.34 times worse. Similarly in tables 2 and 4, when we apply addition to CLH/LH space, for instance, when we add 60000 records to the space of 50000 records, we see CLH Read frequency is 0.62 times better while Write frequency 1.28 times worse. This means that, to find desired records, we don't follow overflow chains any more but come to the data directly through hash function. On the other hand, when we insert/add records, we may move bucket contents many times to avoid overflow chains, which cause more Write frequency.

Let us compare query efficiency (I/O bucket touch counts) of 10000 key search where 5000 keys don't appear. In tables 1 and 3, just after space creation with global thresholds 0.9 and 0.95, we need 212871 touches in average in LH for 10000 key search while we get 12105 in average in CLH, 0.57 times better. Similarly in tables 2 and 4, just after addition with global thresholds 0.9 and 0.95, we need 20263 touches in average in LH for 10000 key search while we get 12092 in average in CLH, 0.60 times better.

Let us see how overflow data works. In tables 1 and 3 of space creation, let us compare overflow records. We see CLH overflow records is 0.46 times less number in average than LH (0.36 times at best). Similarly in tables 2 and 4 of addition, we see CLH overflow records is 0.42 times less in average than LH (0.37 times at best). In either case, we can reduce the hyge number of overflow records and put them into primary hash space. This is why we can improve Read efficiency dramatically.

Clearly overflow records are reduced dramatically, this fact provides us with efficient Read capability at the cost of Write frequency to avoid overflow. That is, overflow management severely damages Read efficiency in LH. Our experiement also shows that Read frequency heavily depends on global threshold, not on the number of records in CLH. It seems easier to manage efficiency only by looking at the threshold but not at data distortion.

THR	Records	PR	PW	IOQ	OF	L	P
0.7	5000	8154	8460	10993	312	10	934
0.7	10000	16330	17252	10788	622	11	1649
0.7	50000	82384	85810	10747	3554	14	949
0.7	60000	100378	105251	10845	4909	14	5492
0.8	5000	9989	9483	11870	553	10	431
0.8	10000	20209	19288	11279	1082	11	870
0.8	50000	103666	98622	11330	5511	13	7135
0.8	60000	123147	116808	11258	6845	14	795
0.9	5000	10435	9319	12417	733	10	188
0.9	10000	21294	19026	11747	1461	11	402
0.9	50000	109496	97701	11912	8187	13	4485
0.9	60000	132641	117943	11878	9416	13	7161
0.95	5000	10578	9113	12630	797	10	107
0.95	10000	21756	18842	11925	1603	11	214
0.95	50000	112395	97364	12200	9322	13	3546
0.95	60000	135854	117206	12215	11133	13	6059

Table 3. CLH Creation based on ZIP Data

5. Conclusion

In this work, we have proposed a new approach of Closed Linear Hash (CLH). There is no overflow area, less number of bucket splitting and less number (say 60 percents) of bucket search at the some cost of bucket writes (about 20 percents more). Unlike LH, overflow records are managed well within hash space even if data distortion happen. We believe CLH is quite promising for LH.

[3] www.rtreeportal.org

[4] In the tables, "PR" means the number of PrimaryRead buckets, "PW" PrimaryWrite, "OR" OverflowRead, "OW" OverflowWrite, "IOQ" the number of I/O for Query and "OF" OverflowRecords.

THR	Records	PR	PW	OR	OW	IOQ	OF	L	P
0.7	5000	8590	5163	6954	3503	14464	708	10	203
0.7	10000	17177	10370	13732	6940	13515	1403	11	409
0.7	50000	84122	49676	69163	35162	14122	9141	13	3483
0.7	60000	101178	59077	85301	43044	13904	10754	13	5879
0.8	5000	8065	3951	8835	4258	15415	867	10	10
0.8	10000	16136	7989	17304	8409	14611	1725	11	21
0.8	50000	78329	38192	83724	41335	15425	11704	13	1383
0.8	60000	92161	43918	100180	49592	16328	16513	13	2680
0.9	5000	7048	2346	10697	4874	22741	1908	9	176
0.9	10000	14027	4691	21026	9703	20641	3863	10	340
0.9	50000	73237	25337	109676	49792	17291	15011	12	3679
0.9	60000	86457	29658	128156	58915	18143	20132	13	668
0.95	5000	6695	1648	12573	5243	25488	2303	9	56
0.95	10000	13472	3358	24653	10490	23166	4487	10	137
0.95	50000	66013	15678	122257	52088	22792	24602	12	1251
0.95	60000	82102	20726	152755	63907	20895	24950	12	3283

Table 1. LH Creation based on ZIP Data

THR	Records	PR	PW	OR	OW	IOQ	OF	L	P
0.7	5000	8298	4551	7756	3859	14170	10208	13	4606
0.7	10000	17353	9798	16038	7802	13840	10407	13	5978
0.7	50000	82050	45911	73624	36929	14357	20639	14	6291
0.7	60000	97977	54199	89613	44771	14439	23533	14	8321
0.8	5000	6923	2997	7848	4007	15954	14091	13	2036
0.8	10000	14045	6055	16247	8155	16268	16225	13	2752
0.8	50000	76407	34910	88350	42815	15741	26001	14	2116
0.8	60000	89889	40373	104722	50955	16216	31237	14	3307
0.9	5000	7175	2666	9648	4695	17409	16729	13	313
0.9	10000	13324	4486	18363	9070	18125	19969	13	704
0.9	50000	66498	19669	106379	48165	19566	40125	13	5114
0.9	60000	85479	27548	136328	60407	17701	36630	13	8112
0.95	5000	7187	1926	14198	5524	22496	26136	12	1981
0.95	10000	16748	5551	31232	12081	20510	23900	12	3502
0.95	50000	64261	14125	122748	51760	23327	51967	13	1920
0.95	60000	76499	16307	149925	62106	23671	58420	13	2667

Table 2. LH Addition based on ZIP Data

THR	Records	PR	PW	IOQ	OF	L	P
0.7	5000	9181	9632	10803	4270	14	2911
0.7	10000	17808	19314	10817	4807	14	5409
0.7	50000	85794	88678	10795	8091	15	3256
0.7	60000	104684	109593	10846	9710	15	8644
0.8	5000	9083	8306	11255	5805	14	20
0.8	10000	19321	18032	11224	6720	14	722
0.8	50000	100232	93967	11313	11630	14	15004
0.8	60000	119369	110766	11236	12705	15	253
0.9	5000	11515	10114	11961	8934	13	5850
0.9	10000	22968	20163	11872	9322	13	7160
0.9	50000	114370	100437	11973	17463	14	9457
0.9	60000	137925	120831	11924	18573	14	12286
0.95	5000	11654	9965	12223	10270	13	4821
0.95	10000	23274	19810	12216	10984	13	6060
0.95	50000	115991	98835	12262	19798	14	7399
0.95	60000	139956	118924	12262	21626	14	10092

Table 4. CLH Addition based on ZIP Data

References

[1] Litwin, W.: Linear hashing - A New Tool for File and Table Addressing, VLDB, 1980

[2] Litwin, W., Neimat, M.-A. and Schneider, D.A. : LH* - Linear Hashing for Distributed Files, ACM SIGMOD, pp. 327-336, 1993

[3] Narata,S,, Miura, T.: On Inserting Bulk Data for Linear Hash Files, 3rd Network Digital Technologies(NDT), Macau, 2011

[4] Rafiei, D. and Hu, C.: Bulk Loading a Linear Hash File, DaWaK 2006, pp.23-32

[5] Distributed Processes on Tree Hash, IEEE Computer Software and Application Conference (COMPSAC), 2006, USA

CLAT: Collaborative Learning Adaptive Tutor [*]

Alaeddin M.H Alawawdeh, César Andrés and Luis Llana
Departamento Sistemas Informáticos y Computación
Universidad Complutense de Madrid
E-28040 Madrid. Spain.
e-mail: alaeddin@fdi.ucm.es, c.andres@fdi.ucm.es and llana@sip.ucm.es

Abstract

In this paper we introduce CLAT*, a multi user online e-learning tutor. This system is used to improve the knowledge of the students in different environments. A novel algorithm to share the knowledge is presented. It is based on letting the students to share their experience with the rest of their partners. In order to do this task,* CLAT *provides a dynamical structure that is updated under some constrains. In particular,* CLAT *adapts its behavior not only individually for each student but also by considering the performance of similar students. The core of its adaptive part is based on the classification of students into classes (groups of students sharing some attributes). By doing that, the past behavior of students of the same class determines how* CLAT *interacts, in the future, with students of that class. That is,* CLAT *learns how to deal with each type of student.*

Keywords: Intelligent Tutors, Education

1 Introduction

Online courses [4, 6, 1] and *collective knowledge* [7] approaches uncover several contradictory findings about the best way to *conduct teaching* and *learning online*. The importance of recognizing the feelings, reactions, and responses of the students in an online environment is really important, but it is a hard task. Most instructors or designers simply move text-based courses to the Internet following some form of pedagogy and there are very few studies that examine this phenomenon. But, there are several promising techniques focusing on this goal based on extracting structured data from unstructured user contributions [2, 3].

In this paper, a **C**ollaborative **L**earning **A**daptive **T**utor is presented CLAT. This system let users to interact with a set of tests, in order to increase their degree of knowledge. CLAT has been developed using *software engineering* principles, and make use of *formal methods* in the testing module. On the one hand, generally speaking, software engineering can be considered as a systematic and disciplined approach to develop software. It concerns all the aspects of the production cycle of software systems and requires expertise, in particular, in data management, design and algorithm paradigms, programming languages, and human-computer interfaces. It also demands an understanding of and appreciation for systematic design processes, non-functional properties, and large integrated systems. On the other hand formal methods refer to techniques based on mathematics for the specification, development, and verification of software and hardware systems. The use of formal methods is especially important in reliable systems where, due to safety and security reasons, it is important to ensure that errors are not included during the development process. Formal methods are particularly effective when used early in the development process, at the requirements and specification levels, but can be used for a completely formal development of a system. It has been argued, usually with very good arguments, both that formal methods are very appropriate to guide the development of systems and that, in practice, they are useless since software development teams are usually not very knowledgeable of formal methods in general, and have no knowledge at all of what academia is currently developing. It is interesting to remark that some of the strongest advocates of formal methods have almost "no experience on the development of real systems". Thus, one of the main goal of this paper is to present CLAT as a formal real system development for e-learning.

Next we briefly sketch the main characteristics of CLAT. First, CLAT combines individual profiles for each student with general profiles for each *class* of students. By doing so, CLAT is able to adapt its behavior with respect to a student not only *individually*, on the basis of her previous fails and successes, but also by taking into account the performance of the rest of students (current studentes as well as former students). That is, the system *learns* how to interact with a student by adapting the general profile of her class(es) to

[*] Research partially supported by the Spanish MCYT project TESIS project (TIN2009-14312-C02-01).

her necessities. However, the progress of the users while experimenting with the system is individually controlled. CLAT keeps profiles for each student where all the relevant information (from her previous sessions) is recorded. In particular, the system points to the next topic that the user should explore once she has reached a certain command in the current topic. Another important feature of CLAT is that it can automatically generate based on the student's skill. We consider that CLAT may increase the success rate of students in two ways. On the one hand, students can regularly check their progress by self-evaluation. On the other hand, teachers can find out which parts of the course are more difficult for the students (CLAT provides the teacher with a private interface that allows her to access the information about the performance of students). An additional contribution of CLAT is a module implemented to verify the identity of the students. This module makes use of Bayesian text filtering to determine the similarity between different executions of the same student.

The rest of the paper is structured as follows. In Section 2 we present the initial structure of CLAT. In Section 3 we present the idea of how to identify the users that are filling the tests. Following, in Section 4 we discuss different features and measures of CLAT. In Section 5 it is presented our system in a collective user scenario solving some features previously discussed. Finally, in Section 6 we give our conclusions and some lines for future work.

2 CLAT structure

In this section the initial structure of CLAT is presented . Let us note that the main advantage of using an intelligent tutor with adaptive capabilities is that it can be automatically adapted to the students. CLAT can adapt its behavior not only on the basis of the fails and successes of the current student, but also on the experiences of the rest of students.

In CLAT, at the same time, n different users can be connected. Previous interactions of the users is recorded on the server. This information is used for obtaining statistical data and for classifying the users in different statements. When a user asks a new test to the server, by sending Do test request, the server sends to this user Send test (x), where x is the test to be solved. After receiving the test, the user sends the answer to the server, and according with the stored results, it will consider it as a good answer or not.

CLAT implements a user classification. This distribution not only depends on the answers provided from the users on different topics tests, but also in the effort provide from each user to generate new tests.

Each group is a set of sorted users (according to the amount of gained points). So, the set of guru users is made of the u_1 best users of the ranking, the set of expert users is made of the next u_2 users, and so on. As it usually happens in knowledge communities, the amount of users in each class should follow a pyramidal structure. Thus, the condition $\mid u_1 \mid \; < \; \mid u_2 \mid \; < \ldots < \; \mid u_m \mid$ will be considered. In Figure 1 a basic users structure is presented in CLAT. There are three different levels, which are Experts (E), Medium (M), and Basic (B). Also, in this Figure the values $0, \ldots, 4$ represent tests suites.

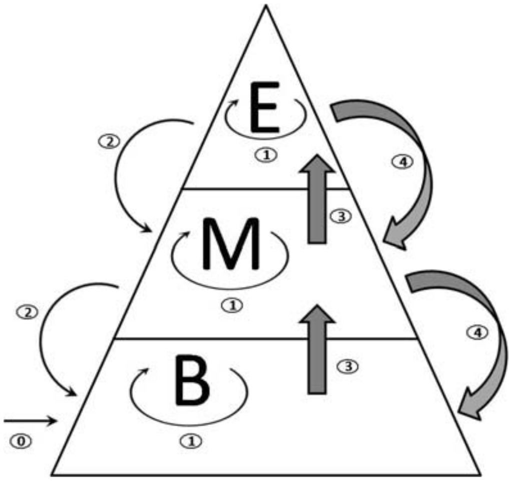


Figure 1. Users structure.

Any new user starts solving tests at point 0, which is represented in the base of the system. If the user passes this initial tests suite, then she will be included into the (B) level.

If users want to continue being in their current level, then they have to perform some tests of this level, that means they are being "updated with the current knowledge" of this level. These tests suites are represented in Figure 1 by number 1, and usually they are proportionated by the other users belonging to this level, or the users of a higher level (denoted by 2). An user can upgrade to a higher level if she performs correctly the tests suite 3, or downgrade if she is unable to answer the tests suite 4 correctly.

Taking into account the user structure presented in Figure 1, then in Figure 2 we present our ideal structure evolution. In this structure is presented how all the users of the system are becoming into experts. At the beginning the most users of the system are in the (B) level, and at the end the most users are in the (E) level. It means, at the end, all new knowledge is spread out into the users of the system.

3 The Validation of a Student ID

In order to ensure a personalize treatment, students access the system through a login page. This allows the system to recover the data from previous sessions. At the beginning of the course, students are provided with a pass-

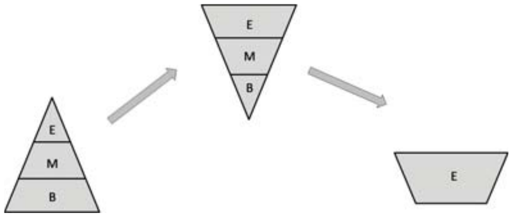


Figure 2. Users structure.

word. They log in by giving their ID-numbers and the password. This mechanism tries to avoid *attacks* to previous sessions of students. For example, an attacker could ask the system for previous exercises and provide wrong answers. Then, when the real student logs in, she will find out that the system thinks that she did not understand the concepts covered in previous sessions.

So, to solve this problem, in `CLAT` we have implemented a novel approach to detect *unexpected behaviours* of the users. Next we present the main steps of this module and next the implementation details. First of all, we have to define what is an unexpected behaviour. Basically, at the beginning of each session the first set of tests that the user answers always contains some tests that she answered in the past. Taking into account that all the interactions of the users and the system are collected in our database, then we can compare what the student answered and what she is answering right now. `CLAT` assumes that student learns the subject, so it expect that some questions that had a *wrong* answer in a first iteration with the student, they are answered correctly answered in a second iteration. However, during the implantation phase of `CLAT`, we detected that there were some situations that happened in `CLAT` that forced us to adapt this validator in order to avoid them. Following we report a situation that show unexpected behaviours. "Some students always use some common expressions in their answers. We detected that some of them always wrongly wrote these expressions (the verb, the subject...). `CLAT` detected a wrong behaviour because of during a session one of this student always wrote correctly the expressions and the session recorded in the following day she continued writing wrongly the expressions. Thus, there was another person answering her tests.".

A typical execution of our this module is following presented. During the rest of the section we will denote by ρ the test that is being compared. The first thing that is extracted from a test is the language, so that we can differentiate, in our case, between English, German, French and Spanish. Next the representation of ρ is analyzed. All the information is extracted from the ρ while *empty words*, that is, words that are not representative of ρ, are removed by using a *stop list* (typically, this list includes words such as "a", "the", "is", "etc", etc). As expected, these lists vary for different languages. Next, ρ is *tokenized* so that the frequency on the test of each relevant term is computed. These terms are calculated by using a *list of inverted terms* (we will deal later with this concept). Given a word i and a document j, the *weight* of the word is computed by using the following expression:

$$wd_{ji} = Frec_{ji} * log_2\left(\frac{NTD}{NDA_i}\right)$$

where $Freq_{ij}$ is the frequency of i in document j, NTD is the total number of documents on the case database for this user and NDA_i is the number of documents that contain the word w. In Table 1 we show how the relevant information contained in each test is recorded by using *events* (contained all the information) and *cases* (containing the information relevant for further processing).

Once all the weights for ρ have been computed, the similarity between this and other documents classified in the case database is studied. For the classification of cases, `CLAT` is based on a list of inverted terms to store all the terms that appear in previous cases as well as a list containing the document numbers that have that word among its list

`Attribute`	`Description`
Subject	The name of the subject
Date	Date of dispatch
Origin	The IP number and the operative system
Message	The answers for the test
Error	Indicates whether it is an unexpected behaviour or not
Terms	Terms extracted from the Message and the Subject
Weights	Weights associated with each term of this test
Document number	Document number in the case database
Language	Language used in this test

`Attribute`	`Description`
Terms	Terms extracted from the message and the subject
Weights	Weights associated with each term of the test
Document number	Document number in the case database

Table 1. Structure of an event and of a test.

`Attribute`	`Description`
Universe's speech	Total number of terms in the list
Number of Documents	Total number of documents in the case database
List of terms	List of inverted terms itself

`Attribute`	`Description`
Total number of occurrences	Total number of occurrences of the word among all documents
Vector of occurrences	Vector with the number of occurrences of the word in each document

Table 2. Structure of the list of inverted terms and of its elements.

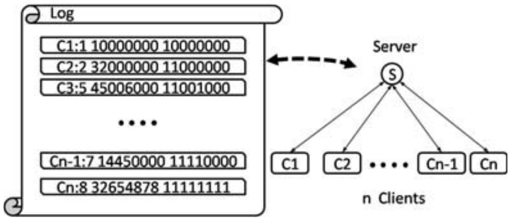


Figure 3. Global Log.

of terms. The list of inverted terms has been implemented with a hash table, having as keys in this table are the new terms. The values of the table are elements with the structure defined in the bottom of Table 2. Finally, the similarity between a document of the database j and ρ is computed as

$$sim_{j\rho} = \frac{\sum_{i=1}^{m} wd_{ji} * wd_{\rho i}}{\sqrt{\sum_{i=1}^{m} wd_{ji}^2 * \sum_{i=1}^{m} wd_{\rho i}^2}}$$

where wd_{xy} is computed as stated before, denoting the weight of word y in document x, and m is the number of weight vectors.

Once we have all the similarities, we choose the three documents providing the highest similarity with ρ. If the highest one is greater than 65% of the addition of the other two, then we take the result corresponding to the highest. Otherwise, we make a *majority voting* among these three cases. Once ρ is processed, and a verdict regarding whether it contains an unexpected behaviour has been reached, then ρ is added to the case database so that it can be used for analyzing forthcoming tests. Let us remark that with this module, CLAT allows us to detect the unexpected behaviours of the students.

4 Security and statistical measures on CLAT

In order to guarantee the fault absence, a methodology of performing testing security task is provided in CLAT. There are several ways to classify testing techniques. In this work, there are used passive testing techniques, which are non-intrusives with respect to the system under test, and the security properties can be represented by using formal syntax. Passive testing do not use any set of test in order to provide a verdict of how good a system is. This technique uses the log of the system in order to recognize erroneous behaviors. We have detected within our architecture several problems. When users are interacting with the system, all actions performed from them are stored in a global log, and the measure of this log is huge.

The global log structure presented in CLATis presented In Figure 3. There, all messages sent from users are recorded into a file. We can express security properties in order to check the correctness of the log. For example, let us consider the following property: "CLAT implements a FIFO user policy". That is, if the user C_1 asks for a test before $C_2, C_3, \ldots$ then, the server will provide firstly to C_1 the test to be solved. According to the global log presented in Figure 3, C_1 asks for `Do Test` before C_2, so the server will answer with `Send Test` to C_1 before to C_2. Let us note, than before receiving C_2 its notification, a new user, by means C_n, asks for a new test. Then, according to the previuos rule, the answer to C_2 should obtain the test before than C_n.

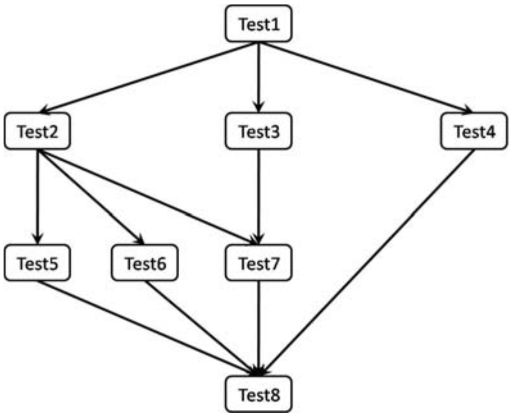


Figure 4. Tests Structures.

Let us consider another property presented in CLAT. As it was mention, CLAT provides a methodology to perform different tests in different topics. Sometimes it is necessary that previous knowledge should be learned before advancing in tests. Let us consider the test structure presented in Figure 4. This structure represents that the initial test to be performed by an user is `Test 0`, and performing this she will be able to answer or `Test 1`, or `Test 2`, or `Test 3`. The arrows from `Test i` imply which tests are *opened* after correctly performs the test `i`. By using passive testing techniques, we are allowed to express properties such as: "At the beginning only `Test 0` can be performed", or "If the `Test 6` is performed then previously `Test 1` and `Test 2` were correctly performed".

The other main features on CLAT is possibility to to get, in an easy way, statistical information of the users. This information is used to provide a good classification of students. This information should be able at the end of each day, and it is extracted from the interactions log of the users recorded in one day. Each day the information has to be mixed with previous information, and a new user hierarchy is provided.

There are three problems when we perform these tasks in the structure presented in Section 2. The first problem

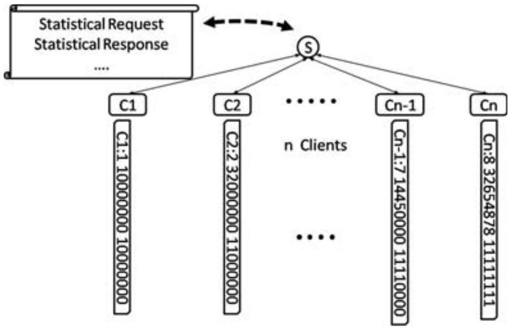


Figure 5. New Structure.

is the huge *size of the log* presented in the server. Let us consider that the amount of users, with respect to time that are connected is denoted by $N(t)$, and the average of the amount of actions of any user with the system is K. Then we have that an usual size of log between two timed values t_1 and t_2 is represented as:

$$\int_{t_1}^{t_2} N(t)\,dt \cdot K \tag{1}$$

The second problem is relative to performed *measures*. When we perform p different properties by using passive testing then we have that the complexity of performing this task is presented as:

$$p \cdot \int_{t_1}^{t_2} N(t)\,dt \cdot K \tag{2}$$

The third problem is relative to perform the *statistical information*. In order to provide m statistical information, we have:

$$m \cdot \int_{t_1}^{t_2} N(t)\,dt \cdot K \tag{3}$$

Let us note that in order to increase the performance of CLAT, we should reduce the values of equations 1. 2 and 3.

5 CLAT in a collective user environment

In this section we present how to reduce the previous problems in CLAT to work in a collective user environment. First, we focus on the size of the log. The previous approach assumes to have a big log, where all user interactions with the server are recorded. In this update, the software performed by users increase the functionality, in order to reduce the memory used in the server. All client software, will be able to store all interactions of the incoming user, and after will process this information, and will send a resume to the server. This idea is presented in Figure 5. When an user finishes interacting with the system, then the passive testing techniques will be used with the local stored data, and will send this information to the server. Let us note, that if we only check security properties in alone nodes then the size of the server for storing the log is null, and the size of the users is K (Similar value than Equation 1).

But, not all properties can be checked in users software. For example, the FIFO property must be checked, the information recorded in users is not enough. The information regarding when the user C_i and the user C_j connect to the system is only provided in the server. Let us assume that we have p properties, where q, with $q \leq p$ are properties that only can be checked in the server. Let K be the average of actions performed by users (see Equation 1), and let K', with $K' \leq K$ be the actions that have to be recorded in order to check the q properties. Then, the amount of data recorded in server are:

$$\int_{t_1}^{t_2} N(t)\,dt \cdot K' \tag{4}$$

Let us note, that the log size in users have not be reduced, because very often the properties $p-q$ that matches the local log need to check the complete interaction log.

The second problem that we focus in this update is the time to check all properties. Let us note that in previous approach, we have a big log, recorded in the server, and all interaction were recorded on it. The time associated to check the properties were presented in Equation 2. By using this new architecture paradigm, we reduce the computation. The passive testing tasks are performed now not only in the server, but also in the n users. The task of performing in users can be done at the same time, because they are in different machines. So that, the time associated with this task is:

$$q \cdot \int_{t_1}^{t_2} (N(t)\,dt \cdot K') + (p-q) \cdot K \tag{5}$$

The last problem is the time to check the statistical properties. As we have computed previously in Equation 3 this task continues being very cost. However, this property only uses isolate user information. That is, this task can be performed with all guarantee in the user clients. So, the server only has to be able to record the results generated by each user clients, at the end of all interactions. The new equation to compute the m statistical information is:

$$m \cdot K \tag{6}$$

This cost is computed in each client, and it does not perform any disadvantage in the server. Next, all information about the statistics of the user are sent to the server. Let us remark another feature presented in this update. In Figure 5 is presented two calls to the server: `Statistical request` and `Statistical response`. This input/output action respectively are used in order to compute two stages. The first stage is focused on classifying users into system; and the second one is used to create/modify/update the tree structure of the topics.

6 Conclusions and Future work

In this work we have presented our tool CLAT. This tool has been developed focusing on solving the problems of testing task and synthesizing statistic information. Both tasks are traditionally performed in the server. We suggest to increase the power of client software, in order to reduce the computational time presented in the server. Furthermore, we have proved that by using the CLAT architecture, the amount of interactions of the users that has to be mixed in logs, in order to perform the testing task, is reduced. In addition we have presented a novel module, based on Bayesian text filterd to identify the ID of the users.

It is worth to point out that a very important part of any tutoring system is the feedback from the users. While designing our system, we have been specially careful at this point. For instance, let us consider the answer of our system after a test is made. If the student provides the right answer then the system returns a congratulations message. The difficulties start when managing wrong answers. The easy solution consists in notifying that the answer was wrong and provide the right answer. In this scenario, the student will try to understand what she was doing wrong by pattern matching. We consider that this is *not* the best practice. We have preferred to return a suggestion about what the student should do (indicating what the error was) instead of giving the right answer.

We have also paid special attention to avoid cheating. As it is pointed out for example in [5], some students tend to learn how to cheat the system instead of learning the current contents. We do not claim that our system is totally fool-proof (actually, we do not think so!) but we have tried to detect some *funny* answers. For instance, if we ask for the value of 3+4 a student may answer 5+2 (non so trivial examples include the application of higher order functions in an unexpected way). Actually, this is a right answer, but it is not what it is expected. If CLAT detects such a *right* answer, it will indicate that it is correct but it will ask for the *most correct* answer. Finally, even though the management of answers has been a specially hard part to develop, we think that the effort has been worth. Firstly, students will see their mistakes and try to correct them. Secondly, they will be soon convinced (we hope) that it is senseless to spend time trying to fool the system.

Students will be also allowed to ask for hints. The type of hints, that the system provides, depends on the number of hints the student has already asked for in the current exercise. For instance, if the student has provided a wrong answer, a first hint will only provide a message saying which type of error it was, that is, whether it was a syntactic error, a type error, or a semantic error. Afterwards, in case the student needs more hints, the error will be explained more precisely. Finally, if a student is not able to provide the right answer, she can press the *give up* button and the answer will be presented.

Thus, as future work, we would like on the one hand to include more intuitive answers in each test, and on the other hand to implement new situations to detect unexpected bahaviours, such as using probabilities and stochastic information.

References

[1] Z. Akyol and D.R. Garrison. The Development of a Community of Inquiry over Time in an Online Course: Understanding the Progression and Integration of Social, Cognitive and Teaching Presence. *Journal of Asynchronous Learning Networks*, page 20, 2008.

[2] S. Auer and J. Lehmann. What have Innsbruck and Leipzig in common? extracting semantics from Wiki Content. In *4th European conference on The Semantic Web, ESWC '07*, pages 503–517. Springer, 2007.

[3] S. Overell, B. Sigurbjörnsson, and R. van Zwol. Classifying tags using open content resources. In *2nd ACM International Conference on Web Search and Data Mining, WSDM '09*, pages 64–73. ACM, 2009.

[4] R.M. Palloff and K. Pratt. *Collaborating online: Learning together in community*. Jossey-Bass San Francisco, 2005.

[5] R. Schank and A. Neaman. Smart machines in education. chapter Motivation and failure in educational simulation design, pages 37–69. MIT Press, 2001.

[6] M.K. Tallent-Runnels, J.A. Thomas, W.Y. Lan, S. Cooper, T.C. Ahern, S.M. Shaw, and X. Liu. Teaching courses online: A review of the research. *Review of Educational Research*, 76(1):93, 2006.

[7] H. Tsoukas and E. Vladimirou. What is organizational knowledge. *Complex knowledge: Studies in organizational epistemology*, pages 117–140, 2005.

A proposal for the improvement of the technique of Earned Value Management utilizing the history of performance data

Adler Diniz de Souza
Universidade Federal do Rio de Janeiro
Program of Systems Engineering and Computing
Av. Horácio Macedo, 2030, Building of the Technical Center, Block H, Suite 319, P.O. Box 68511 – Postal Code 21941-914 – Rio de Janeiro, RJ
adlerunifei@gmail.com

Ana Regina Cavalcanti Rocha
Universidade Federal do Rio de Janeiro
Program of Systems Engineering and Computing
Av. Horácio Macedo, 2030, Building of the Technical Center, Block H, Suite 319, P.O. Box 68511 – Postal Code 21941-914 – Rio de Janeiro, RJ
darocha@centroin.com.br

Abstract - **Although the technique of Earned Value Management - EVM is utilized by several companies in different sectors (software development, civil construction, aerospace, aeronautics, among others) for over 35 years, in order to predict time and cost results, many studies such as [13], [24], detected vulnerabilities in the technique, among them: i) the cost performance data did not always show a normal distribution, which makes it hard to obtain reliable projections; (ii) there is instability in the cost and time performance indicators during the Project ([13], [24]); (iii) there is a trend of deterioration in the cost and time indicators when the projects are near their end ([13]), and others. The present study proposes an extension of this technique, through the integration of the history of performance data as means of improving the technique's cost predicability. The proposed technique is evaluated and compared to the traditional technique through different hypothesis tests, utilizing data from the simulation or real projects. The tecnique showed to be more accurate and precise than the traditional one for the calculation of the Cost Performance Index - CPI and the Estimate At Completion - EAC.**

Keywords: Earned Value Management, Cost Peformance Index – CPI, project management, measurement and analisys

I. Introduction

The PMI currently estimates that approximately 25% of global GNP is spent on projects and that close to 16.5 million professionals are directly involved in project management throughout the world. This volume of projects and changes in an increasingly competitive global scenario, generates the need of faster results, with higher quality, lower costs and shorter deadlines [4].

Therefore, the success of the projects can only be obtained by meeting customer requirements, within the objectives of time, cost and quality defined for the project [21].

To assess whether or not a project will reach its goals of time and cost, several measures are collected during its execution, and various performance indicators are produced and periodically analyzed. When there are deviations larger than the tolerance in some performance indicators, corrective actions are undertaken in order to improve them. Among the main available techniques for the analysis of cost and time - EVM (Earned Value Management), is considered the most reliable [14].

EVM is a technique that integrates scope, time and cost data to measure project performance and predict its cost and deadline, based on the current performance of the team. The technique gained great importance when, in 1967, the United States Department of Defense - DoD, began requiring its use as a means to control the costs of contracted projects [23].

Several formulas derived from EVM measurements are available and have been studied in the last 15 years. However, studies intended to improve the predictability of the results of time and cost have remained stagnant over the last decade and still require further studies [14].

Thus, the research question of this study is the possibility of evolution of the technique of analysis of earned value management, allowing the integration of quantitative information of the subprocesses most relevant to the business goals related to time and cost.

II. Earned Value

The method of earned value management allows the calculation of variances and performance indices of cost and time, which generate forecasts for the project, given its performance so far, allowing the implementation of actions aimed at correcting any deviations [2]. This allows the project's manager and your team to adjust their strategies, make trade-offs based on the goals, on the project's the current performance, on trends, and on the environment in which the project is being conducted [2].

According to [18], EVM has an essential role in the success of projects, responding to managerial issues that are considered critical, such as: i) how efficiently are we using our time? ii) when is the project likely to be finalized? iii) how efficiently are we using our resources ? iv) how much above or below the budget will we be at the end of the project, given the current productivity of the team? The method of analysis of earned value management is based on three basic measures, which are derived to generate other measures and performance indicators. These basic measures are: i) Planned Value - PV: represents planned costs accrued up to a given date; ii) Earned Value - EV: represents the budgeted cost of the work performed up to a given date; and iii) Actual Cost - AC: represents the actual cost of work performed [18].

The basic measures discussed do not allow making predictions of cost and time to complete the project, and answer the questions posed above. For this purpose it is necessary to generate performance indicators, among which the most widely used are the Schedule Performance Index - SPI and the Cost Performance Index - CPI.

The SPI is an indicator of actual progress compared to planned progress of a project [19]. It shows how efficiently the project team is using its time [2], calculated as:

$$SPI = EV / PV \qquad (1)$$

The CPI is a measure of the value of work performed compared to the actual cost or progress made in the project [19]. It shows how efficiently the project team is using its resources [18], calculated as:

$$CPI = EV / AC \qquad (2)$$

The CPI is considered the EVM's most critical indicator, because it measures the cost efficiency of work performed [19].

As the project progresses, the project team can develop a forecast for the Estimate At Completion - EAC, which may differ from the Budget At Completion - BAC, based on project performance [19]. The EAC provides the final estimate of cost and is given by the equation below (assuming that the cost performance remains the same):

$$EAC = BAC / CPI \qquad (3)$$

Figure 1 illustrates the measures and indicators discussed, as well as projections that can be made from the indicators presented, and Table 1 illustrates the other components of the technique of earned value management that were not discussed and/or that will not be used in the context of this study, but that are important for your understanding.

TABLE 1 – OTHER EQUATIONS AND DEFINITIONS OF THE EVM TECHNIQUE

Equation	Definition
BAC = PV∑	Budget At Completion, is the baseline of project cost.
SAC or PAC	Shedule At Completion ou Plan At Completion, is the deadline of project completion.
TAC = PAC / SPI	Time At Completion is the projected deadline, calculated from the SPI.
EAC = AC + BAC – EV	assuming that the CPI will be equal to 1
EAC = AC + [(BAC – VA) / (CPIcumulative x SPIcumulative)]	when one wishes to consider both CPI and SPI indicators

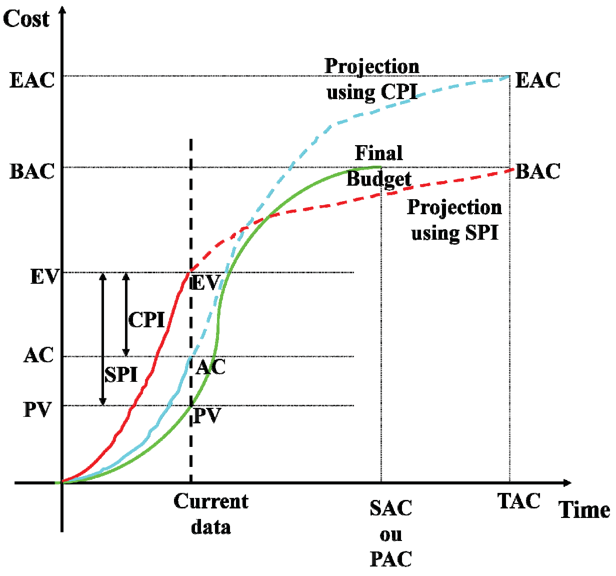


Figure 1 – Performance measures and indicators

III. PROBLEM DESCRIPTION

The Earned Value Management technique uses the CPI to make cost projections at the end of the project. The application and reliability of this indicator for the realization of projections have been widely discussed as shown by the works of [2], [24], [14] and [7].

The discussions focus on the CPI's stability. The statement of the CPI's stability is important, as it is used to make cost projections (EAC = BAC / CPI). According to [2], the CPI is considered stable if there is a variation of plus or minus 10% of the value reached when the project has been 20% executed, i.e., if during a project, its CPI is equal to 0.87, when the project's execution is at least 20%, it is expected that it will not vary more than some value between 0.77 and 0.97.

According to [3], stability can be defined as a state of statistical control that provides with a high degree of confidence, the forecast performance of some variable in the immediate future.

Florac [5] states that the stability of a process is considered by many as the core of process management, and that it is essential for organizations to produce products according to plan and to improve processes so as to produce better and more competitive products.

A study performed in [2] evaluated the stability of the CPI of various projects of the United States Department of Defense (DoD), and found that there was stability of the indicator, after 20% of project execution. This study generalized the result, stating that any project could use the technique reliably after 20% of execution. This information was used as a criterion for keeping or canceling U.S. government projects which had an CPI lower than 0.9% after 20% of execution, because according to the study, the stability of the indicator was evidence that a project with a bad CPI was unrecoverable.

However, several other studies have questioned the generalization of these results in different contexts (projects developed outside the DoD's scope) that showed different results, i.e., the cost performance indicators showed instability during much of the project [24], [14], [7].

To state that the CPI is unstable and varies widely during project execution prevents accurate cost projections at project end (EAC), unless this variation is known or can be expected, as a result of already known factors.

The proposal of evolution of the Earned Value Management technique presented in the next section, is based on the premise that any project consists of a set of processes which have different performance or variability levels for the cost performance indicator. This premise can be confirmed by several studies reported in [19], [17] and [6].

Thus, one explanation for the CPI's instability is the natural variation of the performance of the processes used, according to figure 2, which shows that the CPI $_{\text{Cumulative}}$ is actually a composite of the performance of several processes, therefore the expectation is not that the CPI $_{\text{Cumulative}}$ will be constant or equal to 1, but that it will vary as a result of the execution of each specific process. Knowing this, it is possible to write an equation that takes into account the history of data performance, increasing the predictability of cost, even in projects that have an unstable CPI.

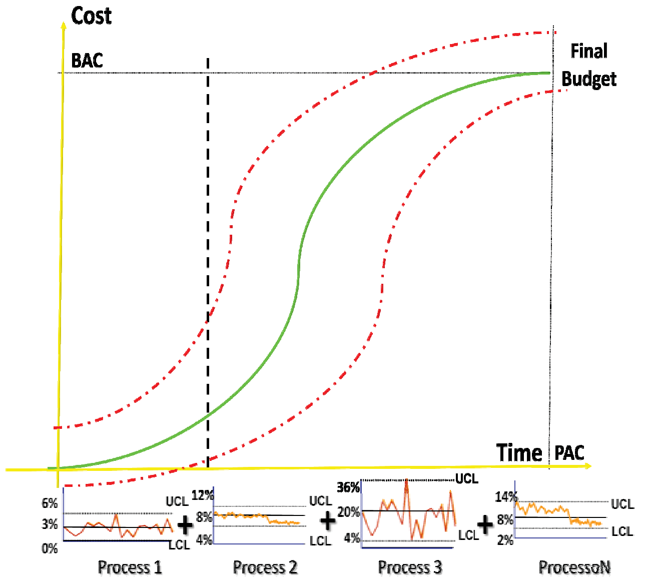


Figure 2 – Justification for the CPI's instability

IV. Proposal of EVM History

If it becomes obvious that the BAC is no longer feasible, the project manager should develop a forecast EAC. The development of an EAC forecast, involves the search of alternatives or prognosys of future conditions and events for the project, based on information and knowledge available at the time when the forecast is made. The information on the performance of work include the project's past performance and any information that may impact it in the future [19].

The history of cost data of the processes can impact the future evolution of the cost performance indicator, but this history, however, is not used to make these projections.

The present study proposes the integration of the EVM with the history of cost performance data, which consists in the collection and utilization of a set of data of each one of the processes, such as: i) distribution of process data; ii) history of process performance; and iii) weight (% of effort or cost) of each process utilized in the project.

The distribution of process data is considered important for the composition of the new proposal, because it defines which statistic indicates the most likely component of a sample. In a normal distribution, for example, the average indicates the sample's most common element, while in an asymmetrical distribution shifted to the right or left, the median is most common that average. The history of performance of the processes is important because it will be utilized to forecast the future behavior of the project's CPI, which is made up by the individual performance of each process. Since the processes have distinct percentages of effort and cost allocated to them, with a directly proportional influence on the composition of the performance indicators, it was decided to assign a weight of zero to one hundred to indicate the weight of each process.

Thus, given a symmetric distribution (normal distribution) or asymmetric (shifted to right or left), given its performance and the weight of the process, it is possible to use an average or weighted median (when dealing with a process with asymmetric distribution) that represents the CPI projected at project end, as in the equation below:

$$CPI_{Est} = (CPI_{AcumN\%}*Weight_{Acum} + CPI_{P1}*Weight_{P1} + CPI_{HistP2}*Weight\ _{P2} + CPI_{PN}*Weight_{PN}...\ ) \qquad (4)$$

Where:

- **CPI$_{\mathbf{Est}}$:** is the estimated CPI that will be used for projections;
- **CPI$_{\mathbf{AcumN\%}}$:** is the CPI that will be calculated normally, with the traditional equation of the EVA (EV / AC), until the project's current date;
- **Weight$_{\mathbf{Acum}}$:** is the percentage of the project already executed expressed in %. Assuming a project with a final cost estimated at $100,000.00, if the sum of the activities of a given process X within this project, is equal to $10,000.00, the weight of this process is 10%;
- **Other CPI's and Weights:** are the history of CPI's (average or median) of each process utilized in the project and how much the weight of these processes represent in the total cost of the current project.

V. Planning of the Study

The study's objective was to answer the following question: "*Is the Earned Value Management technique with the history of performance more accurate and precise than the traditional Earned Value Management technique?*" Thus, the following hypotheses were set up to evaluate the accuracy of the techniques:

- **H0$_{Accuracy}$**: the traditional Earned Value Management technique is as accurate as the Earned Value Management technique with the history of performance.
  - ○ ($Error_{EVM} – Error_{EVMH.} = 0$).
- **H1$_{Accuracy}$**: the traditional Earned Value Management technique is less accurate than the Earned Value Management technique with the history of performance.
  - ○ ($Error_{EVM} – Error_{EVMH.} > 0$)

A similar hypothesis was identified and tested to assess the precision of the techniques.

Three more questions and secondary hypotheses, similar to the first one, were defined, but they were intended to answer if the proposed technique represented more accurately and precisely the traditional technique at the begining (25% executed), at the middle (50% executed), or near the end (75% executed) of a project.

The techniques presented in section IV were evaluated through a feasibility study, in which the objective was to measure the precision and accuracy of both techniques and compare them. According to the [19]precision and accuracy are not the same thing. Precision means that repeated measurement values are grouped and show little dispersion. Accuracy means that the measured value is quite close to the right value. Both measures are not necessarily exact, and an exact measure is not necessarily precise.

Thus, in order to measure the technique's accuracy, each technique's CPI was compared at three distinct times in the project, at the begining, after 25% of execution ($CPI_{25\%}$), ii) at the middle, after 50% of execution ($CPI_{50\%}$) and iii) at the end, after 75% of execution ($CPI_{75\%}$), with the real CPI calculated at the end of the project. The error shown by each technique was assessed in relation to the final calculated CPI, at these three moments. These errors were stored for the performance of statistical tests.

To measure the techniques' precision, the variability of the techniques was compared to the last estimate, i.e., how much an estimate varied in relation to the previous one, at each of the moments when it was measured.

One of the main difficulties presented by [16] and [7] in studies related to this one, was the lack of project performance data, available for studies.

Thus, it was decided to validate the proposed technique through the performance of project simulations, similarly to the studies conducted by [9], [8] and [12].

Microsoft Excel was used to generate and store effort data and consequently cost (random() function). Microsoft Visual Basic was used to create macros that generated different simulations and compared them, while the RExcel and Action tools were used for statistical analysis.

VI. FEASIBILITY STUDY

The largest cost component in a software project are the man-hours necessary for product development, all the necessary simulations required for the calculation of the base measures and indicators of traditional Earned Value Management were based on the planned effort and actual effort for a set of activities of possible processes of any given project, whereby these activities were calculated using the random() function of the MSExcel tool.

The initial simulations have 4 processes with the variation in the random function shown by Table 2.

TABLE 2 – VALUES PASSED TO THE RANDOM FUNCTION FOR THE INITIAL SIMULATION

Process	Variation H_{Est}	Variation H_{Real}	No.of activities
Process 01	8 – 30	3 – 40	12
Process 02	3 – 10	3 – 10	26
Process 03	3 – 12	1 – 12	26
Process 04	3 – 17	1 – 25	26

It was assumed, as premises for the generation of the CPI of the proposed technique that: i) all the processes which were in execution were known; ii) the history data of all previously executed projects utilizing this process were available and iii) the processes were stable, however they showed great variability, or a high standard deviation (standard deviation of 0.4 or higher, reaching 1.9 in some cases). The choice of great variability would indicate the worst case, or the most difficult scenario for the forecast of the final CPI, since this case would show an unstable behavior throughout the project.

The execution of 10 projects was then simulated, with collection of their measurements and calculation of the indicators.

The error and variability errors among the measurements of the 10 evaluated projects were grouped in a single tab of the Excel spreadsheet, allowing the performance of statistical analyses.

The simulations showed that the proposed technique showed better accuracy and precision of cost estimates than the traditional technique, as shown by Figure 3 and Figure 4.

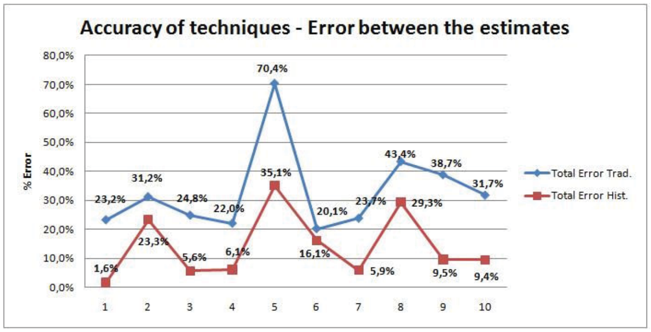
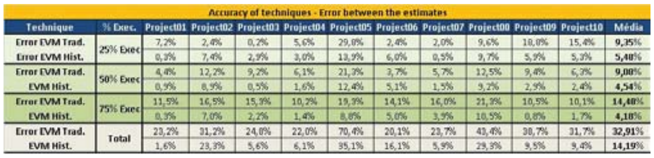


Figure 3 – Accuracy – Total CPI error between the techniques

Figure 3 shows on the X axis each one of the assessed projects, and on the Y axis, the sum of the errors of $CPI_{25\%}$, $CPI_{50\%}$ and $CPI_{75\%}$ for each one of the projects. The gain in accuracy using the proposed technique was at times 15 times better than the traditional technique, which suggests that the proposed technique was 15 times more accurate than the traditional technique, in the context of the simulation. The minimum observed gain was 25% of accuracy, in project 06, as can also be seen in Figure 3.

Statistical tests were performed based on Table 3 and Table 4 to confirm whether the differences in accuracy and precision found in the application of the techniques were significant, thereby approving or rejecting the previously presented hypotheses. The Action tool was used to test the hypotheses of T paired samples, at the 99% significance level.

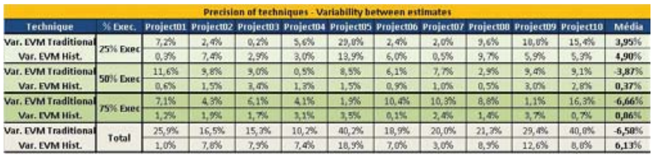
TABLE 3 – ACCURACY (ERROR BETWEEN THE ESTIMATES) OF THE CPI



Accuracy of techniques - Error between the estimates

Technique	% Exec.	Project01	Project02	Project03	Project04	Project05	Project06	Project07	Project08	Project09	Project10	Média
Error EVM Trad.	25% Exec	7,2%	2,4%	0,2%	5,6%	29,0%	2,4%	2,0%	9,6%	10,0%	15,4%	9,35%
Error EVM Hist.		0,3%	7,4%	2,9%	3,0%	13,9%	6,0%	0,5%	9,7%	5,9%	5,3%	5,40%
Error EVM Trad.	50% Exec	4,4%	12,2%	9,2%	6,1%	21,3%	3,7%	5,7%	12,5%	9,4%	6,3%	9,08%
EVM Hist.		0,9%	8,9%	0,5%	1,6%	12,4%	5,1%	1,5%	9,2%	2,9%	2,4%	4,54%
Error EVM Trad.	75% Exec	11,5%	16,5%	15,9%	10,2%	19,9%	14,1%	16,0%	21,9%	10,5%	10,1%	14,48%
EVM Hist.		0,3%	7,0%	2,2%	1,4%	8,8%	5,0%	3,9%	10,5%	0,8%	1,7%	4,18%
Error EVM Trad.	Total	23,2%	31,2%	24,0%	22,0%	70,4%	20,1%	23,7%	43,4%	38,7%	31,7%	32,91%
EVM Hist.		1,6%	23,3%	5,6%	6,1%	35,1%	16,1%	5,9%	29,3%	9,5%	9,4%	14,19%

TABLE 4 – PRECISION (VARIABILITY BETWEEN THE ESTIMATES) OF THE CPI



Precision of techniques - Variability between estimates

Technique	% Exec.	Project01	Project02	Project03	Project04	Project05	Project06	Project07	Project08	Project09	Project10	Média
Var. EVM Traditional	25% Exec	7,2%	2,4%	0,2%	5,6%	29,0%	2,4%	2,0%	9,6%	10,0%	15,4%	3,95%
Var. EVM Hist.		0,3%	7,4%	2,9%	3,0%	13,9%	6,0%	0,5%	9,7%	5,9%	5,3%	4,90%
Var. EVM Traditional	50% Exec	11,6%	9,8%	9,0%	0,5%	8,5%	6,1%	7,7%	2,9%	9,4%	9,1%	-3,87%
Var. EVM Hist.		0,6%	1,5%	3,4%	1,3%	1,5%	0,9%	1,0%	0,5%	3,0%	2,8%	0,37%
Var. EVM Traditional	75% Exec	7,1%	4,3%	6,1%	4,1%	1,9%	10,4%	10,3%	8,8%	1,1%	16,3%	-6,66%
Var. EVM Hist.		1,2%	1,9%	1,7%	3,1%	3,5%	0,1%	2,4%	1,4%	3,7%	0,7%	0,06%
Var. EVM Traditional	Total	25,9%	16,5%	15,3%	10,2%	40,2%	18,9%	20,0%	21,3%	29,4%	40,0%	-6,58%
Var. EVM Hist.		1,0%	7,8%	7,9%	7,4%	18,9%	7,0%	3,0%	8,9%	12,6%	8,8%	6,13%

The analysis of the data in Table 3, allows to state, at the outset, that the proposed technique provides more accuracy in the cost estimates, when the project is at 50% and 75% of execution, and in general taking the three reported moments, at a 99% confidence level. Even though the results presented in Table 5 show that the proposed technique shows an average error lower than the traditional technique, when the project was at 25% of execution, this result is not significant, according to the hypothesis test performed.

TABLE 5 - TESTS OF HYPOTHESIS OF ACCURACY

Hypothesis	Test	T	p	Conclusion
$\text{H0}_{\text{Accuracy}}$	$\text{Error}_{\text{EVM}} - \text{Error}_{\text{EVMH.D.}} > 0$	3,59	0,00292	Refute H0
$\text{H0}_{25\%\ \text{Accuracy}}$	$\text{Error}_{\text{EVM25\%}} - \text{Error}_{\text{EVMH.D.25\%}} > 0$	2,21	0,0269	Accept H0
$\text{H0}_{25\%\ \text{Accuracy}}$	$\text{Error}_{\text{EVM50\%}} - \text{Error}_{\text{EVMH.D.50\%}} > 0$	4,809	0,00048	Refute H0
$\text{H0}_{75\%\ \text{Accuracy}}$	$\text{Error}_{\text{EVM75\%}} - \text{Error}_{\text{EVMH.D.75\%}} > 0$	21,271	2.63×10^{-9}	Refute H0

On the other hand, the gains in accuracy shown in Figure 4 were up to 8 times superior to those of the traditional technique, which suggests that the proposed technique is 8 times more precise than the traditional one. In Figure 4, the values shown in axis Y represent the sum of the variations among the estimates at $\text{CPI}_{25\%}$, $\text{CPI}_{50\%}$ and $\text{CPI}_{75\%}$.

The analysis of the data in Table 4 and Table 6 also allows to state, that the proposed technique provides more accuracy in the cost estimates, when the project is at 50% and 75% of execution, and in general, taking the three reported moments, at a 99% confidence level. Even though the results presented in Table 4 also show that the proposed technique shows an average error lower than the traditional technique, when the project was at 25% of execution, this result is not significant, according to the hypothesis test performed.

TABLE 6 - TESTS OF HYPOTHESIS OF PRECISION

Hypothesis	Test	T	p	Conclusion
$\text{H0}_{\text{Precision}}$	$\text{Var}_{\text{EVM}} - \text{Var}_{\text{EVMH.D.}} > 0$	5,602	0,00016	Refute H0
$\text{H0}_{25\%\ \text{Prec}}$	$\text{Var}_{\text{EVM25\%}} - \text{Var}_{\text{EVMH.D.25\%}} > 0$	1,684	0,0632	Accept H0
$\text{H0}_{50\%\ \text{Prec}}$	$\text{Var}_{\text{EVM50\%}} - \text{Var}_{\text{EVMH.D.50\%}} > 0$	5,736	0,00014	Refute H0
$\text{H0}_{75\%\ \text{Prec}}$	$\text{Var}_{\text{EVM75\%}} - \text{Var}_{\text{EVMH.D.75\%}} > 0$	2,882	0,009	Refute H0

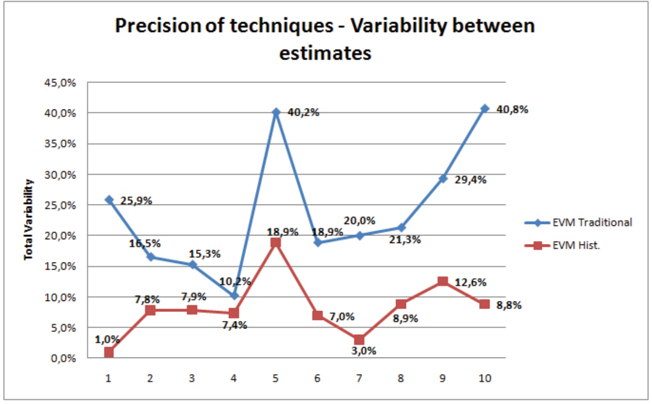


Figure 4 - Precision - Total variability of the CPI between the techniques

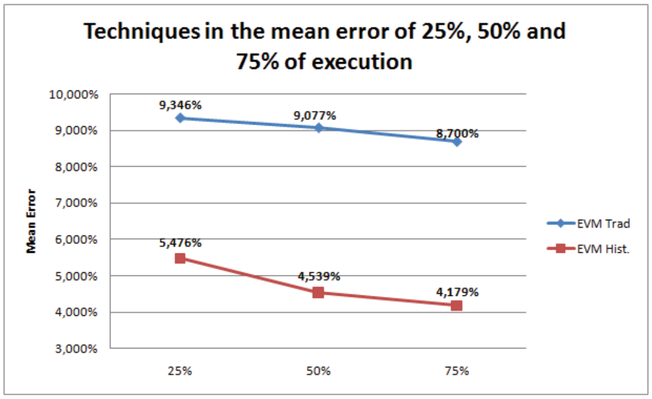


Figure 5 - Average error at the project's start, middle and end ($\text{CPI}_{25\%}$, $\text{CPI}_{50\%}$ e $\text{CPI}_{75\%}$)

Lastly, Figure 5 shows that both techniques show an increase in accuracy and precision when the project advances in its execution, as reported by several authors like [16], [7]. However, the proposed technique showed better accuracy and precision at project start, middle and end.

VII. CONCLUSION

This study described the proposal of a technique of Earned Value Management, which integrates performance data history as a way to improve the predictability of project costs. The study consisted of a feasibility study based on simulated project data with the purpose of determining whether the technique was more accurate and precise compared to the traditional technique, at the begining (25% of execution), at the middle

(50% of execution) and at the end (75% of execution) of the project. In order to evaluate the feasibility of the proposed techniques, several tests of hypotheses were performed about the different research questions posed during the validation of the techniques.

The simulation assumed as premise for the utilization of the EVM_{Hist} technique, that the project had a data history of all processes of the utilized life cycle, with the result that the proposed technique showed more precision and more accuracy than the traditional technique at the project's start, middle and end. All the tests of hypotheses showed that the results were significant at the 99% significance level.

VIII. References

[1] ANBARI, F.T., 2003, "Earned Value Project Management Method and Extensions", Project Management Journal, v. 4 (Dec 2003), pp. 12.

[2] CHRISTENSEN, D., SCOTT R. HEISE, 1993, "Cost Performance Index Stability", National Contract Management Journal, v. 25, pp. 7-15.

[3] DEMING, W.E., 1993, The New Economics for Industry, Government, Education Cambridge Mass.: Massachusetts Institute of Technology, Center for Advanced Engineering.

[4] DINSMORE, C.E.C., 2003, Como se Tornar um Profissional em Gerenciamento de Projetos: Livro-Base de "Preparação para Cerfiticação PMP - Project Management Professional Rio de Janeiro Qualitymark. (How to Become a Professional in Project Management: Base Manual for "Preparation for PMP Certification")

[5] FLORAC, W.A., A. D. CARLETON, 1999, Measuring the Software Process: Statistical Process Control for Software Process Improvement, Addison-.Wesley.

[6] FLORAC, W.A.R.E.P., ANITA D. CARLETON, 1997, Practical Software Measurement: Measuring for Process Management and Improvement Pittsburgh, Software Engineering Institute - SEI Joint Program Office.

[7] HENDERSON, K., OFER ZWIKAEL, 2008, "Does Project Performance Stability Exist A Re-examination of CPI and Evaluation of SPI(t) Stability, " Cross Talk, v. 21 (April, 2008), pp. 04-07.

[8] IMAN, A., OW SIEW HOCK, 2009, "Implementation and Evaluation of Earned Value Index to Achieve an Accurate Project Time and Cost Estimation and Improve "Earned Value Management System"", International Conference on Information Management and Engineering.

[9] IMAN, A., OW SIEW HOCK, 2009, "Using Enhancement Method to Improve Earned Value Index to Achieve an Accurate Project Time and Cost Estimation", International Conference on Future Computer and Communication.

[10] IRANMANESH, H., N. MOJIR, S. KIMIAGARI 2007, "A new formula to "Estimate At Completion" of a Project's time to improve"Earned value management system", International Journal of Project Management.

[11] LIPKE, W., 2003, "Schedule is different", The Measurable News, v. March & Summer 2003 (March 2003).

[12] LIPKE, W., 2004, "Connecting Earned Value to the Schedule", The Measurable News, v. Winter 2004 (Winter 2004).

[13] LIPKE, W., 2004, "Independent Estimates at Completion – Another Method", Cross Talk The Journal of Defense Software Engineering, v. 17 nro 10 (october 2004), pp. 32.

[14] LIPKE, W., 2006, "Statistical Methods Applied to EVM: The Next Frontier", Department of Defense - USA, v. 19, pp. 32, June.

[15] LIPKE, W., 2008, "Project Duration Forecasting: Comparing Earned Value Management Methods to Earned Schedule", Cross Talk The Journal of Defense Software Engineering, v. 21 no. 12 (December 2008), pp. 32.

[16] LIPKE, Z., ANBARI, HENDERSON, 2009, "Prediction of project outcome, the application of statistical methods to Earned Value Management and Earned Schedule performance indexes", International Journal of Project Management,.

[17] PFLEEGER, S. L. e FENTON, N.E., 1997, Software Metrics: A Rigorous and Practical Approach Boston, PWS Publishing Company, 2nd Edition.

[18] PMI, 2005, Practice Standard Earned Value Management Pennsylvania, Project Management Institute.

[19] PMI, 2009, Project Management Body of Knowledge - PMBOK Newton Square, Project Management Institute.

[20] PUTNAM, L.H., 2003, Five Core Metrics: The Intellligence Behind Successful Software Management, Dorset House.

[21] RUBINSTEIN, D., 2009, "Standish Group Report: There's Less Development Chaos Today", 2009

[22] VANDEVOORDE, S., MARIO VANHOUCKE, 2006, "A comparison of different project duration forecasting methods using earned value metrics", Project Management Journal, v. 24 (accepted 14 October 2005), pp. 289 - 302.

[23] ZWIKAEL, O., ET AL, 2000, "Evaluation of Models for Forecasting the Final Cost of a Project." ", Project Management Journal v. 31.1 pp. 53-57.

A Goal-Driven Method for Selecting Issues Used in Agent Negotiation

Yen-Chieh Huang and Alan Liu
Department of Electrical Engineering
National Chung Cheng University
Chiayi, Taiwan
aliu@ee.ccu.edu.tw

***Abstract*—Agent negotiation is an important process in cooperation among intelligent agents, and the selection of suitable issues will have a direct impact on negotiation results. However, the way to determine such issues is usually ad hoc. To introduce a systematical way to extract issues based on requirements analysis, this study proposes a method to find negotiation issues using goal-driven use case analysis. The issues are derived based on the nonfunctional requirements that are part of the user goals. After deriving possible issues, it is important to rate how well the issues can be used. For evaluating issues, we use the analytic hierarchy process to compare and choose a combination of issues. The contributions of this paper include, first, a method for providing issues traceable to their sources and clear information about the relationship between the issues. Second, a selection method is provided for choosing a suitable set of issues from a number of proposals.**

Keywords--goal-driven analysis; agent negotiation; negotiation issues; requirements analysis; agent-based software engineering

I. INTRODUCTION

Agent cooperation and negotiation have become important research topics in multiagent systems [1][2]. Agents perform negotiation based on a set of issues, and the choice of issues may influence the result of negotiation. If issues are chosen randomly, the result of negotiation may not be desirable [3]. With correct issues chosen, the negotiation process may speed up and also result in better conflict resolution. However, many methods assume that the issues for negotiation are already given or there is a mediator who is able to make the negotiation possible for agents involved [4]. Since the method of finding issues has not been studied well, this article proposes a method based on goal-driven requirements analysis to find usable negotiation issues based on user requirements. With a possible set of issues on hand, the next process is to find the most suitable set of issues to be used in negotiation. For this challenge, our method provides a way to analyze and compare different sets of issues in order to choose the most suitable set of issues for a given negotiation problem.

For establishing a procedure of how to derive and evaluate a set of issues, we use a goal-driven method, called the Goal-Driven Use Case (GDUC) approach [5], to analyze the user requirements for keeping track of the derived goals from the functional and nonfunctional requirements. GDUC uses goals to model use cases and their corresponding relationship and to derive user goals. During the process, rigid and soft goals are identified, and functional and nonfunctional requirements are evaluated. Negotiation issues are extracted based on these user goals. In order to provide a way of comparing issues, we have studied some multi-criteria decision methods and decided to use AHP [6] because of its qualitative approach for comparing pairs of issues.

The rest of the paper is organized as follows. The next section discusses our proposed method. Section three presents an illustration along with an experiment. The last section gives a brief conclusion with discussion.

II. PROPOSED METHOD

As shown in Figure 1, our method consists of three phases. First, we apply GDUC to analyze the goals and their relationship from requirements. Second, we analyze the goals with attributes involved in objects involved to extract possible issues to be used in negotiation. Lastly, we use AHP to evaluate all issues for forming a set of proposals containing issues. Finally, the ranking of the proposals is produced.

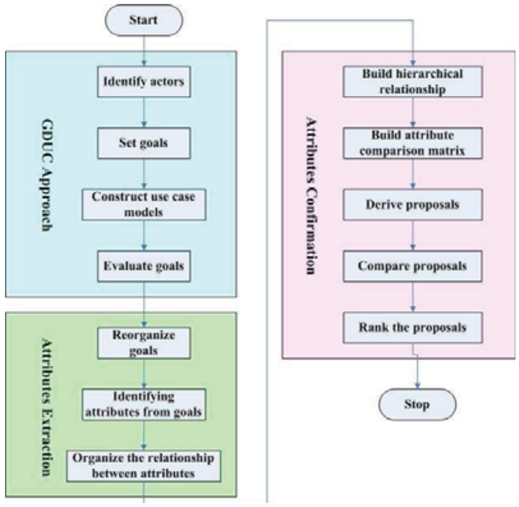


Figure 1. Relationship between goals and use cases

This research is partially supported by National Science Council, Taiwan under the grant numbers NSC-100-2221-E-194-011 and NSC-97-2221-E-006-160-MY3.

With the GDUC approach, we can model functional and nonfunctional requirements by extending the use cases while analyzing the requirements as shown in Figure 2. A goal is associated to the original use case (U1), and U1 is extended to more detail use cases having their respective goals to achieve. These goals carry richer information that will be discussed in the next section. A goal can be classified with three facets: competency, view, and content. Looking into competency, we can model a goal as a rigid goal or a soft goal. A rigid goal is the goal that needs to be fulfilled fully while the requirement of a soft goal is more relaxed. The view of requirements can be actor-specific or system-specific, and the content can be functional or nonfunctional requirements. The content of a goal is derived based on functional or nonfunctional requirements.There are four steps used repeatedly in this process: identifying the actors, setting goals, constructing a use case model, and evaluating the goals.

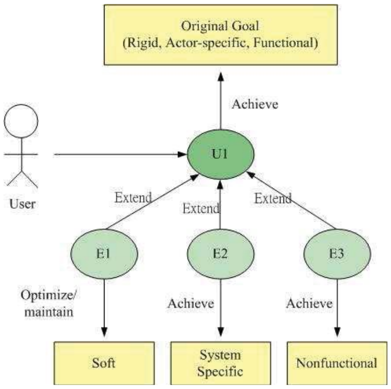


Figure 2. Relationship between goals and use cases

After defining the categories and the relationship among goals, we now focus on identifying what issues are associated to the goals [7]. First, we can organize the existing information among goals, including goal categories, roles, cooperative goals, conflicting goals, and counterbalanced goals. Goal categories contain issues that will affect the result of analysis, and the actors and use cases provide attributes that describe the actors and their relationships. With such attributes discovered, we can analyze their relationship to see if they are cooperative, conflicting, or counterbalancing to each other.

Identifying possible issues based on these attributes, we need to select which issues are best fit for the negotiation process. Comparing all issues at once causes heavy computation, so we use a hierarchical approach in evaluating issues. Issues are compared in pairs and the resulting scores are recorded in a matrix. The first step is to define the levels, and then the matrix with comparison results is constructed. Proposals containing different set of issues are derived, and the proposals are compared. Finally, the order of favorable proposals is determined.

III. Illustration and Experiment

A. Scenario and Assumption

For presenting the detail steps of our approach, a scenario about a market activity is presented below to illustrate how our method can be used. There are three players in this purchasing negotiation: a boss, a customer, and a salesperson. The boss expects to gain the largest benefit from a transaction and has the control over the pricing activity. He gives the price guideline to the salesperson while interacts with the customer for special needs. The customer has a certain habit for purchase and is looking for a lower price for merchandise. For ordinary purchase, he interacts with the salesperson, but for some special requests, he works out with the boss. The salesperson needs to meet the assigned sales quota by using his ability to persuade customers. He has some freedom in lowering price, but the boss has the final say. In a usual setting like in Figure 3, the customer first encounters with a salesperson, and the salesperson promotes some merchandise to the customer. When there is a special purchase request such as large volume of purchase, the customer may be directly dealing with the boss.

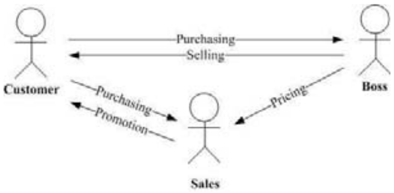


Figure 3. Relationship between participants in selling items

B. Gaol-Driven Analysis

We start from identifying actors as *Boss*, *Customer*, and *Salesperson* with their corresponding use cases as shown in Figure 4. For these actors, we define the corresponding goals, such as *boss* to get the highest benefit, *customer* to purchase necessary items, and *salesperson* to sell items. Using three facets (competency, view, and content) to analyze these goals, we observe that these goals are rigid since they must be fulfilled. They are also actor-specific and functional. In enriching the user model, we extend the original use case with more use cases to provide more information that is embedded in the requirements. These extended use cases are able to associate with goals that may be rigid or soft goals with actor- or system-specific and functional or nonfunctional requirements.

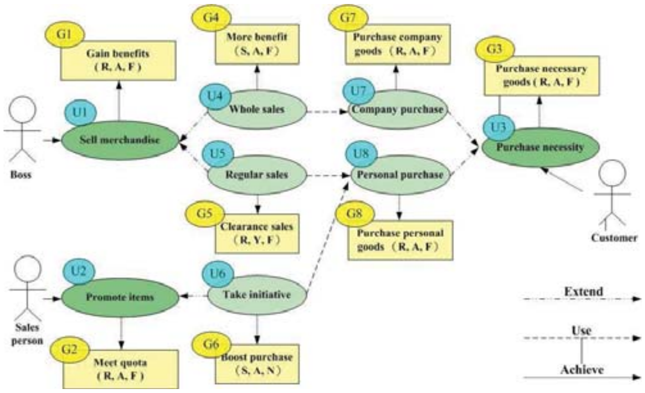


Figure 4. Use case model with associated goals

Identifying the relationship between the use cases and corresponding goals based on our scenario, we can extend it to “Whole sales” and “Regular sales” from the original use case, “Selling Merchandise”. The use case “Purchase Necessity” is then extended to “Company Purchase” and “Personal Purchase”, and the use case “Promote items” is then extended to “Take Initiative”. Each goal is labeled with *R* or *S* (rigid or soft goal), *A* or *Y* (actor-specific or system-specific requirements), and *F* or *N* (functional or nonfunctional requirements). Lastly, the goal evaluation takes place for evaluating the relationship between the use cases and goals, the interaction between goals in the use case level, and the interaction between goals in the system level.

For analyzing the relationship between the goals and the use cases in terms of their degree of relevance, we use four levels to describe their relationship. A pair of use case and its corresponding goal is compared to see whether they are highly relevant (labeled as ++) with each other. Otherwise, they can be partially relevant (+), partial irrelevant (-), or totally irrelevant (--) to each other. The result of checking such relationship based on the example in Figure 4 is recorded in Table I.

TABLE I. RELATIONSHIP BETWEEN USE CASES AND GOALS

	G1	G2	G3	G4	G5	G6	G7	G8
U1	++	+	+	+	++	-	++	++
U2	+	++	-	+	+	++	--	+
U3	++	+	++	+	++	-	++	++
U4	++	+	+	++	++	-	++	--
U5	++	+	+	+	++	+	--	+
U6	+	+	-	+	+	++	-	+
U7	++	+	+	++	++	--	++	--
U8	++	+	+	+	++	+	--	++

By analyzing the relationship between goals at the system level, we state that if both goals are cooperative to each other for all use cases, then they are cooperative to each other at the system level. If all of them are conflicting to each other at the use case level, then they are conflicting to each other at the system level. We use the concept of cooperative, counter-balanced, and conflict relationship to describe such interactions, and the result is shown in Table II.

TABLE II. RELATIONSHIP BETWEEN GOALS
(Co: Cooperative, Bal: Counterbalanced, Conf: Conflict)

G1,G2	G1,G3	G1,G4	G1,G5	G1,G6	G1,G7	G1,G8
Co	Bal	Co	Co	Bal	Bal	Bal
G2,G3	G2,G4	G2,G5	G2,G6	G2,G7	G2,G8	G3,G4
Bal	Co	Co	Bal	Bal	Bal	Bal
G3,G5	G3,G6	G3,G7	G3,G8	G4,G5	G4,G6	G4,G7
Bal	Bal	Bal	Bal	Co Ba	l	Bal
G4,G8	G5,G6	G5,G7	G5,G8	G6,G7	G6,G8	G7,G8
Bal	Bal	Bal	Bal	Conf	Bal	Bal

When comparing the interaction between a pair of goals, we consider a goal consisting of a set of entries corresponding to use cases and compare each pair of entries. Thus, using Table I as example, we check two columns to compare the entries row by row. If all rows are ++ or +, then the pair of such goals are cooperative. If each rows have one entry with ++ or + while the other entry has – or --, then they are conflicting to each other. If the entries are mixed, then they are called counter-balanced.

C. Acquisition of Possible Negotiation Issues

We construct a table indicating the information of each goal, which consists of the category of a goal and the associated actor along with the information about cooperative, conflicting, and counterbalanced goals. Based on such information, we identify a possible issue that will be most influential to a goal. The result is shown in Table III, where we see that the goal G1 is a rigid goal containing actor-specific and functional requirements. G1 belongs to the actor Boss, and is cooperative to goals G2, G4, and G5. It is counter-balanced to the goals G3, G6, G7, and G8, but does not conflict to any other goals. Because of such characteristics, we identify an issue, *Price*, as the most representative issue concerning this goal. The rest of the issues, namely *Eloquence* (ability of presentation), *Necessity*, *Quantity*, *Quality*, and *Budget*, are derived in the same manner.

TABLE III. GOAL ISSUES AND RELATIONSHIP

	Category	Actor	Coop	Conf	Counterbalanced	Issue
G1	R,A,F	Boss	G2,G4,G5		G3,G6,G7,G8	Price
G2	R,A,F	Sales	G1,G4,G5		G3,G6,G7,G8	Eloquence
G3	R,A,F	Customer			G1,G2,G4,G5G6,G7,G8	Necessity
G4	S,A,F	Boss	G1,G2,G5		G3,G6,G7,G8	Quantity
G5	S,A,F	Boss	G1,G2,G4		G3,G6,G7,G8	Price, Quality
G6	S,A,N	Sales		G7	G1,G2,G3,G4G5,G8	Price, Eloquence
G7	R,A,F	Customer		G6	G1,G2,G3,G4 G5,G8	Budget
G8	R,A,F	Customer			G1,G2,G3,G4 G5,G6,G7	Price, Quality

With six kinds of issues identified, we can examine the pair-wise relationship between these issues and take the geometric mean to define weighted comparison as shown in Figure 5. A hierarchical relationship is shown in Figure 6 to organize the information, in which Level 2 shows the possible issues. Based on these issues, we can derive possible proposals in Level 3. Proposal A contains the two issues with highest values: *Budget* and *Necessity*; Proposal B adds *Price* to Proposal A; and Proposal C includes *Quality* to Proposal B.

$$\begin{array}{r} \text{Price} \\ \text{Eloquence} \\ \text{Necessity} \\ \text{Quantity} \\ \text{Quality} \\ \text{Budget} \end{array} \begin{bmatrix} (1 * 7 * 1 * 3 * 2 * 1/3)^{1/6} \\ (1/7 * 1 * 1/9 * 1/3 * 1/3 * 1/9)^{1/6} \\ (1 * 9 * 1 * 5 * 5 * 1/3)^{1/6} \\ (1/3 * 3 * 1/5 * 1 * 1/5 * 1/7)^{1/6} \\ (1/2 * 3 * 1/5 * 5 * 1 * 1/3)^{1/6} \\ (3 * 9 * 3 * 7 * 3 * 1)^{1/6} \end{bmatrix}$$

$$= \begin{bmatrix} 1.552 \\ 0.241 \\ 2.053 \\ 0.423 \\ 0.891 \\ 3.455 \end{bmatrix} \Rightarrow \begin{bmatrix} 0.18 \\ 0.03 \\ 0.24 \\ 0.05 \\ 0.10 \\ 0.40 \end{bmatrix}$$

$$\text{Sum} = 8.615 \qquad \text{Sum} = 1.0$$

Figure 5. Calculation for deriving weighting for issues.

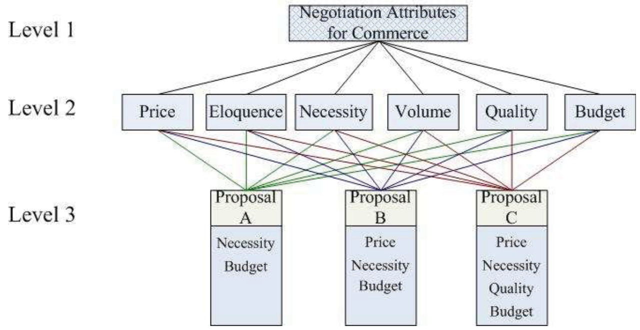


Figure 6. Three proposals containing most promising issues.

Using AHP, we can rate the issues according to each proposal. Tables II and III are used as reference for making the decision for what points to give to each issue in a proposal. If the issues belong to the goals that are cooperative, then the points are higher, and the points are lower if the goals are conflicting to each other. Table IV shows three proposals with the points given for different issues. For each issue, we compare three proposals pair-wise by finding geometric mean for weighting the issues, and the results are listed in Table V.

TABLE IV. POINTS FOR ISSUES IN DIFFERENT PROPOSALS

Proposal	Price	Eloquence	Necessity	Quantity	Quality	Budget
A	1	1	5	3	1	5
B	5	3	6	5	6	3
C	6	4	6	7	8	3

TABLE V. WEIGHTING FOR ISSUES ACCORDING TO PROPOSALS

Proposal	Price	Eloquence	Necessity	Quantity	Quality	Budget
A	0.083	0.125	0.294	0.200	0.067	0.455
B	0.417	0.375	0.353	0.333	0.400	0.273
C	0.500	0.500	0.353	0.467	0.533	0.273

Considering Table V as a matrix M_{ij} and the normalized matrix in Figure 5 as N_j, we can evaluate proposals by the sum of the product of weights. Thus, $P_i = \sum M_{ij} N_j$ where P_i is the evaluation for Proposals A to C, produces $P_0=0.376$, $P_1=0.376$, and $P_2=0.376$. From the result, we see that the proposal C is most promising. However, it contains more issues and requires more computation than Proposals A or B. If the computation is a concern, then Proposal B is a good alternative.

IV. DISCUSSION AND CONCLUSION

The effectiveness (E) of a set of issues can be calculated by $E= \Sigma IU_i / \Sigma PI_i$, where the numerator is the sum of weights of all issues used (a set *IU*) in a proposal and denominator is the sum of weights of all possible issues (a set *PI*) available. When negotiating a purchase, usually the issue *Price* is a natural choice as most influential to affect the sales of an item. However, compared to those proposals mentioned above, *E* becomes small if *Price* is the only issue in *IU*. Besides this intuitive discussion, we have developed a simulation environment to execute negotiation process to compare the use of different sets of issues. For experiment, negotiation is run 50 times. The success rate of a salesperson is set as 20% for a correctly chosen issue; otherwise, a salesperson has only 5% success rate to land a deal. If no deal is reached, then 1% to 3% price cut is tried until the price offered is lower than 90% of the original price. The negotiation based on the single issue *Price* resulted in 19 successes and 31 failures while Proposal A had 28 successes, Proposal B 32 su ccesses, and Proposal C 38 successes. A notable performance of Proposal C is that it enabled a deal with 5 or fewer rounds of negotiation. As seen in Table VI, the result of the simulation showed that the more effective the issues are, the higher the number of success becomes.

TABLE VI. SIMULATION RESULTS

	Success	Rounds to reach a deal						Failure
		1	2	3	4	5	6~10	
Price	19	6 (31.6%)	8 (42.1%)	1 (5.3%)	1 (5.3%)	1 (5.3%)	2 (10.6%)	31
A	28	9 (32.1%)	5 (17.9%)	8 (28.6%)	2 (7.1%)	3 (10.7%)	1 (3.6%)	22
B	32	8 (25%)	11 (34.4%)	3 (9.4%)	3 (9.4%)	4 (12.5%)	2 (6.25%)	18
C	38	14 (36.8%)	7 (18.4%)	7 (18.4%)	4 (10.5%)	6 (15.8%)	0 (0%)	12

Many agent negotiation strategies depend on th e correct issues to work on. The selection of issues for negotiation is important, and there is a need for a process in determining usable issues. This paper proposes a systematical method based on goal-driven requirements analysis to produce a set of issues for agent negotiation. The issues are traceable to user requirements for better evaluation of effectiveness of issues used and also the maintenance of issues. The relationship between issues is studied to u nderstand whether they are in cooperative or conflicting matter. Another important advantage of this method is to have information for comparing the number of issues to be used in a negotiation process. By integrating the GDUC and AHP methods, our method provide a way for the user to determine suitable issues according to the user requirements. The future work is to consider the dependency among issues, so that a proposal consists of a set of mutually related issues.

REFERENCES

[1] G. Lai and K. S ycara, "A Generic Framework for Automated Multi-attribute Negotiation," *Group Decision and Negotiation*, vol. 18, no. 2, pp. 169–187, Mar. 2009.

[2] R. Krovi, A. C. Graesser and W. E. Pracht, "Agent Behaviors in Virtual Negotiation Environments", *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol. 29, no. 1, pp.15-25, 1999.

[3] M. Hall and G. Holmes, "Benchmarking Issue Selection Techniques for Discrete Class Data Mining", *IEEE Trans. Knowledge and Data Eng.*, vol. 15, no.6, pp.1437-1447, 2003.

[4] N. R. Jennings, P. Faratin, A. R. Lomuscio, S. Parsons, C. Sierra and M.Wooldridge, "Automated negotiation: Prospects, methods and challenges", *Int Journal of Group Decision and Negotiation*, vol. 10, pp.199–215, 2001.

[5] J. Lee and Nien-Lin Xue. "Analysis User Requirements by the Use Cases : A Goal-Driven Approach" , *IEEE Software*, vol. 16, no. 4, pp. 92−101, 1999.

[6] T. L. Saaty, "Highlights and Critical Points in The Theory and Application of The Analytic Hierarchy Process", *European Journal of Operational Research*, vol.74, no. 3, pp. 426-447, 1994.

[7] J. Lee , N.L. Xue, and K.Y. Kuo, "Structuring requirements specifications with goals," *Information and Software Technology*, vol. 43, pp. 121-135, 2001.

Using Cell Phones for Mosquito Vector Surveillance and Control

S. Lozano-Fuentes, S. Ghosh, J. M. Bieman, D. Sadhu, L. Eisen
Colorado State University
Fort Collins, CO 80523, USA

E. Hernandez-Garcia, J. Garcia-Rejon
Universidad Autonoma de Yucatan
Merida, Mexico

F. Wedyan
Hashemite University
Zarka, Jordan

D. Tep-Chel
Instituto Tecnologico Superior de Motul
Motul de Carrillo Puerto, México

***Abstract*—Novel, low-cost approaches to improving prevention and control of vector-borne diseases, such as mosquito-borne dengue and malaria, are needed in resource-constrained environments. The Chaak application supports the use of cell phones for field capture and rapid transfer of mosquito vector surveillance data to a central database. The cell phones exploit existing communication infrastructure, introduce near real-time monitoring, and provide rapid feedback to field data collectors. Dengue is a mostly an urban disease, thus occurring in environments that often have good cell phone coverage. Cell phones eliminate the need for physical data communication. A preliminary evaluation shows that the use of cell phones can lower labor costs, data collection time, and transcription errors.**

Keywords-Android applications; dengue; mosquito control; mosquito immatures; public health; field testing

Introduction

Vector-borne diseases, such as mosquito-borne dengue and malaria inflict a terrible and unacceptable burden on mankind and block socio-economic development in many parts of the world. Colorado State University and partners in Mexico (Universidad Autonoma de Yucatan and the public health institution of Servicios de Salud de Yucatan) have developed a software application as a low-cost solution to improve the surveillance and control of mosquito vectors of dengue virus. Data collected with this application can help public health institutions to determine where and when mosquito control efforts should be focused.

Since mosquito *immatures* (larvae and pupae) develop in water, the risk of mosquito exposure can be estimated by collecting information about the presence of immatures in stagnant water. Surveyors typically go door-to-door to count the containers on the premises (inside and outside) with and without water. Containers may include buckets, tires, and cisterns. The surveyors also count the containers holding immatures. This data is collected on paper forms and entered by humans into a computer in batches. This process of data entry is both error-prone and time consuming, which reduces the accuracy and slows vector control responses. A timely response is critical for diseases with explosive outbreak dynamics, such as dengue [4, 6, 13].

The new Chaak application uses cell phones to collect data. Chaak eliminates the need for manual data entry from a paper form, avoiding data transcription errors. The process is also faster; data can be transmitted to a central database as soon as it is entered into the phone. Using cell phones also eliminates the need to carry bulky laptops and transfer data via flash drives. Dengue is primarily an urban disease, and cell phone infrastructure is generally available in urban areas in developing countries.

Design and Implementation of Chaak

Chaak users include a system administrator, management personnel for vector control, and surveyors who enter container data into the phones. System administrators and management personnel use a desktop client and surveyors use Android phones to enter the data.

A simple geographic information system based on Lozano-Fuentes *et al.* [9] was extended to allow the system to create maps and manage geographic data. Reports are generated for managers to view mosquito data in various geographic areas.

This research was sponsored by the National Institutes of Health grant number 1R21AI080567-01A1 and a generous gift from Qualcomm to purchase cell phones.

Contact Author: Saul Lozano-Fuentes (saul.lozano-fuentes@colostate.edu)

Roles and functionality

System administrators can create valid Chaak users. Administrators can also register authorized cell phones for use in collecting data. Managers can define geographical entities to be surveyed, container types, and assign tasks to surveyors.

Every country has its own geographical hierarchy (e.g., country, states, cities, and neighborhoods). Addresses are expressed in country-specific ways. A geographical hierarchy is matched with the printed address of the premises so that surveyors can be assigned tasks involving visits to a set of premises. Defining geographic entities involves (1) defining the elements in the hierarchy (GeoClasses, such as Country) and (2) defining instances of elements (GeoEntities, i.e., Mexico).

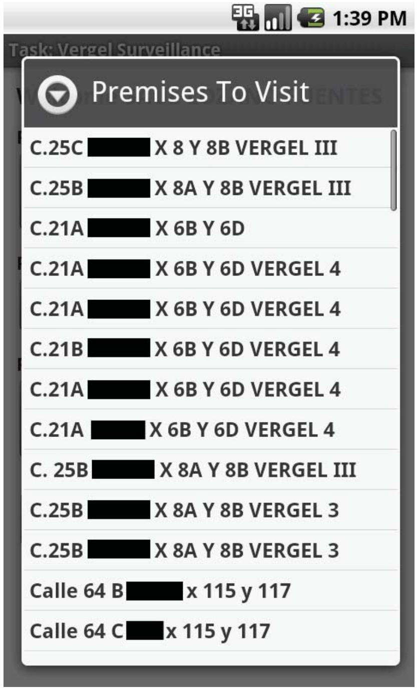
Container types (e.g., buckets, tires, or cisterns) are defined and added for data collection.

Managers create surveillance tasks — specific premises are assigned to surveyors for completion in a given time period. A new task will include specific geo-entities (premises in neighborhoods) and surveyors.

Surveyors log in to a phone to see a list of premises to visit. Figure 1 is a phone screenshot showing the list of premises as used in Merida, Mexico. Surveyors enter the number of each type of container and the number containing larvae or pupae using the interface shown in Figure 2. A surveyor can transmit the data or save the data in the phone and upload it later.

For robust performance, the cell phone can connect to the main server using three options:

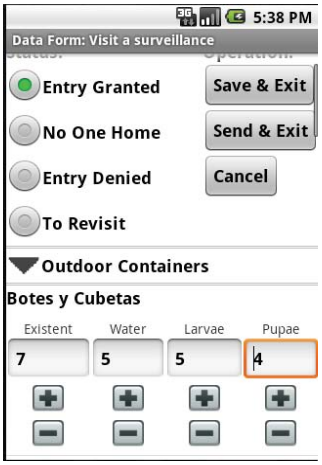
1. ***Continuous Internet Access:*** The surveyor enters data into the phone and sends it instantly to the main server over the internet.
2. ***Periodic Indirect Access:*** The surveyor enters data and saves it in the local application database on the phone. The surveyor sends the data to the main server over the internet when a connection is available.
3. ***Desktop/laptop Data Transfer:*** The surveyor enters data and saves it in database on the phone. The data is transferred to a computer at the main office. An internet connection is not needed.



Premises To Visit

C.25C [redacted] X 8 Y 8B VERGEL III
C.25B [redacted] X 8A Y 8B VERGEL III
C.21A [redacted] X 6B Y 6D
C.21A [redacted] X 6B Y 6D VERGEL 4
C.21A [redacted] X 6B Y 6D VERGEL 4
C.21B [redacted] X 6B Y 6D VERGEL 4
C.21A [redacted] X 6B Y 6D VERGEL 4
C.21A [redacted] X 6B Y 6D VERGEL 4
C. 25B [redacted] X 8A Y 8B VERGEL III
C.25B [redacted] X 8A Y 8B VERGEL 3
C.25B [redacted] X 8A Y 8B VERGEL 3
Calle 64 B [redacted] x 115 y 117
Calle 64 C [redacted] x 115 y 117

Phone Client Screenshot Showing List of Premises



Data Form: Visit a surveillance

● Entry Granted
○ No One Home
○ Entry Denied
○ To Revisit

Outdoor Containers

Botes y Cubetas

Existent	Water	Larvae	Pupae
7	5	5	4

Phone Client Screenshot Showing Details of One House

Design and Implementation

Chaak is a multi-tier application. The top tier is the application tier consisting of thin clients on desktops running Windows .NET and smartphones running Android. We used a variety of Android phones (Google Nexus 1, S and Galaxy Nexus, HTC Desire, HTC Legend and Motorola Backflip) running Android versions 2.1 and higher. The Android application communicates with the

Business Tier using web services in the Business Access Tier. The web services are hosted on an Apache webserver on Mono. The business tier supports data validation, decisions, evaluations, calculations, and CRUD operations. Data is stored in a PostgreSQL server. A PostGIS plugin manages spatial data such as blocks and premises represented as polygons and dots in space.

Empirical Evaluation and Field Test

The goal of our evaluation was to determine whether or not using cell phones as the data capturing device would increase the surveyor's performance and decrease the number of errors made. The performance could instead decrease because surveyors had to learn to use a new device and the device itself can be difficult to read under direct sunlight. Additionally, simple things like sweat or body oils can interfere with the touchscreen response. Error in data capture might increase by adding new sources of errors in the workflow.

The surveyors had extensive experience performing entomological surveys using paper forms, giving an advantage to the pen-and-paper method. However, the surveyors were familiar with cell phones and touchscreen technology. To minimize the discrepancy between the methods, the surveyors were trained in the use of the cell phone interface.

To assess differences between the methods, we performed two experiments. One measured time differences and the other measured error. The time experiment was performed in actual houses under field conditions and the error experiment was performed in a controlled environment because even experienced surveyors can overlook breeding sites..

Time experiment. We hypothesized that the time used for completing entomological surveys was the same regardless of the method used. Through a pilot study, the required experiment sample size was estimated at 118 premises to be visited per survey method.

To achieve the desired sample size, we randomly selected 13 neighborhoods in the city of Merida, and from each neighborhood we randomly picked 2 blocks. All premises in the selected blocks were to be visited and if possible, surveyed by one of two teams. Each surveying team consisted of 3 surveyors. One block was surveyed using only mobile devices and the other using only pen-and-paper.

The survey time was defined as the time used to complete a survey from the entry of the surveyor into the premises, having obtained permission from the tenant, until the surveyor left the premises. We did not record the time used moving between premises or between the headquarters and the premises because this time is independent of the method used. On the other hand, the surveyors that used the pen-and-paper method had to re-enter the data in the paper forms into a computer spreadsheet. We added this time to the time used for surveying.

Due to logistical difficulties, only 10 neighborhoods of the selected 13 were visited. Using a cellphone, 300 premises were visited, and 299 premises were visited using pen-and-paper. Despite the effort, not all visited premises were surveyed either because the tenant was not home or because the surveyor was not allowed to enter the premises. Only 24% of the visited premises were surveyed using a cellphone and 34% using paper. The total time utilized during the survey was similar for both methods with 567 and 552 minutes for paper and mobile method respectively. However, it took an additional 384 minutes to reenter data from the paper forms into the system.

Method Performance

Method	Total time	No. of Surveys	Minutes/survey
Paper	961	100	9.6
Mobile	552	71	7.8

In all, the pen-and-paper method required 961 minutes of effort with a performance of 9.6 minutes per survey. In contrast, the surveyor using mobile devices had a performance of 7.8 minutes per survey (Table 1). This increase in performance translates into a 19% improvement when using mobile devices.

We tested the hypothesis that the average time ($T\mu$) was the same for both methods at $\alpha = 0.05$:

$$H_0: T\mu_{mobile} = T\mu_{paper}$$

We obtained $|Z| = 2.4$ and $P(|Z| \leq 2.4) = 0.016$ thus rejecting the H_0. Therefore there are significant differences between the amounts of time used by the methods. Having found this statistical difference, we proceeded to perform a one-sided test.

$$H_0: T\mu_{mobile} \geq T\mu_{paper}$$

In this test Z = -2.4 and $P(Z \geq -2.4) = 0.0082$ consequently we reject the null hypothesis. Thus, there is no statistical evidence to affirm that the time used while using the mobile device is greater or equal than the time using pen-and-paper.

Error experiment. The objective was to evaluate if there was a significant difference in the number of errors made between the two data collection methods. The pen and paper method was divided in two stages: the survey and the transcription to an electronic spreadsheet. In the cell phone method, only one stage is performed because there is no need to re-capture the data entered on the mobile devices.

In order to have exact numbers for this experiment, the containers and the premises were simulated. The containers were represented with printouts in letter size paper divided in four panels. The first panel had a drawing representing the container type, panel 2 the presence of water, panel 3 whether the container had larvae, and panel 4 whether the container had pupae. Only containers with water had larvae and/or pupae. The premises were represented by sections in a large room divided into 6 sections.

In each mock premises we places 20 and 35 container printouts with varying numbers of immatures. The number of containers and their condition (with water and immatures) were selected at random. The entire experiment was divided into 10 rounds of 5 minutes each. The survey method was assigned randomly to each surveyor. In summary, each surveyor made 10 mock entomological surveys, completing 30 surveys for method. At the end of the day, the surveyors using pen-and-paper reentered the collected data on a computer.

Errors were scored as deviations from the known exact value. A value of 0 means a perfect score and a value of 1 means that the surveyor recorded all possible values incorrectly. Each breeding site has 4 records in the survey form as previously described. Because there are 11 different container types in the survey we can say that the surveyors had to record 44 values, i.e., they had 44 chances of success. Thus, the proportion of errors is the total sum of errors divided by the total number of chances of success. The total proportion of errors (E) is simply the averaged proportion of errors by method.

The proportion of errors for the pen-and-paper method was E_{paper} = 0.23 while the proportion of errors for the mobile method was E_{mobile} = 0.17. We tested the following one-sided hypothesis:

$$H_0: E_{mobile} \geq E_{paper}$$

We obtained Z = 3.865 accordingly $P(Z \leq 3.865) < 0.001$, and thus we can reject the null hypothesis. Consequently the proportion of errors using cell phones is less than the proportion of errors using the pen-and-paper method.

Related Work

Donner [3] documents the growth of cell phone use throughout the developing world. Mobile phones are now commonplace throughout most of the world, with "over a billion mobile phones in the developing world" by 2008.

Brewer *et al.* [1] find that cell phone technology supports data collection improvements for disease management. In developing countries, effectiveness requires it "be designed around asynchronous communication and only intermittent connectivity". Chaak can pass information between clients and server through undependable connections.

Cell phone use is commonplace throughout the developing world. Thus, field workers with cell phones do not look out of place, even in the poorest communities. Curioso [2] reports that by replacing stacks of folders with a cell phone, public health nurses in Peru can discretely and more safely visit the poorest rural communities without arousing the suspicions of residents, or appearing as likely robbery victims. The success of the project is Peru is because, in part, the system developers understood the problem domain, and obtained feedback from health professionals. That is why we developed Chaak in close association with field workers and entomologists in Merida, Mexico. Kahn *et al.* [5] emphasize the need for evaluations of the benefits of using what they call "mobile health or m-health" technologies. That is why we conducted comprehensive field tests of the cell phone client data collection systems.

Kaplan [7] did not find convincing evidence that cell phone technology can be an effective tool for healthcare interventions for two primary reasons: lack of access to phones by citizens, and limited evaluations of effectiveness. The Chaak system relies on cell phone use by field workers rather than community members, and we are evaluating effectiveness.

Krishna *et al.* [8] conducted a systematic literature review that identified 25 empirical evaluations of the effects of cell phone use on public health. These studies focused on relationships between health care providers and patients and the effects on patient outcomes. Our study focuses on cell phone use by field workers rather than patients, and on the collection of data for the management of the insect vector of the disease.

In a study conducted near Durban South Africa, Tomlinson *et al.* [12] found that the use of cell phones by community health workers to collect field data provided significant benefits over the use of personal digital assistants (PDAs) as well as paper forms. The cell phones uploaded data directly to a server from the field, and avoided the delays and labor of processing paper forms. Data was uploaded as it was gathered, dramatically reducing data loss.

In multiple large-scale evaluations conducted in South Africa, Seebregts *et al.* [11] found that the use of Palm^{TM} Pilot PDAs provided benefits over paper forms in collecting health-oriented survey data. Both respondents and field workers preferred the PDA system to paper, with significantly lower overall costs. These studies collected approximately 90,000 interview records using as many as 200 Palm^{TM} Pilot PDAs used for approximately 50 device-years. Battery life was the major problem encountered.

Conclusions

We present the design, implementation, and field evaluation in Mexico of a smartphone-based application for the collection of data on mosquito vectors of dengue virus. The application improved the accuracy of data collection and significantly increased the speed of data transcription.

Acknowledgment

We thank Fernando Chan and Mildred Lopez who helped to field test the Chaak application.

References

E. Brewer, M. Demmer, B. Du, M. Ho, M. Kam, S. Nedevschi, J. Pal, R. Patra, S. Surana, K. Fall. The case for technology in developing regions. IEEE Computer , vol.38, no.6, pp. 25- 38, May 2005.

W. H. Curioso. New technologies and public health in developing countries: The Cell-PREVEN project. In: Murero M, Rice R, editors. The Internet and health care: Theory, research and practice. Mahwah (NJ): Lawrence Erlbaum Associates. pp. 375–393, 2006.

J. Donner. Research approaches to mobile use in the developing world: a review of the literature. The Information Society, 24:140-159, 2008.

P. Jeefoo, N.K. Tripathi, M. Souris. Spatio-Temporal Diffusion Pattern and Hotspot Detection of Dengue in Chachoengsao Province, Thailand. Int. J. Environmental Research and Public Health. 2011; 8(1):51-74.J. G. Kahn, J. S. Yang, J. S. Kahn. Mobile health needs and opportunities in developing countries. Health Affairs, 29:2, pp. 254-261, 2010.

J.G. Kahn, J.S. Yang, J.S. Kahn. Mobile health needs and opportunities in developing countries. Health Affairs, 29:2, pp. 254-261, 2010.

Chih-Chun Kan, Pei-Fen Lee, Tzai-Hung Wen, Day-Yu Chao, Min-Huei Wu, Neal H. Lin, Scott Yan-Jang Huang, Chuin-Shee Shang, I-Chun Fan, Pei-Yun Shu, Jyh-Hsiung Huang, Chwan-Chuen King, and Lu Pai Two Clustering Diffusion Patterns Identified from the 2001–2003 Dengue Epidemic, Kaohsiung, Taiwan Am J Trop Med Hyg September 2008 79:344-352

W.A. Kaplan. Can the ubiquitous power of mobile phones be used to improve health outcomes in developing countries? Globalization and Health, 2:9, 2006.

S. Krishna, S.A. Boren, E.A. Balas. Healthcare via cell phones: a systematic review. Telemedicine and e-Health. 15(3): 231-240, 2009.

S. Lozano-Fuentes, D. Elizondo-Quiroga, J. A. Farfan-Ale, M. A. Loroño-Pino, J. Garcia-Rejon, Gomez-Carro Salvador et al., Use of Google EarthTM to strengthen public health capacity and facilitate management of vector-borne diseases in resource-poor environments. Bull World Health Organ. 2008 Sep; 86(9): 718-725.

C. Pop-Elechesa, H. Thirumurthy, J. P. Habyarimanae, J. G. Zivinf, M. P. Goldsteing, D. de Walqueg, L. MacKeen, J. Habereri, S. Kimaiyoj, J. Sidlek, D. Ngarem, and D. R. Bangsberg. Mobile phone technologies improve adherence to antiretroviral treatment in a resource-limited setting: a randomized controlled trial of text message reminders. AIDS. 25(6), 27 March 2011, p 825–834.

C. J. Seebregts, M. Zwarenstein, C. Mathews, L. Fairall, A. J. Flisher, C. Seebregts, W. Mukoma, K. Klepp. Handheld computers for survey and trial data collection in resource-poor settings: Development and evaluation of PDACT, a PalmTM Pilot interviewing system. Int. J. Medical Informatics, 78 (2009) 721-731.

M. Tomlinson, W. Solomon, Y. Singh, T. Doherty, M. Chopra, P. Ijumba, A. C. Tsai, F. Debra Jackson. The use of mobile phones as a data collection tool: A report from a household survey in South Africa. BMC Medical Informatics and Decision Making, 9:51, 2009.

G. M. Vazquez-Prokopec, U. Kitron, B. Montgomery, P. Horne, S. A. Ritchie. (2010) Quantifying the Spatial Dimension of Dengue Virus Epidemic Spread within a Tropical Urban Environment. PLoS Negl Trop Dis 4(12): e920. doi:10.1371/journal.pntd.0000920

Proactive Two Way Mobile Advertisement Using a Collaborative Client Server Architecture

Weimin Ding and Xiao Su
Department of Computer Engineering
San Jose State University
San Jose, CA 95192, USA
Email: weimin.ding@gmail.com, xiao.su@sjsu.edu

Abstract—more and more advertisements have been pushed into mobile devices by advertising providers. Advertisers, not the consumers, decide and control the content and delivery time of the advertisements. This one-way push-based architecture doesn't truly meet consumers' demands and can quickly develop resistance from consumers. In this paper, we propose new client-server based architecture to provide proactive mobile advertisements in mobile devices. With the proposed solution, application developers can seamlessly integrate advertisements into their applications, based on users' behavior and preferences. Users will not be interrupted by the pushed advertisement and can stay more focused with their main tasks. We will present our detailed design, implementation, and evaluation of this proactive solution.

Keywords – mobile, advertisement, REST, content-driven, proactive advertisement

I. INTRODUCTION

Mobile marketing is booming. More Smartphone's shipped than PC's since 2010. With the increasing marketing of mobile devices, mobile advertising has become crucial for the current advertisement industry. Compared to the conventional TV and PC, the number of mobile handheld devices is twice as large. In addition, mobile devices have more personality characteristics because they always belong to specific owners. This will give the advertisement providers more room to trim their advertisements to fit different customers. According to Smaato White Paper [1], the mobile advertising market in US continues to expand every year; and it is predicted to reach over $5 billion in 2015.

Currently, advertising providers push content to mobile devices, regardless of users' interest and preferences. There are several problems with current push technology.

First, consumers are just passive receivers and pushed advertisements will not truly meet consumers' real demands. Consumers will quickly develop resistance to those pushed advertisements and reduce the advertisements hit rate. Second, the content of the advertisements and actual applications are separated and have no relationship. The pop-up advertisements are kind of the interruptions to users' normal applications. Most users will have antipathies to pop-up advertisements, so these advertisements actually bring opposite effect to consumers.

In this paper, we propose a client-server based solution of providing proactive advertisements in mobile devices. The client side includes content detection and collection of user behaviors to define and filter the accurate advertisements for specific users. The server side includes REST interface to communicate with clients and to provide dynamic contents for clients.

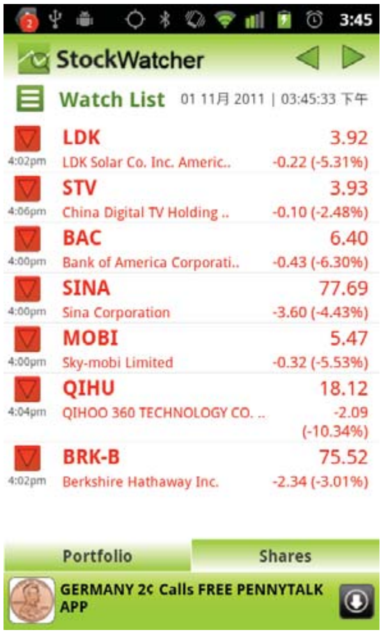
This solution can help application developers integrate advertisements into their applications seamlessly. Users will not feel the interference of the pushed advertisements, and the advertising content can be part of the application. With the rich content provided by the proactive advertisement server, developers can make more and more attractive applications.

In the rest of the paper, we will elaborate on the problems with push-based advertising and describe in detail the design, implementation, and evaluation of our proposed proactive solution.

II. PROBLEM OF PUSH-BASED ADVERTISING

Mobile advertising is a new area for the advertising industry and even the advertising giant such as Google is still exploring the correct model in this area. With the rapid growth of the market size, more and more vendors will involve in this area.

However, in current mobile advertisement solution, the need of customer is ignored and irrelevant content is pushed to customers. Let us see an example:


3:45

StockWatcher

Watch List 01 11月 2011 | 03:45:33 下午

LDK 3.92
4:02pm LDK Solar Co. Inc. Americ.. -0.22 (-5.31%)

STV 3.93
4:06pm China Digital TV Holding .. -0.10 (-2.48%)

BAC 6.40
4:00pm Bank of America Corporati.. -0.43 (-6.30%)

SINA 77.69
4:00pm Sina Corporation -3.60 (-4.43%)

MOBI 5.47
4:00pm Sky-mobi Limited -0.32 (-5.53%)

QIHU 18.12
4:04pm QIHOO 360 TECHNOLOGY CO. .. -2.09 (-10.34%)

BRK-B 75.52
4:02pm Berkshire Hathaway Inc. -2.34 (-3.01%)

Portfolio Shares

GERMANY 2¢ Calls FREE PENNYTALK APP

Figure 1. current mobile advertisement

First, we see that the banner advertisement has taken up a considerable amount of user's display screen: generally about 1/8 screen size is displaying the advertisement. In other words, if a user pays 500 dollars for the device, he/she pays about 60 dollars for the advertisement placeholder.

Second, we also observe that the advertisement is irrelevant from the content. Customers are watching stock quo. This advertisement offering cheap telephone cards may not motivate the mass untargeted customers to click the advertisement for more details. It is really a waste of resources for both advertisement publishers and consumers. Those advertisements are delivered to the wrong target. In this case, if it could display some special deals of stocker traders, the results would be better.

This current solution is one-way push. Consumers are always passive receivers. There is no way for them to publish their content into the network.

To resolve the above constraint of current mobile advertisements, our work adopts a new approach. Using a client/server architecture, consumers are able to publish mobile advertisements using its easy-to-use APIs and are able to enjoy better user experience by using mobile devices without intruding advertisements and by receiving only relevant and interesting advertisements based on their past interests and behaviors. Our design objectives include:

- No banner advertisement -- users will not be irritated by wasted screen
- Smart and Relevant Advertisements -- advertisements should be relevant of the content
- Interactive advertisement – consumers can be developers and publishers too.

III. System Arachitecture

A. system architecture

Our proposed system comprises of a client-server architecture. The client application resides in mobile devices and communicates with server using REST service [2].

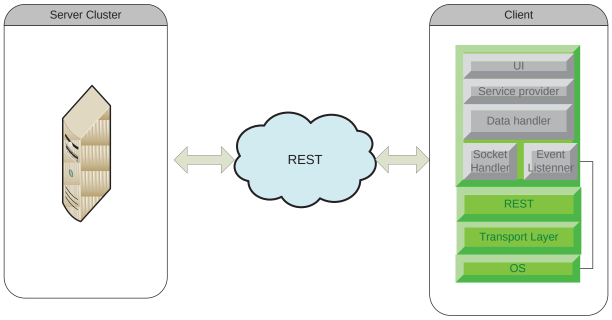


Figure 2. Proactive mobile advertisement

The client application pops up a message box to display incoming SMS and advertisement. The application doesn't modify or delete user's SMS message while parsing this message. Meanwhile, client application sends the content and receives advertisement response from server.

The client uses message queue to store both text SMS and advertisement messages, and uses a pop-up window to display the content from message. Message queue is a FIFO queue. After receiving the response from server, client displays the advertisements in the pop-up windows for specified timeout period, which is defined in the server response. User can finish reading by pressing cancel key in the device.

Upon receiving the request, the server analyzes the incoming content and finds helpful information. Based on this information, it queries the internal database with the content analysis results and location information. If more than one advertisements match in query, server will provide client the advertisement which has the highest PPC (Pay per Click).

B. *Protocol*

Server provides the restful API for invoking the request. This API can be enabled to support multiple client platforms.

CreateAd: to create an advertisement in server, the client sends a standard HTTP POST request to server AD URL. The following parameters have been added to the request message:

Post http://URL:Port/ads/

Attribute name	Type	Status
Category	String	optional
Vendor	String	mandatory
Content	String	mandatory
location	String	mandatory
Ppc	Integer	Mandatory

GetAd: to get a advertisement from server, client will send a standard HTTP GET request to AD URL followed with an advertisement ID.

Get http://URL:Port/ads/id

DeleteAd: to delete on advertisement in server, client will send a standard HTTP Delete request to specific AD URL.

Delete http://URL:Port/ads/id

UpdateAd: to update an advertisement in server, client will send a standard HTTP PUT request to specific AD URL . The following parameters have been added to the request message:

Put http://URL:Port/ads/id

Attribute name	Type	Status
Category	String	optional
Vendor	String	mandatory
Content	String	mandatory
location	String	mandatory
Ppc	Integer	Mandatory

GetBestAd: to get the best match advertisement form server, client will send a standard HTTP POST request to server AD URL. The following parameters have been added to the request message:

Post http://URL:Port/ads/

Attribute name	Type	Status
context	String	Mandatory
location	String	Optional

GetTopAd: to get TOP 10 advertisement in server, client will send a standard HTTP POST request to server Vendor URL. The following parameters have been added to the request message:

Post http://URL:Port/vendors/

GetMyAd: to get the own published advertisements from server, client will send a standard HTTP POST request to server Vendor URL with vendor name.

Post http://URL:Port/vendors/ vendorname

C. *System modules*

Our design has leveraged POS (*Part of Speech [3]*) tagger technology to realize content-driven advertisement pushing.

The central part of our design is implemented in MAD (Mobile Advertisement) server, which is responsible for providing the mad service to clients in different platform.

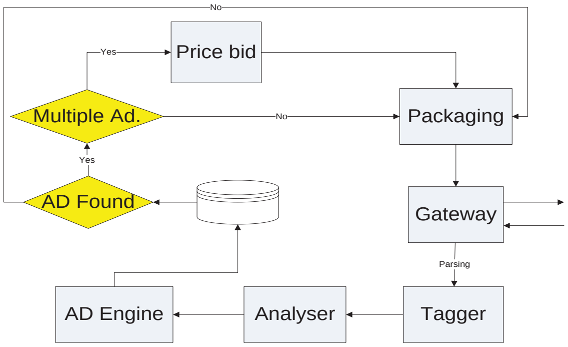


Figure 3. server work flow

Let us explain different system components (illustrated in Figure 3) in server work flow.

Gateway/Packaging takes care of web service handling and parses the request data to next module and it also sends out the results to client application.

POS Tagger reads the incoming context and gives the tag for each word. The tagged content will be sent to Analyzer to find the real meaning.

Analyzer finds useful keywords according to the tagged content. It outputs this content to Ad engine to help it to find matched advertisements.

Ad Engine queries the database according to the request data from Analyzer and prepares the advertisement content from the queue results. Ad Engine are in charge of creating, updating, reading, and deleting advertisement from database.

Database stores the advertisement content.

Figure 4. Shows the work flow in the client application.

Client uses message queue to store both text SMS and advertisement messages, and uses pop-up windows to display the content from the FIFO-based message queue.

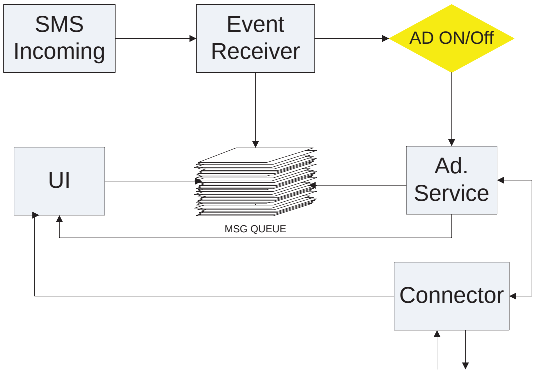


Figure 4. client work flow

UI will present the content of incoming SMS and advertisement to end users. Users can configure the display timeout and enable/disable the advertisement filter through the application menu. This module has one message queue to store incoming SMS and advertisement content. UI displays the top listed advertisement in the system and provides the interface for end users to input and post their advertisements.

Event Receiver will listen the SMS and system event and trigger the ad service wake up.

Connector handles the incoming/outgoing traffic to server. It packs and unpacks data and queues messages.

AD service prepares the request data to server, which filters the keywords in the incoming SMS and collects the location information.

IV. System Implementation

We have implemented our proposed solution in three tiers: front-tier, middle-tier, and data tier.

A. *Front-Tier Implementation*

The client uses Android as the target platform, and it inherits Activity, Service and Broadcast receivers from Android SDK. Next we present our front-tier implementation

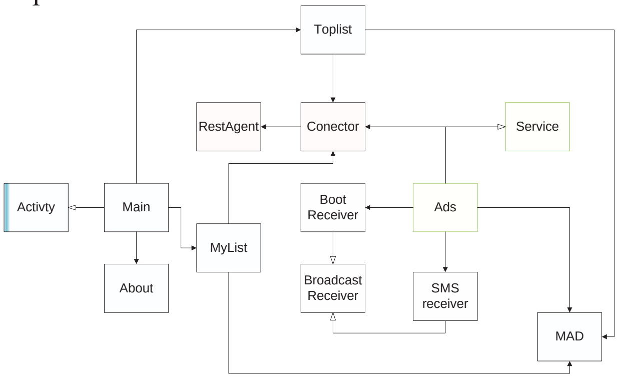


Figure 5. Front-Tier Overview

a) Main

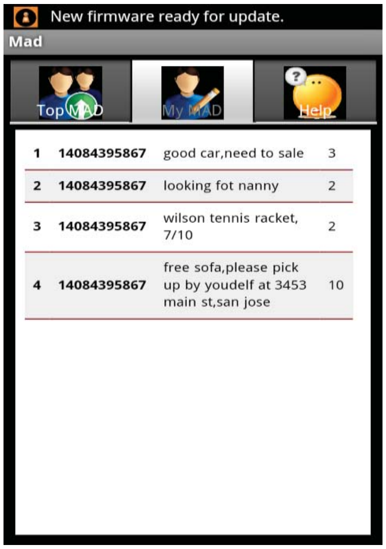
The main activity will be launched when the user starts the application.

b) Top list

It displays top 10 advertisements counted by the display times in history. Users can display the related mobile advertisements by clicking the vendor's name. The specific advertisement will be displayed if the user clicks it from the list.

c) My list

It displays the advertisement published by the owner. For example, in this screen shown in Figure 6, the user publishes four mobile advertisements by its device (14084395867). Users can display their own advertisements by clicking them.



New firmware ready for update.

Mad

1	14084395867	good car,need to sale	3
2	14084395867	looking fot nanny	2
3	14084395867	wilson tennis racket, 7/10	2
4	14084395867	free sofa,please pick up by youdelf at 3453 main st,san jose	10

Figure 6. User Posted Advertisement

Users can also publish advertisements from the devices directly. The vendor's name will be connected to its phone number, and the user can change it before submission. Users have the option to choose publishing free advertisements or paid advertisements.

d) Mad

It displays retrieved mobile advertisements and incoming SMS. Users can switch between SMS and advertisements. MAD will be triggered while

- Users are clicking the advertisement in Top list or My list
- SMS messages are arrived
- Advertisements are retrieved

Besides the activity we demonstrated, the client also implements a service called Ads in Device. This service can be started manually by users or automatically when the application starts or the device reboots. After Ad received mobile advertisements from server, it will relay this advertisement to MAD activity. If more than 10 seconds have passed, it relays the advertisement immediately; otherwise, it waits until the user finishes reading SMS in 15 seconds. Client uses Connector and RestAgent to communicate with server. The Restful action is performed by that class.

B. Middle-Tier Implementation

To get the hidden meaning of the context, Mid-Tier application implements POS tagger technology. It tags the context and retrieves meaning words.

Based on the frequency and position of the meaningful words, server start queries to Mongo database. The results might include more than one responses. In that case, the top one will be decide by pay per click value defined by advertisement provider. Server uses MongoMapper as Object-relational mapping to communicate with data-tier.

C. Data-Tier Implementation

Data-Tier is implemented by Mongo database, which is a document based, schema-free, scalable and high performance database.

A document-based data model can hold rich data structure from simple JSON to complex array, so the administrator does not need to hold the complex content within multiple relation tables.

The advantage of this schema makes it easy for the administrator to dump multiple formats of the advertisements into the system. Also it can keep the data backward compatible.

As the database will be expanded when the system gains more and more users and advertisements, predictable scalability is important to the system.

Performance is also a concern when choosing the database. The high performance brought by Mongo DB meets the performance requirements of the system.

V. EXPERIMENTAL RESULTS

A. Experimental Setup

To measure the performance of the system, we designed a test to check system capability of handling different contents. The test setup includes a computer running MAD server and Mongo DB and an Android device running MAD client. The computer is a 2.53 GHz dual core machine with 3GB memory. The Android device is equipped with a 1GHz CPU and 512 MB memory.

B. Performance Results

In the experiments, we compared the elapsed times while MAD system handles different sizes of SMS. A SMS will be sent to the device, which has installed the MAD client. The client will send request to the server. On server side, it analyzes the incoming request and responds with the correct advertisement based on the contents. The elapsed times are monitored and captured to measure the efficiency of the system. DB elapsed time presents the duration that it takes for server to render the advertisement after querying database. E2E time represents the duration between device sending request and receiving response. Also we record the returned vendor for such SMS test campaign to verify the quality of returned advertisement

TABLE I. TEST MATRIX

Content	Words	Vendor	E2E Time (ms)
Lunch	5	One local restaurant	1273
Let us watch movie tonight	25	AMC	1570
Would you please bring the book to me? I am in library.	53	Amazon	1794
The recipe of pumpkin pie: 2 spoon of salt, 3 pound of pumpkin, 1 pint of sugar. blend them and	104	Safeway	1912

bake in 450F			
Debates concerning the nature, essence and the mode of existence of space date back to antiquity; namely, to treatises like the Timaeus of Plat o, or Socrates in his	140	Santa Clara Library	2178

We evaluated their performances about five different SMS si zes: 5, 25, 53,104 and 140 with different contents. As you can see the table, when the SMS size is increasing, the tim e taken to get advertisements is becom ing longer. The behavior is expected as more time will be consum ed by the transmission and content analyzing.

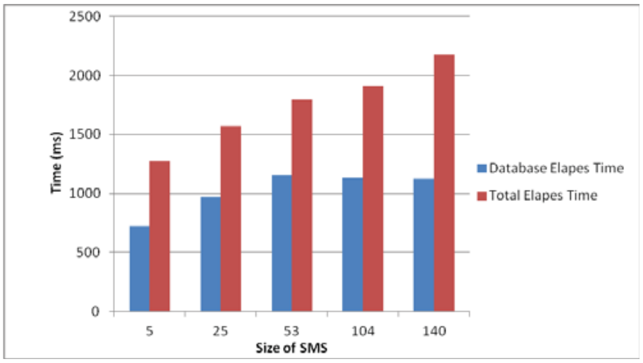


Figure 7. Database Elapse Time/Total Elapes Time

VI. FUTURE ENHANCEMENTS

In the future, we plan to include location information in the analy sis and preparat ion of advertisements. When a consum er goes to a targ et territory, relevant advertisements should be delivered to the consumer, considering his location. Com bining location information, users will receive location-aware and content-aware advertisements tailored to their interests.

I. CONCLUSION

Our proactive mobile advertisement system provides convenient restful A PI for developers to embed mobile advertisements into the application. This system conquers the problem s that happened in existing mobile advertisements solutions. It uses convenient RESTful API and avoids the banner display in mobile devices. Our work presents the initial step to chang e the current advertising model. We believe that in the future, g ood content-related advertisement providers should help advertisement publishers deliver the adv ertisements to targ et customers precisely and shoul d provide the friendly user interface to m otivate customers to involve into the advertisement system and to give them the opportunity to post and display advertisements.

GLOSSARY

[1] API: Application Interface which is a interface provide to develop to access the service provided by provider

[2] REST:Representational State Transfer is style of web service. Service consumer could access the service and resource by stand HTTP call

[3] POS Tagger: Part-of-speech tagging is a pr ocess to markup the word regards to the context

[4] PPC: Pay per click which define the bid pr ice by vendor for each clicking in their advertisement

REFERENCES

[1] White Paper on The USmA: The United States of Mobile Advertising retrieved by July 06,2011
http://www.smaato.com/media/Smaato_WhitePaper_USma_11022010.pdf

[2] RESTful Service retrieved by July 10,2011
http://en.wikipedia.org/wiki/Representational_state_transfer#RESTful_web_services

[3] Mobile Marketing Association Mobile Advertising Guidelines(Version 5.0) retrieved July 06, 2011, from http://www.mmaglobal.com/mobileadvertising.pdf

[4] Silicon Angle 2009 Mobile Advertising revenue market share leader
Retrieved July 07, 2011
from http://siliconangle.com/blog/2009/12/11/2009-mobile-advertising-revenue-market-share-leaders/

[5] Google Inc. Admob API Document

[6] Google Inc. Google Adsense API

The COIN Platform: Supporting the Marine Shipping Industrial Sector

Achilleas P. Achilleos, Georgia M. Kapitsaki, George Sielis, and George A. Papadopoulos
Department of Computer Science, University of Cyprus, Nicosia, Cyprus
E-mail: {achilleas, gkapi, sielis, george}@cs.ucy.ac.cy

Abstract – The COIN (Enterprise COllaboration and INteroperability) platform allows exposure, integration and application of interoperability and collaborative services in various business domains. As part of the COIN FP7 project, the objective was to exploit the COIN platform and apply its services in the marine shipping sector. In this demo, we demonstrate how COIN services are used to expedite and simplify business processes in the marine shipping domain. The demo showcases the execution of the two marine business processes, which are implemented using the COIN service platform. On the basis of the case studies, the necessary results are acquired based on the feedback received from marine experts. This reveals the positive impact of the COIN platform, in terms of reducing the time required to execute marine processes.

Keywords - Web-based Enterprise Systems; Web-based Business Processes; Collaboration and Interoperability

I. Introduction

The business use cases studied and developed in the form of COIN pilots in volve the marine shipping domain in Cyprus. The first use case refers to the "Negotiations between UPT and charterers f or the voyage's pre-fixture queries", which produces the "Recap" document as an outcome of negotiations and logical amendments for the vessel's voyage. The second scenario refers the creation of the Proforma Disbursement Account (PDA). The PDA details all estimated port costs that the port agent will have to pay in order for the vessel to have a smooth voyage.

The platform forms the backbone, integrating Web Services for Enterprise Collaboration (EC) and Enterprise Interoperability (EI) in various business domains [1]. It fulfils the COIN vision of providing a perv asive service platform to h ost Baseline and Innovative Web Services for EI and EC, which can be used by European enterprises for running their business in a secure, reliable and efficient way. The COIN platform is develo ped ontop of Liferay, which is an enterprise, web-based portal f or building technology-oriented, business applications that deliver immediate results and lon g-term value. Using the COIN platform we have implemented the marine pilots using the ProcessMaker application (i.e. bu siness process management and workflow software) offered by the platform. Through this ap plication we can invoke the necessary COIN services that allow executing the required business tasks. In this demo we will allow conference participants to execute the pilots via a Web-Based portal.

Table 1 shows the COIN EC/EI services, which follow the notion of "Software as a Service" (SaaS) [2], used for the pilots implementation. The following services [3] were deemed essential based on the initial requirement analysis performed with our industrial partners: (1) Collaboration Visualization Tool (CVT) – Formulation and visualization of human collaboration networks, including users and their discovered relations (e.g. joint activities), (2) Trusted Information Sharing (TIS) – Flexible sharing of business related information (e.g. documents) on the basis of CVT relations, (3) Interoperability Space Service (ISS) – A negotiation tool f or exchanging and n egotiating business documents in standardized UBL format and (4) Baseline Communication Services – A suite of services that include Skype call, instant messaging and notification.

Table 1: Developed workflows, EC/EI Services and successful executions

	EC/EI Services Used	Workflow Executions	Successful Executions
BUC1	CVT, TIS, ISS, Communication	14	11
BUC2	CVT, ISS, Communication	11	10

The execution of the pilots was carried out initially by the developers at the University of Cyprus (UCY). Actors from our industrial partner Donnelly Tanker Management (DTM) were also involved. At this stage DTM employees were acquainted with the functionalities offered by the platform and the EC/EI services used in the developed workflows. Developers trained DTM actors h ow to u se the service platform to execute the workflows. Next, they were involved in solo execu tion and evaluation of the workflows. Finally, a m eeting was setup at DT M offices where DTM employees engaged in the pilots execution and provided the needed feedback. Developers undertook the roles of the remaining parties.

The initial training of DTM employees by developers was sufficient for successful executions of the use cases. Failures were recorded in the first executions, since an initial learning curve was essential to avoid errors and omissions. The workflows design foreseen in improving two factors: reduce execution time and achieve efficient voyages management by a single operator. Currently, the communication methods used are e- mails and telep hone.

This makes the process time consuming. An operator is also responsible to operate more than one voyage per day. Therefore, an operator needs to be aware of several recap documents, forward each recap to the right captain and at the same time be able to manage each trip by following the recap instructions that corresponds to a voyage. Hence, the use of structured workflows will aid the operator in managing several voyages and executing business processes more efficiently.

II. THE COIN EXPERIENCE

The test-bed environment was prepared and executed at DTM offices, in Limassol, Cyprus. Figure 1 presents a small part of the second modeled workflow; due to space limitations. It includes the necessary tasks executed for the formulation and the distribution of the PDA document. The preliminary data is recorded into the initial PDA, as a direct result of the negotiations between DTM and the selected port agent. The PDA is then distributed to the captain and the DTM accounts department. Then, the accounts department is responsible to handle the payment of the appointed agent's fee. The captain and the agent are then in contact and follow the PDA terms to manage and execute the voyage. The vessel captain is also responsible to report daily to DTM. This will lead to the successful execution of the voyage and delivery of goods from the loading to the destination port without any predicaments.

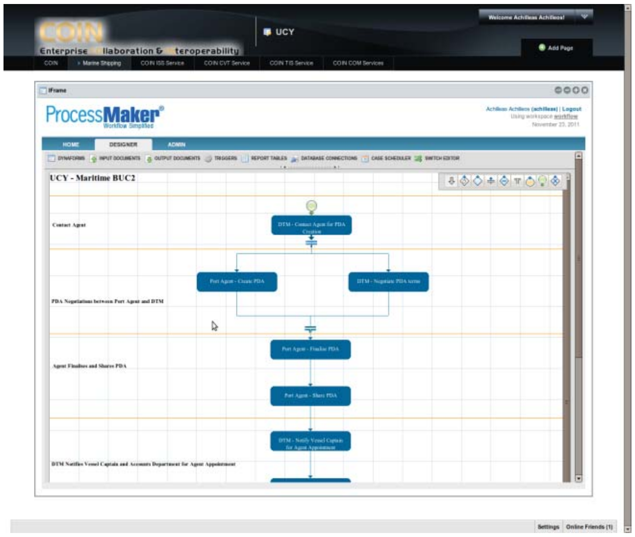
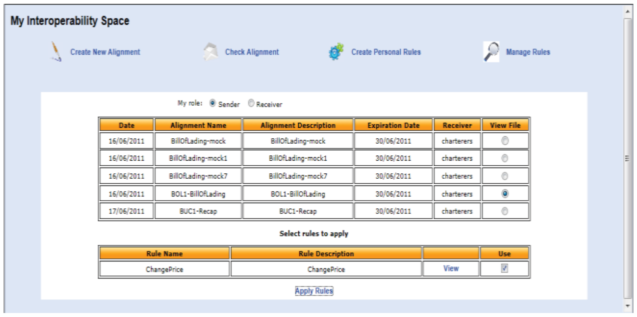


Figure 1: Part of the workflow model of the business use case for the production of the Recap document

A single task is presented, to showcase the changes performed in the marine processes. In this task the agent should create the PDA document that contains the voyage details and terms (e.g. agent fee) under negotiation (Figure 2). Currently, numerous documents exist in accordance to the agent's company. These hard copies include the same data but in different formats. Hence, the agent completes the form and sends it by fax to the DTM operator. In parallel, they exchange emails or phone calls to inform on the changes and negotiate the terms of agreement. With the structured workflow the data is formalized and defined in an electronic PDA form. The agent completes the form and submits it. Upon submission a transformation script is executed. Thus, the UBL format (i.e. XML-based) of the PDA is automatically generated and uploaded into the COIN ISS to kick-off the negotiations.


My Interoperability Space

My role: ● Sender ○ Receiver

Date	Alignment Name	Alignment Description	Expiration Date	Receiver	View File
16/06/2011	BillOfLading-mock	BillOfLading-mock	30/06/2011	charterers	○
16/06/2011	BillOfLading-mock1	BillOfLading-mock1	30/06/2011	charterers	○
16/06/2011	BillOfLading-mock7	BillOfLading-mock7	30/06/2011	charterers	○
16/06/2011	BOL1-BillOfLading	BOL1-BillOfLading	30/06/2011	charterers	●
17/06/2011	BUC1-Recap	BUC1-Recap	30/06/2011	charterers	○

Select rules to apply

Rule Name	Rule Description		Use
ChangePrice	ChangePrice	View	☑

Apply Rules

Figure 2. Using ISS for contacting the negotiations

III. LESSONS LEARNED & BEST PRACTICES

The COIN platform has provided benefits in terms of decreasing the development time to implement the pilots and in terms of reducing the time required for executing the business use cases. This efficiency improvement was a result recorded by the DTM actors during the execution of real-life business workflows as shown in Table 1. An important factor is that pilots development was impeded less by developers and was driven mostly by business requirements. The initial learning curve required 3-4 executions assisted by developers, prior to the business partners being able to engage on their own in the successful execution of the workflows.

The pilots' execution via a Web-Based portal aided the partners to familiarize and navigate easily the assigned tasks. The workflow management tool allowed having a clear view of pending tasks. This was very critical for our partners since it provided structure, control and facilitated coordination of the use cases; specifically for the operator. It also allowed DTM managers to view the progress of the workflow to monitor the tasks and be aware of the voyage status. This was missing since ad-hoc procedures did not provide any structure. Thus, it was very difficult to have easy control, coordination, supervision and monitoring of the processes. In overall, the efficiency of the processes as stated by our business partners was improved through the offered collaboration and interoperability services.

REFERENCES

[1] E. D. Grosso, S. Gusmeroli, A. Olmo, A. Garcia, D. Busen, G. Trebec, "Are Enterprise Collaboration and Enterprise Interoperability enabling Innovation scenarios in industry? The COIN IP perspective in Automotive", eChallenges 2010.

[2] P. Buxmann, T. Hess and S. Lehmann, "Software as a Service", WIRTSCHAFTSINFORMATIK, Volume 50, Number 6, 500-503, DOI: 10.1007/s11576-008-0095-0.

[3] Florian Skopik, Daniel Schall, Schahram Dustdar, "Trust-based Adaptation in Complex Service-oriented Systems", IEEE Conference on Engineering of Complex Computer Systems (ICECCS), 2010.

A proposal for the improvement of the technique of EVM utilizing the history of performance data

Adler Diniz de Souza
Universidade Federal do Rio de Janeiro
Av. Horácio Macedo, 2030, Building of the Technical
Center, Block H, Suite 319, P.O. Box 68511 – Postal
Code 21941-914 – Rio de Janeiro, RJ
adlerunifei@gmail.com

Ana Regina Cavalcanti Rocha
Universidade Federal do Rio de Janeiro
Av. Horácio Macedo, 2030, Building of the Technical
Center, Block H, Suite 319, P.O. Box 68511 – Postal
Code 21941-914 – Rio de Janeiro, RJ
darocha@centroin.com.br

Abstract - **The present study proposes an extension of the technique Earned Value Management – EVM, through the integration of the history of performance data as means of improving the technique's cost predicability. The proposed technique is evaluated and compared to the traditional technique through different hypothesis tests, utilizing data from the simulation projects. The tecnique showed to be more accurate than the traditional one for the calculation of the CPI.**

Keywords: Earned Value Management, Cost Peformance Index – CPI, project management, measurement and analisys

I. Introduction

To assess whether or not a project will reach its goals of time and cost, several measures are collected during its execution, and various performance indicators are produced and periodically analyzed. When there are deviations larger than the tolerance in some performance indicators, corrective actions are undertaken in order to improve them. Among the main available techniques for the analysis of cost and time - EVM, is considered the most reliable [4].

Several formulas derived from EVM measurements are available and have been studied in the last 15 years. However, studies intended to improve the predictability of the results of time and cost have remained stagnant over the last decade and still require further studies [4].

II. Earned Value management - EVM

The method of EVM allows the calculation of variances and performance indices of cost and time, which generate forecasts for the project, given its performance so far, allowing the implementation of actions aimed at correcting any deviations [1]. This allows the project's manager and your team to adjust their strategies, make trade-offs based on the goals, on the project's the current performance, on trends, and on the environment in which the project is being conducted [1].

The method of EVM is based on three basic measures (Planned Value – PV, Earned Value – EV and Actual Cost - AC), which are derived to generate other measures and performance indicators, , i.e. Cost Performance Index – CPI.

III. Problem description

The application and reliability of the CPI for the realization of projections have been widely discussed as shown by the works of [1], [4] and [2].

The discussions focus on the CPI's stability. The statement of the CPI's stability is important, as it is used to make cost projections. According to [1], the CPI is considered stable if there is a variation of plus or minus 10% of the value reached when the project has been 20% executed.

A study performed in [1] evaluated the stability of the CPI of various projects of the U. S. DoD, and found that there was stability of the indicator, after 20% of project execution. This study generalized the result, stating that any project could use the technique reliably after 20% of execution.

However, several other studies have questioned the generalization of these results in different contexts that showed different results, i.e., the CPI showed instability during much of the project, [4], [2].

IV. Proposal of EVM History

The present study proposes the integration of the EVM with the history of cost performance data, which consists in the collection and utilization of a set of data of each one of the processes, such as: i) history of process performance; and ii) weight (% of cost) of each process utilized in the project, as in the equation below:

$$CPI_{Est} = (CPI_{AcumN\%}*Weight_{Acum} + CPI_{P1}*Weight_{P1} + CPI_{HistP2}*Weight_{P2} + CPI_{PN}*Weight_{PN}...) \quad (1)$$

Where:

- **CPI_{Est}:** is the estimated CPI that will be used for projections;
- **$CPI_{AcumN\%}$:** is the CPI that will be calculated normally, with the traditional equation of the EVM (EV / AC), until the project's current date;
- **$Weight_{Acum}$:** is the percentage of the project already executed expressed in %.
- **Other CPI's and Weights:** are the history of CPI's (average) of each process utilized in the project and how much the weight of these processes represent in the total cost of the current project.

V. Planning of the study

The study's objective was to answer the following question: "*Is the EVM technique with the history of performance more accurate than the traditional EVM technique?*"

Thus, the following hypotheses were set up to evaluate the accuracy of the techniques:

- **$\mathbf{H0}_{\mathbf{Accuracy}}$**: the traditional EVM technique is as accurate as the EVM technique with the history of performance ($\text{Error}_{\text{EVM}} - \text{Error}_{\text{EVMH.}} = 0$).
- **$\mathbf{H1}_{\mathbf{Accuracy}}$**: the traditional EVM technique is less accurate than the EVM technique with the history of performance ($\text{Error}_{\text{EVM}} - \text{Error}_{\text{EVMH.}} > 0$).

Thus, in order to measure the technique's accuracy, each technique's CPI was compared at three distinct times in the project, at the beginning, (25% of execution), at the middle, (50% of execution) and at the end, (75% of execution), with the real CPI calculated at the end of the project, through a feasibility study.

One of the main difficulties presented by [5] and [3] in studies related to this one, was the lack of project performance data, available for studies. Thus, it was decided to validate the proposed technique through the performance of project simulations, similarly to the studies conducted by [3].

VI. Feasibility Study and Discussion

The largest cost component in a software project are the man-hours necessary for product development, all the necessary simulations required for the calculation of the base measures and indicators of traditional EVM were based on the planned effort and actual effort for a set of activities of possible processes of any given project, whereby these activities were calculated using the random() function of the MSExcel tool, as shown in Table 1.

Table 1 – Values passed to the random function for simulation

Process	Variation H_{Est}	Variation H_{Real}	No.of activities
Process 01	8 – 30	3 – 40	12
Process 02	3 – 10	3 – 10	26
Process 03	3 – 12	1 – 12	26
Process 04	3 – 17	1 – 25	26

It was assumed, as premises for the generation of the CPI of the proposed technique that: i) the history data of all previously executed projects utilizing this process were available and ii) the processes were stable, however they showed great variability. The simulations showed that the proposed technique showed better accuracy of cost estimates than the traditional technique, as shown by Figure 1.

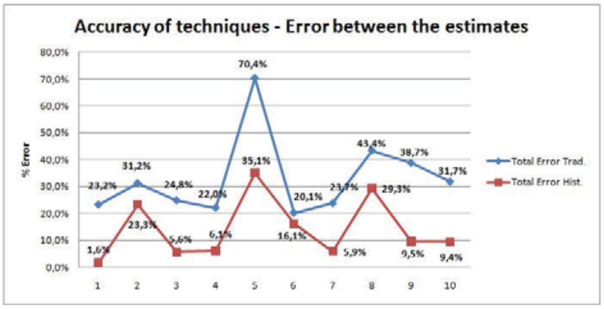
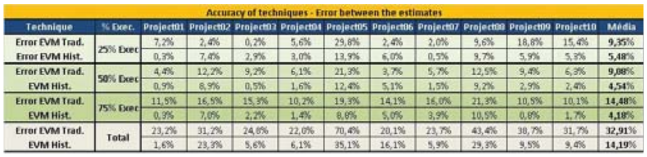


Figure 1 – Accuracy – Total CPI error between the techniques

Figure 1 shows that the gain in accuracy using the proposed technique was at times 15 times better than the traditional technique, in the context of the simulation.

Statistical tests were performed based on Table 2 to confirm whether the differences in accuracy found in the application of the techniques were significant, thereby approving or rejecting the previously presented hypotheses. The Action tool was used to test the hypotheses of T paired samples, at the 99% significance level.

Table 2 – Accuracy (error between the estimates) of the CPI



Accuracy of techniques - Error between the estimates

Technique	% Exec.	Project01	Project02	Project03	Project04	Project05	Project06	Project07	Project08	Project09	Project10	Média
Error EVM Trad.	25% Exec	7,2%	2,4%	0,2%	5,6%	29,8%	2,4%	2,0%	9,6%	18,8%	15,4%	**9,35%**
Error EVM Hist.		0,3%	7,4%	2,9%	3,0%	13,9%	6,0%	0,5%	9,7%	5,9%	5,3%	**5,48%**
Error EVM Trad.	50% Exec	4,4%	12,2%	9,2%	6,1%	21,3%	3,7%	5,7%	12,5%	9,4%	6,3%	**9,08%**
EVM Hist.		0,9%	8,9%	0,5%	1,6%	12,4%	5,1%	1,5%	9,2%	2,9%	2,4%	**4,54%**
Error EVM Trad.	75% Exec	11,5%	16,5%	15,3%	10,2%	19,3%	14,1%	16,0%	21,3%	10,5%	10,1%	**14,48%**
EVM Hist.		0,9%	7,0%	2,2%	1,4%	8,8%	5,0%	3,9%	10,5%	0,8%	1,7%	**4,18%**
Error EVM Trad.	Total	23,2%	31,2%	24,8%	22,0%	70,4%	20,1%	23,7%	43,4%	38,7%	31,7%	**32,91%**
EVM Hist.		1,6%	23,3%	5,6%	6,1%	35,1%	16,1%	5,9%	29,3%	9,5%	9,4%	**14,19%**

The analysis of the data in Table 2 and Table 3, allows to state, at the outset, that the proposed technique provides more accuracy in the cost estimates, when the project is at 50% and 75% of execution, and in general taking the three reported moments, at a 99% confidence level.

Table 3 - Tests of hypothesis of accuracy

Hypothesis	Test	T	*p*	Conclusion
$\mathbf{H0}_{\mathbf{Accuracy}}$	$\text{Error}_{\text{EVM}} - \text{Error}_{\text{EVMH.D.}} > 0$	3,59	0,00292	Refute H0
$\mathbf{H0}_{\mathbf{25\%\ Accuracy}}$	$\text{Error}_{\text{EVM25\%}} - \text{Error}_{\text{EVMH.D.25\%}} > 0$	2,21	0,0269	Accept H0
$\mathbf{H0}_{\mathbf{50\%\ Accuracy}}$	$\text{Error}_{\text{EVM50\%}} - \text{Error}_{\text{EVMH.D.50\%}} > 0$	4,809	0,00048	Refute H0
$\mathbf{H0}_{\mathbf{75\%\ Accuracy}}$	$\text{Error}_{\text{EVM75\%}} - \text{Error}_{\text{EVMH.D.75\%}} > 0$	21,271	2,63 x 10^{-9}	Refute H0

VII. References

[1] CHRISTENSEN, D., SCOTT R. HEISE, 1993, "Cost Performance Index Stability", National Contract Management Journal, v. 25, pp. 7-15.

[2] HENDERSON, K., OFER ZWIKAEL, 2008, "Does Project Performance Stability Exist A Re-examination of CPI and Evaluation of SPI(t) Stability, " Cross Talk, v. 21 (April, 2008), pp. 04-07.

[3] IMAN, A., OW SIEW HOCK, 2009, "Using Enhancement Method to Improve Earned Value Index to Achieve an Accurate Project Time and Cost Estimation", International Conference on Future Computer and Communication.

[4] LIPKE, W., 2006, "Statistical Methods Applied to EVM: The Next Frontier", Department of Defense - USA, v. 19, pp. 32, June.

[5] LIPKE, Z., ANBARI, HENDERSON, 2009, "Prediction of project outcome, the application of statistical methods to EVM and Earned Schedule performance indexes", International Journal of Project Management.

[6] PMI, 2009, Project Management Body of Knowledge - PMBOK Newton Square, Project Management Institute.

Checking Contracts for AOP using XPIDRs

Henrique Rebêlo, Ricardo Lima, Alexandre Mota, César Oliveria, and Márcio Ribeiro

Federal University of Pernambuco
Brazil

{hemr,rmfl,acm,calo,mmr3}@cin.ufpe.br

ABSTRACT

Over the last years, several proposals have advocated that a notion of interface between the base code and aspect code is necessary for reasoning about aspect-oriented programming (AOP), and for overcoming pointcut fragility. However, existing work that are AOP based, have not shown how one can specify these interfaces to facilitate modular reasoning and specify control effects, such when advice does not proceed.

In this demonstration, we show how our crosscut programming interfaces with design rules, or XPIDRs allow modular understanding and enforcement of control flow effects. The key idea behind our design methodology is to introduce a design phase for each crosscutting concern. Hence, a designer establishes a crosscutting design rule interface to decouple the base and the aspect design. Such a crosscutting design rule is based on the well-known crosscut programming interfaces (XPIs). The main difference is that we present XPIs with the notion of behavioral rules. We also show that since our approach with XPIDRs do not require any new AOP construct; they can be adopted in a straightforward manner by the AOP community.

Demonstration Overview

This demonstration showcases the features and benefits of the XPIDRs through the figure editor system , which is the classical example used in several AOP papers. The current infrastructure for developing XPIDRs is also demonstrated. It requires the AJDT development toolkit used by the Eclipse IDE. Hence, features like weaving and syntax highlighting are straightforward. In addition, there is an XPIDR library required to specify the design rules of the base and aspect code. This library includes all the JML annotations along with specific ones used to specify the control flow effects of the program.

Presenter Biography

Henrique Rebêlo is one of the authors of the XPIDRs. He has extensive experience in separation of concerns and design by contract techniques. He co-developed the aspect-oriented JML compiler known as ajmlc. This compiler uses AOP for enforcing JML contracts at runtime. He was a researcher intern in 2010 at Microsoft Research working on program analysis and program verification. He has given talks on design by contract and AOP at prestigious venues like SEKE'11, FTFJP'11, SAVCBS'09, ICST'08, and SAC'08.

Acknowledgments

This work has been partially supported by FACEPE under grant No. IBPG-1664-1.03/08 for Henrique Rebêlo. Ricardo Lima is also supported by CNPq under grant No. 314539/2009-3.

Author's Index

Reviewer's Index

A

Alain Abran
Silvia Teresita Acuna
Taiseera Albalushi
Edward Allen
Thomas Alspaugh

B

Doo-hwan Bae
Ebrahim Bagheri
Hamid Bagheri
Rami Bahsoon
Xiaoying Bai
Purushotham Bangalore
Ellen Francine Barbosa
Fevzi Belli
Ateet Bhalla
Swapan Bhattacharya
Alessandro Bianchi
Karun N. Biyani
Borzoo Bonakdarpour
Ivo Bukovsky

C

Kai-yuan Cai
Gerardo Canfora
Jaelson Castro
Raul Garcia Castro
Cagatay Catal
Peggy Cellier
Christine Chan
Keith Chan
Kuang-nan Chang
Ned Chapin
Shu-Ching Chen
Zhenyu Chen
Stelvio Cimato
Peter Clarke
Esteban Clua
Nelly Condori-fernandez
Fabio M. Costa
Maria Francesca Costabile
Karl Cox
Jose Luis Cuadrado
Juan J. Cuadrado-gallego

D

Ernesto Damiani
Dilma Da Silva
Jose Luis De La Vara
Marian Fernandez De Sevilla
Scott Dick
Massimiliano Di Penta
Jing Dong
Weichang Du
Philippe Dugerdil
Hector Duran

E

Christof Ebert
Ali Ebnenasir
Raimund Ege
Magdalini Eirinaki
Faezeh Ensan

F

Davide Falessi
Behrouz Far
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

Desmond Greer
Eric Gregoire
Christiane Gresse Von Wangenheim
Katarina Grolinger

H

Hao Han
Xudong He
Miguel Herranz
Mong Fong Horng
Shihong Huang

J

Clinton Jeffery
Jason Jung
Natalia Juristo

K

Selim Kalayci
Eric Kasten
Taghi Khoshgoftaar
Jun Kong
Nicholas Kraft
Anesh Krishna
Sandeep Kulkarni
Vinay Kulkarni
Gihwon Kwon

L

Jeff Lei
Bixin Li
Ming Li
Tao Li
Yuan-Fang Li
Qianhui Liang
Shih-hsi Liu
Xiaodong Liu
Yan Liu
Yi Liu
Hakim Lounis
Joan Lu

M

Jose Carlos Maldonado
Antonio Mana
Vijay Mann
Riccardo Martoglia
Hong Mei
Hsing Mei
Emilia Mendes
Ali Mili
Alok Mishra
Ana M. Moreno

N

Kia Ng
Ngoc Thanh Nguyen
Allen Nikora

O

Edson Oliveira Jr.

P

Kunal Patel
Xin Peng
Antonio Piccinno
Alfonso Pierantonio
Antonio Navidad Pineda

R

Rick Rabiser
Damith C. Rajapakse
Rajeev Raje
Jose Angel Ramos
Marek Reformat
Robert Reynolds
Ivan Rodero
Daniel Rodriguez

S

Samira Sadaoui
Masoud Sadjadi

Claudio Sant’Anna
Salvatore Alessandro Sarcia
Douglas Schmidt
Andreas Schoenberger
Naeem (jim) Seliya
Tony Shan
Rajan Shankaran
Michael Shin
Qinbao Song
George Spanoudakis
Jing Sun
Yanchun Sun
Gerson Sunye

T

Jeff Tian
Genny Tortora
Mark Trakhtenbrot
Peter Troeger
T.h. Tse

V

Giorgio Valle
Sylvain Vauttier
Silvia Vergilio
Akshat Verma
Sergiy Vilkomir
Arndt Von Staa

W

Huanjing Wang
Limin Wang
Hironori Washizaki
Victor Winter
Guido Wirtz
Eric Wong
Franz Wotawa

X

Dianxiang Xu
Haiping Xu

Y

Chi-lu Yang
Hongji Yang
Ji-Jian Yang
Junbeom Yoo
Huiqun Yu

Z

Cui Zhang
Du Zhang
Hongyu Zhang
Yong Zhang
Zhenyu Zhang
Hong Zhu
Xingquan Zhu
Eugenio Zimeo

Poster/Demo Presenter's Index

Program for the Twenty-Fourth International Conference on Software Engineering & Knowledge Engineering

SEKE

San Francisco Bay
July 1-3 2012

Copyright © 2012
Printed by
Knowledge Systems Institute
Graduate School
3420 Main Street
Skokie, Illinois 60076
(847) 679-3135
office@ksi.edu
www.ksi.edu
Printed in USA, 2012